Mapping Parallel Programs onto Parallel Systems with Torus and Mesh Based Communication Structures

Lixin Tao

A DISSERTATION

in

Computer and Information Sciences

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

1988

____
Supervisor of Dissertation

____
Graduate Group Chairperson

To my parents

Acknowledgements

I am grateful to my advisor Dr. Eva Ma for her persistent support and advice during my four year study at the University of Pennsylvania. My dissertation benefits greatly from her insights and demand for high quality research and presentation. I would also like to thank Dr. Michael Palis, Dr. Insup Lee, and Dr. I. Hal Sudborough for serving on my dissertation defense committee and providing many good comments and suggestions for my research.

I am indebted to Prof. Tong-han Chang for introducing me into computer science. Dr. Charles Culmer's suggestions on writing styles have been very helpful.

This work was supported in part by National Science Foundation grant DCR84-51408, IBM research grant, AT&T Information System research grant, National Science Foundation CER grant MCS82-19196, and Army Research Office grant DAAG-29-84-K-0061.

ABSTRACT

Mapping Parallel Programs onto Parallel Systems with Torus and Mesh Based Communication Structures

Lixin Tao
Supervisor: Dr. Eva Ma

The major objectives of this research are (1) to design efficient schemes for mapping parallel programs onto parallel processing systems to minimize the communication overhead incurred by the mismatch between the communication characteristics of the parallel programs and those of the parallel processing systems, and (2) to support logical inter-process communication at execution time to improve program readability, verifiability, productivity, and portability.

We use graph mapping as the mathematical model of the program mapping problem. We introduce a rich class of low dilation cost graph embedding functions for toruses and meshes of various dimensions and various shapes (with lines, rings, and hypercubes as special cases). We design contraction functions to generalize the one-to-one embeddings to achieve optimal or good many-to-one graph mappings.

We propose an efficient program loading approach based on inverses of mapping functions and a broadcast network. We design the *shortest-path data routing scheme* to carry out automatically our data routing strategies at execution time to simulate on the system any permutation type set or scatter type set of parallel neighboring communications in the task graph. For most of our mapping functions, the data routing complexities are the same as the corresponding dilation costs. For the rest, the data routing complexities are less than four times the corresponding dilation costs. Since our approach supports task graph level communication at execution time, even the object code of parallel programs can be completely transparent to system topologies.

Contents

List of Figures

Chapter 1
Introduction

1.1 Introduction

The history of computer industry displays a continuous effort to increase the computational speed of computer systems. But not even the state-of-the-art in VLSI technology can fully satisfy the ever-growing computational demands from diverse fields such as artificial intelligence, image processing, robot vision, and real-time process control. Now it is clear that we cannot depend solely on the improvement of devices to enhance system performance. Only parallel processing, the technique of utilizing the processing power of multiple processors, can satisfy the requirements of future applications.

Although parallel processing is not a new concept in computer science, its deviation from the traditional von Neumann computation model has introduced many new problems. The extra complexity required for data communications among the processors can degrade system performance, and also make programming on a parallel processing system much harder than on a uniprocessor system. If each of the processors works autonomously, the synchronization among different processes will further increase the complexity of the system. Unless we have a clearer understanding of these problems and effective tools to attack these complexities, the full power of parallel processing cannot be exploited.

This research is aimed at investigating the communication problems in parallel processing systems. Our research scope will be restricted to parallel processing systems for supporting *data independent algorithms* (algorithms for which the communication patterns are independent of input data). Data independent algorithms are common in fields such as image processing, robot vision, and scientific computation, which are our current major application domains.

1.2 Research Objectives

Most of the communication problems in parallel processing systems come from the fundamentally different approaches adopted by uniprocessor systems and parallel processing systems to support inter-process communication. In a uniprocessor system, all processes reside in a single processor, and all inter-process communications are supported by main memory references. As a result, any process can easily send a message to any other process with a uniform delay determined principally by the main memory clock cycle. On the other hand, in a parallel processing system, different processes usually reside in different processors, and most inter-process communications are supported by an *interconnection network.* The delay incurred in an interconnection network is much greater than that introduced by the main memory references, and may be dependent on system size (number of processors) and the communication patterns of the parallel programs. We call the extra inter-processor communication time in a parallel processing system the *communication overhead.*

There are three main sources of the extra communication overhead of a parallel processing system: (1) the time for the messages to pass through long communication links; (2) the time for the messages to go through one or more intermediate processors in the absence of a direct communication link between the two processors accommodating the two communicating processes; (3) the contention for a single physical link by more than one message at the same time. While the delay introduced by the first source is mainly determined by system size, silicon chip layout, and package design, the delays introduced by the other two sources result from the mismatch of the communication characteristics of the parallel programs and those of the parallel processing systems. It is one of our main objectives in this research to minimize the extra communication overhead introduced by the last two sources.

A parallel program usually consists of a series of *computation phases,* each of which has a particular communication pattern. Thus, the communication characteristics of a parallel program can be represented by a series of *task graphs,* with one such graph for each phase, in which each node represents a process and each edge represents a possible communication requirement. For simplicity, we assume that a program consists of one computation phase unless stated otherwise.

Similarly, we can view a parallel processing system as a *system graph* in which each node represents a processor and each edge represents a link. If the task graph of a program and the system graph of a system have the same topology, then the program can be executed on the system efficiently since each communication reqirement can be satisfied directly by a single corresponding link. Otherwise, some messages have to go through several intermediate processors before they can reach their destinations, and more than one message could contend for a single link. As a result, the system performance in supporting the program could be degraded. Since programs can assume an infinite number of different topologies, and a system can usually assume only one topology, a mismatch of the communication characteristics between the programs and the system is the usual situation.

To minimize the communication overhead, we have to find an efficient way to map parallel programs onto parallel systems. We call this the *program mapping problem.* The program mapping problem consists of two parts: (1) how to allocate the parallel processes to the processors and how to load the codes for these different processes into the allocated processors, and (2) how to implement the parallel inter-process communications under a particular mapping scheme. A good solution to the program mapping problem should minimize the overall communication overhead.

In addition to the communication overhead problem, inter-processor communication in parallel processing systems also introduces problems concerning programming. In practice, *inter-process* communication in parallel programs is usually expressed in the form of *inter-processor* communication. Parallel programs including inter-processor communication must be tailored to system topology. As a result, programs designed for one system cannot be executed on another system with a different topology. In addition, since the communication in these programs is specified on the low physical system level, it is difficult to separate computation from low-level data routing. As a result, the readability, verifiability, and productivity of these programs are degraded. In this research we advocate and support the design of parallel programs that are independent of system topology to avoid the problems above.

The following are the two major objectives of this research:

1. Design efficient schemes to map parallel programs onto parallel processing systems to minimize the communication overhead incurred by the mismatch of the communication characteristics of the parallel programs and those of the parallel processing systems.

2. Support logical inter-process communication *at execution time* to improve program readability, verifiability, productivity, and portability.

In this thesis, we study only the program mapping problem in which the task graph and the system graph are both toruses or meshes. Many parallel processing systems use variants of these graphs as their system graphs [Sny82, Pot83, PBe85, LM87b, LM87a, Hil85, Oru84, KWA82, PV79], and many algorithms in image processing, robot vision, and scientific computation have these structures as their task graphs [Fox83, HKS*83, RK82, BB82].

1.3 Issues Studied in This Dissertation

To achieve our research objectives, in this dissertation we address the following issues.

Graph mappings among toruses and meshes

We use graph mapping to model the program mapping problem. We use task graphs to represent the communication characteristics of the parallel programs, and system graphs to represent the parallel processing systems.

In Chapter 3, we study bijective graph mappings, or *embeddings.* The guest graph and the host graph of an embedding have the same size. We use *dilation cost,* which is defined to be the maximum distance in the host graph between the images of any pair of neighboring nodes in the guest graph, as our optimization measure. Dilation cost models the maximum length of the shortest communication paths between neighboring processes after the parallel program is mapped onto the parallel processing system. Although graph embedding in general is an NP-complete problem, we show that if we restrict the problem domain to toruses and meshes, many embeddings can be performed with optimal dilation costs by simple embedding functions.

In Chapter 5, we study many-to-one graph mappings. In a many-to-one graph mapping, the guest graph has more nodes than the host graph. We define *node evenness* of a many-to-one graph mapping to be the maximum ratio of the numbers of nodes in the guest graph mapped into any two nodes in the host graph. We use dilation cost and node evenness as our optimization measures. In the corresponding program mapping problem, all of the processes mapped into the same processor have to be executed sequentially. Node evenness models the degree of balance of the number of processes mapped into each processor. An even distribution of the processes over the processors minimizes the computation time. We show that for toruses and meshes, we can achieve many optimal many-to-one graph mappings by first contracting the guest graph into an intermediate graph of the same size as the host graph, and then embedding the intermediate graph into the host graph with one of our bijective embedding functions.

The graph mapping results that we derive provide the mathematical framework for our solutions to the program mapping problem as well as a broad range of application problems such as finding storage representations for data structures [DEL78a, LED76, Ros79, RS78] and finding efficient layouts of circuits on chips [LR82, Ros83, Tho79, Val81].

Mapping parallel programs onto parallel processing systems

This mapping problem can be viewed as a "real world version" of the graph mapping problem. There are two major problems to be tackled here. The first is how to load in an efficient way the code for each process into the corresponding processor under a particular mapping scheme. This problem corresponds to the mapping of the nodes in a guest graph into the nodes in a host graph. To achieve this loading, we have to compute for each process the address of its physical destination processor. For a large parallel program and a large parallel processing system, this loading process can be computation intensive as well as communication intensive.

The second problem is data routing. We want to support parallel communication among neighboring nodes in the task graph at execution time. Although dilation cost in the graph mapping problem bounds the maximum number of links a message must traverse from any process to one of its neighboring processes, it does not address the link contention problem. If more than one message needs to be transferred between different pairs of neighboring processes, some links may be requested by more than one message at the same time. This may introduce extra delay.

In Chapters 4 and 5, we use the mapping functions developed in Chapters 3 and 5 to solve the program mapping problem. We first show that all of our mapping functions have simple inverse functions. Based on these inverse functions, each processor can calculate the logical addresses of the processes to be mapped into it, and all of the processors can perform the calculations in parallel. We also propose a simple scheme for loading the codes for different processes into the different processors specified by a mapping function. This scheme utilizes only a simple broadcast network and has time complexity proportional to the number of different code types, instead of the number of parallel processes. Two methods are discussed for parallel generation of translation tables. These translation tables can automatically transform inter-process communication specified in programs into inter-processor communication at execution time. For each of our mapping functions, we design a data routing strategy. We also propose a simple data routing scheme that can automatically carry out our data routing strategies at execution time. We show that because of the regularity of our mapping functions, we can efficiently simulate in the system graph without link conflicts either any permutation type set or any scatter type set of parallel neighboring communications in the task graph. For most of our mapping functions, the data routing complexities are equal to the corresponding dilation costs. For the remaining mapping functions, the data routing complexities are less than four times the corresponding dilation costs.

Since our program mapping approach supports inter-process communication at execution time, parallel programs can specify all communication on the task graph level. Even the object code of these programs is independent of system topology and can be easily transported from one system to another. With our program mapping approach, programmers are freed from responsibilities for low-level data routing steps. Parallel programs can be designed on task graphs most suitable to the problems themselves, instead of on the system graphs. As a result, our program mapping approach can support parallel programs with improved readability, verifiability, productivity, and portability.

Chapter 2
Literature Survey

2.1 Introduction

In this chapter, we review the literature on four subjects. They are (1) graph embedding, (2) task graph contraction, (3) data routing in single-stage interconnection networks, and (4) mapping of parallel programs onto parallel systems. For each subject, we begin with a brief description of the problem, and end with a brief comment about the differences between our work and those in the literature.

2.2 Graph Embedding

Given a pair of graphs G and H, an embedding of G into H is an injection (one-to-one mapping) of the nodes in G to the nodes in H. Many variations of the graph embedding problem have been studied in the literature [AR82, BMS87, DEL78b, DJ86, Ell88, Har66, HMR83, HMR73, KA88, LED76, LW87, MS88, RS78, Ros78, Ros79, Wu85]. These variations differ principally in the relative sizes of G and H, the constraints imposed on the embeddings, and the optimization measures used in the embeddings. Many important problems in parallel processing can be formulated as the graph embedding problem. These include the problem of mapping parallel programs onto parallel processing systems (by interpreting G as the task graph and H as the system graph) and the problem of evaluating the relative performance of a pair of interconnection networks (by interpreting both G and H as interconnection networks).

The most commonly used optimization measure for graph embeddings is dilation cost. Given an embedding of G into H, the dilation cost of the embedding is the maximum distance in H between the images of any two adjacent nodes in G [HMR83]. This cost gives a measure of the *proximity* in H among the adjacent nodes in G under a given embedding.

Another important optimization measure for graph embeddings is expansion cost. Given an embedding of G into H, the expansion cost of the embedding is the ratio of the size of G to the size of H. Usually, for a fixed pair of graphs, the greater the expansion cost is allowed, the smaller the dilation cost can be.

In this subsection, we list the principal embedding results in the literature. The results are classified according to the domains and ranges of the embeddings.

Embedding of ring into general graph:

- Given any ring and any connected graph of the same size, the ring can be embedded into the graph with dilation cost ≤ 3. This bound is optimal. [RS78]

Embeddings among meshes:

- An (n, n)-mesh can be embedded into a line with optimal dilation cost n. [Fit74]
- An (n, n, n)-mesh can be embedded into a line with optimal dilation cost $\lceil \frac{3}{4}n^2 + \frac{1}{2}n \rceil$. [Fit74]
- Given a 2-dimensional mesh G and a node x in G at a distance at least n steps from the mesh boundaries, let G_n be the *induced subgraph* of G consisting of all of the nodes in G at a distance less than or equal to n from x. G_n has $2n^2 + 2n + 1$ nodes. G_n can be embedded into a line with optimal dilation cost $n + 1$. [Fit74]
- Let (E, D) represent the pair of the expansion cost and the dilation cost of an embedding. Any rectangular mesh can be embedded into a square mesh with cost pairs of $(1.2, 15)$, $(1.45, 9)$, $(1.8, 3)$, or $(4, 1)$. [AR82]
- 2-dimensional rectangular meshes of large aspect ratio can be embedded into rectangles of smaller aspect ratios with small expansion and dilation costs. In particular, width can be reduced by a factor of up to 2 with optimal expansion cost and dilation cost (2). A factor of 3 can be obtained with dilation cost 3. In general, any rectangular mesh can be embedded into a square mesh that is no more than unity larger on the side than the minimum possible, with dilation cost no more than 3. [Ell88]
- The *simulation* of a graph B by another graph A is a mapping from the nodes of B to the nodes of A such that a constant maximum number of nodes in B are mapped into any node in A. Let L be a mesh of shape $(l_1, l_2, \cdots, l_d)$ and W be a mesh of shape $(w_1, w_2, \cdots, w_d)$, for both of which the lengths of the dimensions are in nonincreasing order. L can simulate W with dilation cost $O(\alpha)$, where $\alpha = \max_{1 \le i \le d}(l_1 \cdots l_i / w_1 \cdots w_i)^{\frac{1}{i}}$. This bound is optimal for fixed value of d. [Ata85, KA88]
- Let L, W, and α be the same as in the preceding result. Any embedding of W into L must have average dilation cost $\Omega(\alpha)$. [Ata85, KA88]

Embedding among toruses:

- An (m, n)-torus $(m \ge n)$ can be embedded into a ring of the same size with optimal dilation cost n. [MN86]

Embeddings of mesh, torus or tree into hypercube:

- A mesh of size some power of 2 can be embedded into a hypercube of the same size with unit dilation cost. [MT87, CS86, SS85]
- About 88.6% of 2-dimensional meshes can be embedded into hypercubes with a dilation cost of 2 and an expansion cost of 2. [HJ87]
- The *optimal hypercube* for a mesh G is the smallest hypercube that has at least as many nodes as G.

(1) There is an embedding of a mesh of shape (m, k) into its optimal hypercube with dilation cost 2, provided that:

$$\lceil \log m \rceil + \left\lceil \log \left(\left\lceil mk/2^{\lceil \log m \rceil} \right\rceil + \left\lfloor \frac{\lceil \log m \rceil}{2} \right\rfloor \right) \right\rceil \leq \lceil \log mk \rceil .$$

(2) For any $k < d$, there is an embedding of a mesh of shape $(a_1, a_2, \ldots, a_d)$ into its optimal hypercube with dilation cost $k + 1$, provided that:

$$\sum_{i=1}^{d-1} \lceil \log a_i \rceil + \lceil \log B_k \rceil \leq \left\lceil \sum_{i=1}^{d} \log a_i \right\rceil ,$$

where

$$B_k = \frac{a_d \prod_{i=1}^{k} a_i}{\prod_{i=1}^{k} 2^{\lceil \log a_i \rceil}} + \sum_{i=1}^{k} \left\lfloor \frac{\lceil \log a_i \rceil}{2} \right\rfloor .$$

(3) Let $f(k) \geq k$ be a function of k and G be a k-dimensional mesh of shape $(f(k), f(k), \ldots, f(k))$. G can be embedded into its optimal hypercube with dilation cost $\Omega(k \log(k + \log f(k))/(\log f(k) \log(k \log f(k))))$ as $k \to \infty$. This shows that the dilation cost of embedding a k-dimensional mesh into its optimal hypercube must grow with k.

[BMS87]

- A ring of size l can be embedded into a hypercube of size 2^n with unit dilation cost if l is even and $4 \leq l \leq 2^n$. [SS85]
- For $n \geq 3$, it is impossible to embed an n-level complete binary tree into the subgraph obtained by removing one of the nodes of a hypercube of size 2^n. [SS85, DJ86]
- A complete binary tree of height $h > 2$ cannot be embedded into a hypercube with dilation cost 1 and expansion cost less than 2. [Wu85]
- A complete binary tree of size $N - 1$ can be embedded into a hypercube of size N with dilation cost 2. [Wu85]
- A complete binary tree of size $N - 1$ can be embedded into a hypercube of size $2N$ with unit dilation cost. [Wu85]
- Two complete binary trees, each of size $\frac{N}{2} - 1$, can be embedded into a hypercube of size N with unit dilation cost. [DJ86]
- A *stretched* binary tree is a binary tree with an auxiliary node of degree 2 inserted between the root and one of its two sons. A stretched complete binary tree of size N can be embedded into a hypercube of the same size with unit dilation cost. [DJ86]
- A k-ary tree K_d of height d can be embedded into a $(d-1)\lceil \log_2 k \rceil + 1$ dimensional hypercube with dilation cost $2 \cdot \lceil \log_2 k \rceil$. [Wu85]
- Every N node complete binary tree can be embedded into a hypercube with $O(N^{1.71})$ nodes with unit dilation cost. [BCLR86]

- Every binary tree can be embedded into a hypercube with dilation cost 10 and expansion cost 4. [BCLR86]
- An arbitrary binary tree can be embedded into a hypercube with dilation cost 3 and expansion cost $O(1)$. [MS88]
- Every binary tree can be embedded into the smallest hypercube that has at least as many nodes as the tree with dilation cost 5. [MS88]
- There is a bounded-degree universal graph of N nodes that includes all binary trees of size less than or equal to N as subgraphs. [BCLR86]

Embeddings among trees:

- For every h, there is an embedding of any complete ternary tree of height h into the complete binary tree of height $2h$ with dilation cost 2 and expansion cost $\Omega(n^\lambda)$, where $\lambda = \log_3(\frac{4}{3})$. [HMR83]
- There is a constant $\alpha > 0$ such that, for infinitely many heights h, any embedding of the complete ternary tree of height h into a complete binary tree with expansion cost less than 2 has dilation cost $> \alpha \log\log h$. [BCLR86]
- There are generic binary trees B_n into which all n node binary trees are embeddable with dilation cost $O(1)$ and expansion cost $O(n^c)$, for some fixed constant c. [BCLR86]
- Let T be any universal binary tree that has every binary tree of size less than or equal to n as its subgraph. T has size $\Omega(n^{(\log n)/2})$. [BCLR86]

Embeddings of mesh into tree:

- If an (n, n)-mesh can be embedded into some binary tree H with dilation cost $T(n)$, then
$$T(n) \geq \log n - 3/2.$$
[DEL78b]
- If an (n, n)-mesh can be embedded into some binary tree H with average dilation cost $A(n)$, then
$$A(n) \geq n/12.$$
[DEL78b]
- If n is a power of 2, there is a binary tree H such that an (n, n)-mesh can be embedded into H with average dilation cost 8. [DEL78b]
- An $(n, n, \cdots, n)$ d-dimensional mesh $(n > 1)$ can be embedded into the leaves of a 2^d-ary tree with average dilation cost $< 4 - 2^{\lfloor \log_2 n \rfloor}$, or in the leaves of a binary tree with average dilation cost $< (4 - 2^{\lfloor \log_2 n \rfloor})d$. [RS78]

Comment on graph mapping

In this dissertation, we concentrate on embeddings among toruses and meshes of various dimensions and of various shapes, with lines, rings, and hypercubes being special cases. All of our embeddings are bijections and in the form of simple embedding functions. Many of them are proved to be optimal. For all of the known optimal embeddings among toruses and meshes in the literature, except for the case of embedding a hypercube into a mesh, our embeddings have dilation costs either optimal or within a multiplicative constant of the optimal dilation costs.

2.3 Task Graph Contraction

Given a parallel program with more processes than the processors available in a parallel processing system, we have to design a mapping from the nodes of the task graph to the nodes of the system graph to resolve two differences between the task graph and the system graph before we can execute the program. The first is the difference in topology. The second is the difference in size. While it is ideal to resolve the two differences at the same time, it is easier to tackle the problem in two steps: first contract the task graph into an intermediate graph of the same type as the task graph and of the same size as the system graph, and then embed the intermediate graph into the system graph. The objective of task graph contraction is to resolve the difference in the sizes of the task graph and the system graph.

Since the processes mapped into a single processor must be executed sequentially, we should try to balance the computation load over the processors. If we assume that all of the processes have the same computation time, this balance in computation load is achieved with an even distribution of the processes over the processors. Thus, one major optimization measure for task graph contraction is the evenness of the distribution of the nodes in the task graph over the nodes in the system graph.

Another major optimization measure for task graph contraction is dilation cost. To minimize the communication delay, neighborship in the task graph should be maintained in the intermediate graph if possible. Since the intermediate graph usually belongs to the same graph family as the task graph, and more than one node in the task graph can be mapped into a single node in the intermediate graph, task graph contraction for toruses and meshes usually has dilation cost 1 or 2.

Edge grammar based task graph contraction

Berman and Snyder [BS84, BGK*85, BS87] reported a method for task graph contraction based on edge grammars. Each parallel program is abstracted into a family of graphs $\{G_m\}$, one for each problem size. To embed a large instance G_n into the system graph H, the following two steps are taken:

1. Embed G_n into a smaller graph G_k from the same graph family, i.e., contract the program as if the program and the architecture had the same type of topology.

2. Lay out the small graph G_k on the system graph H, assigning at most one node in G_k to each node in H.

The first step is accomplished by the help of *edge grammars* [Ber83]. Edge grammars define graph families by generating pairs of vertex labels (edges) using conventional formal language mechanisms. For a class of graph families, edge grammars provide an automatable means by which large members of a graph family can be contracted to smaller members of the family. Graph families that are definable and contractable using edge grammars include square meshes, square toruses, complete binary trees, hex-connected meshes, and cube-connected cycles. The methods in [BS84, BGK*85] are basically designed for the CHiP Computer [Sny82]. All links are assumed to have equal communication loads.

We now outline the principal definitions and results in [Ber83].

Definition 2.3.1 A Type 3 Edge Grammar is a 4-tuple $\Gamma =< N, T, G, P >$ where T is a set of ordered pairs of strings over a finite alphabet Σ, N is a set of nonterminals, G in N is the start symbol, and P is a finite set of productions. All of the productions in P have the form $A \rightarrow BC$, $A \rightarrow B$, or $A \rightarrow C$ where A and B are nonterminals, and C is a terminal. The *concatenation* of two edge sets A and B is defined to be the set $AB = \{(vv', ww')|(v, w) \in A \text{ and } (v', w') \in B\}$. □

Definition 2.3.2 Let Γ be an edge grammar. The n-th graph generated by Γ, $G(n)$, is the graph with vertices and edges

$$V(n) = \{v|\text{for some } w,\ ((G \overset{*}{\Rightarrow} (w, v) \text{ or } G \overset{*}{\Rightarrow} (v, w)) \text{ and } |v| = |w| = n)\},$$

$$E(n) = \{(v, w)|G \overset{*}{\Rightarrow} (v, w), v \neq w \text{ and } |v| = |w| = n\}.$$

□

Definition 2.3.3 Let Γ be an edge grammar. The graph family generated by Γ, $G(\Gamma)$, is the set $\{G(n)\}_{n>0}$ where $G(n)$ is the n-th graph generated by Γ. □

For example, to generate the family of all complete binary trees, we can define $\Gamma = < \{T_0, T_1, T_R, R, T\}, \{(0, 0), (1, 1), (2, 2), (2, 0), (2, 1)\}, T, P >$ where the productions are

$$\begin{array}{lll}
T \rightarrow T_0 & T \rightarrow T_1 & T \rightarrow T_R \\
T_0 \rightarrow T(0,0) & T_1 \rightarrow T(1,1) & T_R \rightarrow R(2,0) \\
T \rightarrow (2,0) & T \rightarrow (2,1) & T_R \rightarrow R(2,1) \\
T \rightarrow R & R \rightarrow (2,2) & R \rightarrow R(2,2)
\end{array}$$

The first three graphs in the family are shown in Figure 2.1.

Definition 2.3.4 Let $\{G(n) = (V(n), E(n))\}$ be a graph family. Then $\{G(n)\}$ is *contractable* if for each n, there is a mapping $c : V(n+1) \rightarrow V(n)$ such that $c(V(n+1)) \subseteq V(n)$ and $\{(c(v), c(w))|(v, w) \in E(n+1)\} \subseteq E(n)$. □

Definition 2.3.5 Let $G = (V, E)$ be a graph whose labels are strings in Σ^*. Let m be the mapping that assigns to each label xa ($x \in \Sigma^*$, $a \in \Sigma$) in V the label x in Σ^*. Then the graph $m(G) = (\{m(v)|v \in V\}, \{(m(v), m(w))|(v, w) \in E\})$ is said to be a *truncation* of G. □

Definition 2.3.6 A graph family $\{G(n)\}$ is *truncatable* if for each $n > 0$, $G(n)$ is a truncation of $G(n+1)$. □

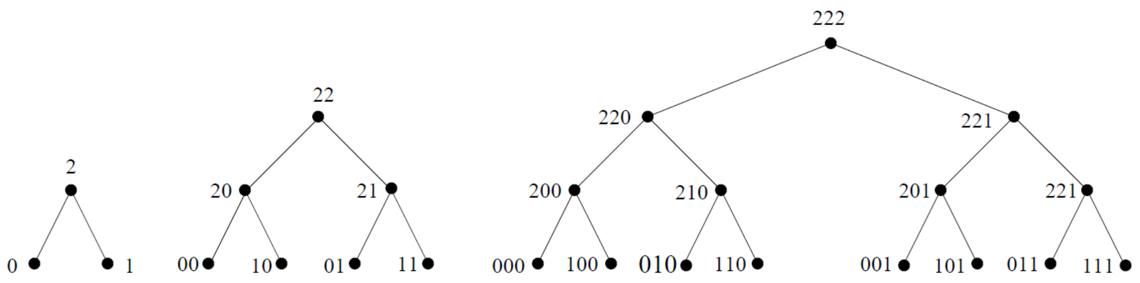


Figure 2.1: Complete binary trees

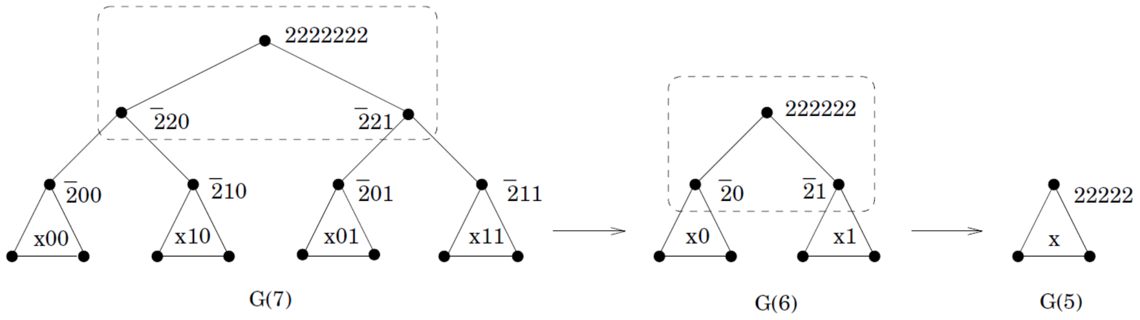


Figure 2.2: Contraction of the 255 node tree into the 63 node tree

Theorem 2.3.1 *If a graph family is truncatable, it is contractable.* □

Theorem 2.3.2 *Let Γ be a Type 3 edge grammar. Assume that*

1. *for each (v, w) in T, $|v| = |w| = 1$; and*
2. *for each nonterminal A in $N - \{G\}$, there is a production $G \to A$, where G is the start symbol.*

Then $\{G(n)\}$ is truncatable. □

Assume that a graph family $\{G(n)\}_{n>0}$ is truncatable. Given any two integers x and y such that $x > y > 0$, we can contract $G(x)$ into $G(y)$ in this way: for any two nodes in $G(x)$, if their labels are of forms uv and uw where $|u| = y$, then they are mapped to a single node in $G(y)$ with label u. Figure 2.2 uses a complete binary tree to show how a tree of size 255 ($G(7)$) is contracted into another complete binary tree of size 127 ($G(6)$), and the latter tree is then contracted into another complete binary tree of size 63 ($G(5)$).

Task graph contraction case study

Nelson and Snyder [NS86] pointed out the limitations inherent in the approach adopted by [BS84, BGK*85] and provided some case studies of task graph contraction. In [NS86], task graphs and system graphs were taken from the same graph families. Algorithms for trees, meshes, and hypercubes were used as examples. For each algorithm, they compared several possible contractions. For trees, they proved that Leiserson's layout technique [Lei83] was the best for contracting tree algorithms for finding minimum or sum. For mesh algorithms, they conjectured that coalescing by maximizing the area for a given perimeter is optimal for the algorithms with balanced edge loadings. Finally, they presented two algorithms for hypercube that require different contractions to produce optimal results.

Heuristic task graph contraction in Prep-P project

Berman [JGD87] reported an automatic mapping software system for the CHiP Computer called Prep-P. She pointed out that the task graph contraction based on edge grammars has narrow graph domain and restrictive description power. In the Prep-P project, heuristic algorithms were used to solve the following version of the general contraction problem: *Given an undirected graph with m nodes, find a partition of the nodes into at most $n \leq m$ groups such that a given cost function is minimized.* In the Prep-P project, the cost function was the number of edges between distinct partitions in the induced (contracted) graph. To simplify the design process, the following assumptions were made:

(1) The processes identified with each node perform roughly the same number of reads and writes.

(2) Parallelism is maximized when the m processes are distributed almost equally over the n partitions.

(3) Intra-processor communication is more efficient than inter-processor communication.

With this general approach, the graph domain is the set of undirected graphs. Several heuristic techniques for performing contraction were tested, including simulated annealing, local neighborhood search, branch-and-bound, greedy. The most promising of these techniques were reported to be simulated annealing and local neighborhood.

Finite element modeling program contraction

Sadayappan, Ercal, and Martin [SEM87] addressed the contraction and mapping problem in the context of implementing finite element modeling programs on two dimensional meshes. A finite element task graph is a two dimensional graph with irregular boundaries. A heuristic two-step mapping scheme with polynomial-time complexity was developed. The first step generates a graph partition for the *nearest neighbor mapping* of the finite element task graph onto the mesh graph. The second step performs heuristic boundary refinement procedure to incrementally alter the initial partition for improved load balancing among the processors. Successful application of the approach is reported only for some example finite element task graphs.

Comment on task graph contraction

The task graph contraction scheme based on edge grammars is the only non-heuristic scheme in the literature that works for more than one graph family. However, since it allows for only one parameter, graph size, in the definition of graph families, the definitional power of the edge grammar is limited. For example, in the mesh family, each edge grammar can define only the square meshes of a fixed dimension, which is a small subset of the entire mesh family. For the same reason, within a truncatable graph family, for any integers x and y such that $x > y$, there is only one way to contract $G(x)$ into $G(y)$. As pointed out in [NS86], this is not optimal for many common parallel algorithms. By edge grammar, a mesh cannot be contracted into another mesh of different dimension.

In this dissertation, we study the task graph contraction for toruses and meshes of various dimensions and of various shapes. The contraction schemes are all defined in the form of simple contraction functions. For every case in which the edge grammar can be used, our contractions can achieve at least the same contraction quality (with the contraction function $\bar{\nu}_{\mathcal{L}}$ defined in Chapter 5). A torus or a mesh can be contracted into another torus or another mesh of either higher or lower dimension.

2.4 Data Routing in Single-stage Interconnection Networks

In a parallel processing system, if more than one message must be sent from a source to a destination at the same time, some links can be contended for by more than one message at the same time. Since each link can support the communication of only one message at any instant, this contention introduces extra communication delay into the system. A good data routing algorithm should support parallel communication in the system with minimum delay.

There are two principal kinds of data routing mechanisms: circuit switching and packet switching. In circuit switching, a physical path is established between the source and the destination. In packet switching, data are put in a packet and routed through the interconnection network without establishing a physical connection path. Circuit switching is generally much more suitable for bulk data transmission, while packet switching is more efficient for many short messages. In systems with torus or mesh topologies, packet switching is usually preferred.

There are two control strategies for packet switching: centralized control and distributed control. For centralized control, the decisions for packet routing are made by the host computer based on global information. For distributed control, each processor decides how to route the data based on its own information.

Data routing approaches in the literature generally fall into two categories: those for multistage interconnection networks and those for single-stage interconnection networks. Since this thesis is about systems with torus or mesh structures, we review only data routing approaches for single-stage interconnection networks.

Sorting-based data routing

Let N be the number of processors in a parallel processing system. Nassimi and Sahni [NS80b, NS81] reported sorting-based solutions to the following two general data routing problems for SIMD machines:

1. *Random Access Read (RAR):* An index $S(i)$ is contained in PE(i), $0 \leq i < N$. PE(i) is to receive data from PE($S(i)$). We assume that the data to be transmitted to PE(i) are originally in register $D(S(i))$. ($D(j)$ denotes register or memory cell D in PE(j).) If PE(i) is not to receive data from any other PE, then $S(i) = \infty$.

2. *Random Access Write (RAW):* An index $W(i)$ is contained in PE(i), $0 \leq i < N$. Data from the D register of PE(i) are to be transmitted to PE($W(i)$), $0 \leq i < N$. If $W(i) = \infty$, then data from PE(i) are not transmitted to any PE.

Some special applications of RAR's and RAW's can be found in [NS80a] and [NS82]. Nassimi and Sahni's sorting-based routing algorithm consists of the following steps.

1. *Sort:* Records are rearranged in nondecreasing order of a specified key. Let $G(i)$ denote the record in PE(i), $0 \leq i < N$. Let $H(i)$ be the key field of record $G(i)$. $H(i)$ is also in PE(i). Following a sort, the records will have been rearranged so that $H(i) \leq H(i+1)$, $0 \leq i < N-1$.

2. *RANK:* The *rank* of a selected record is the number of selected records in PE's with a smaller index. For example, assume that we have eight PE's each containing one record. Let the key values for these eight records be $(6, 4, 2, 2^*, 6, 6^*, 3^*, 4^*)$, where an asterisk over a key value denotes a flag or selected record. The ranks of the flagged records are $(-, -, -, 0, -, 1, 2, 3)$.

3. *CONCENTRATE:* Let $G(i_r)$, $0 \leq r \leq j < N$ be a set of records with $G(i_r)$ initially in PE(i_r). Assume that the records have been ranked so that $H(i_r) = r$. A *concentrate* moves record $G(i_r)$ to PE(r), $0 \leq r \leq j$. Assume that $G(0:7) = (A, -, -, B, -, C, -, D)$, $i_0 = 0$, $i_1 = 3$, $i_2 = 5$, and $i_3 = 7$. Following a concentrate, $G(0:7) = (A, B, C, D, -, -, -)$.

4. *DISTRIBUTE:* Let $G(i)$, $0 \leq i \leq j < N$ be a set of records with $G(i)$ initially in PE(i). Let $H(i)$, $0 \leq i \leq j$ be a set of destinations such that $H(i) < H(i+1)$, $0 \leq i < j$. A *distribute* routes $G(i)$ to PE($H(i)$), $0 \leq i \leq j$. A distribute is the inverse of a concentrate. For example, suppose that $G(0:7) = (A, B, C, -, -, -, -)$, that $H(0) = 1$, $H(1) = 5$, and $H(2) = 6$. Following a distribute, $G(0:7) = (-, A, -, -, -, B, C, -)$.

5. *GENERALIZE:* A *generalize* makes multiple copies of records. The initial configuration is record $G(i)$ in PE(i), $0 \leq i \leq j < N$. Each record has a field H (height). The H values are arranged such that $0 \leq H(0) < H(1) < \cdots < H(j) \leq N-1$, and $H(i) = \infty$ for $j < i < N$. Generalize copies record $G(i)$ into PE's $H(i-1)+1$ through $H(i)$, $0 \leq i \leq j$. (We assume, for convenience, that $H(-1) = 0$.) Let $G(0:7) = (1, 5, 6, \infty, \infty, \infty, \infty, \infty)$. Following a generalize, $G(0:7) = (A, A, B, B, B, B, C, -)$.

Nassimi's RAR algorithm is best described by considering an example (Figure 2.3). In this example we have $N = 8$ PE's and $S(0:7) = (2, 6, 2, \infty, 5, 6, \infty, 6)$. Let $T(i) = i$ and FLAG(i) $= 1$, $0 \leq i < N$. The RAR algorithm begins by sorting the records $G(i) = \langle S(i), T(i), \mathrm{FLAG}(i) \rangle$. Records are sorted on S; T is used to resolve ties. During the sort, whenever a comparison between $G(i)$ and $G(j)$ is made, if $S(i) = S(j)$ and $T(i) < T(j)$, then FLAG(i) is set to zero. As a result, following the sort, FLAG(i) $= 1$ only for records with distinct S values. For records with the same S value, FLAG$= 1$ only for the record with the highest T value. Lines 3 and 4 of Figure 2.3 give the result of the sort. The S values with an asterisk above them correspond to records with a FLAG of 1.

The next step is to rank the records with a flag of 1. This results in the rank assignment of line 5 (Figure 2.3). For PE's containing a record G with FLAG$= 1$, we define a new

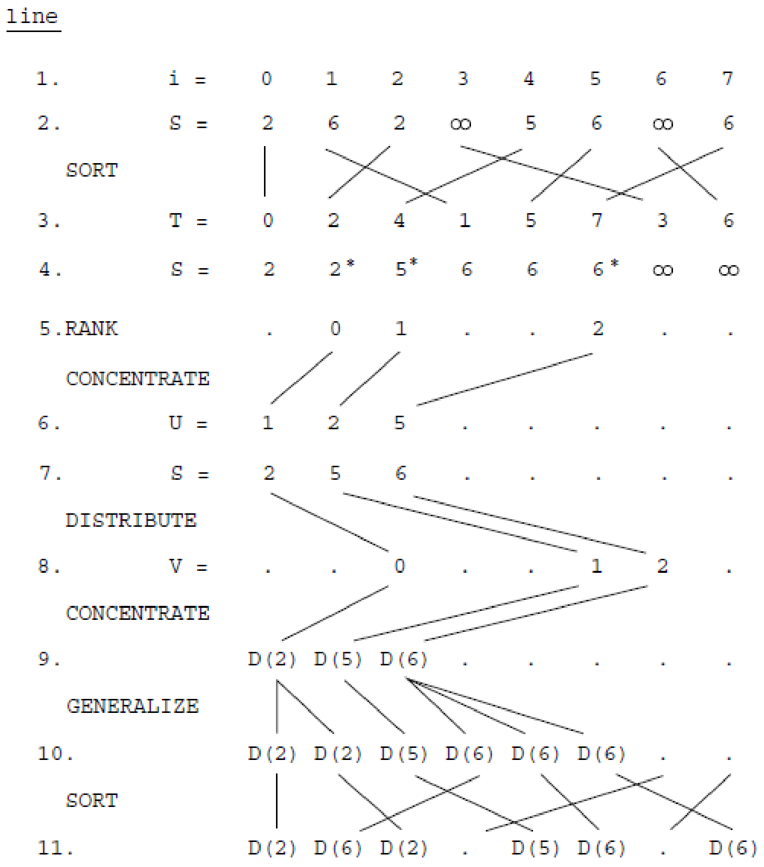


Figure 2.3: RAR example

record G', where $G'(i) = \langle R(i), U(i), S(i) \rangle$, $R(i)$ is the rank just determined, $U(i) = i$, and $S(i)$ is as in line 4 of Figure 2.3. The $G'(i)$'s are concentrated to obtain the configuration of lines 6 and 7. At this point, we define a new record G'' for each PE containing a G' type record. $G''(i) = \langle S(i), V(i) \rangle$, where $V(i) = i$. The newly defined G'' type records are distributed according to S to get the result in line 8. Observe that a PE now contains a G'' type record if and only if its data are to be transmitted to another PE. Let $D(i)$ be the data in PE(i) that is to be broadcast. The T, U, and V registers of each PE contain return addresses that are now used to broadcast the data.

First, the data to be broadcast are concentrated using the ranks contained in the V registers (line 9). Next, the data are generalized using the values in the U registers as the corresponding H values in the definition of generalize. This yields the configuration of line 10. Finally, the broadcast data are sorted using the T value in each PE as the sort key. As the result (line 11), all of the data have been broadcast to the PE's that requested them. This algorithm solves the RAR problem.

The RAW problem is similar and is omitted. Using Kung's sorting algorithm for meshes [TK77], or Batcher's merge sort adapted for cube-connected or perfect shuffle computers, Nassimi and Sahni [NS81] proved the following theorem.

Theorem 2.4.1 *Based on sorting algorithms, a RAR can be accomplished with complexity $O(q^2 n)$ on a q-dimensional n^q PE mesh-connected computer and $O(\log^2 N)$ on an N PE cube-connected or perfect shuffle computer; a RAW can be accomplished with complexity $O(q^2 n + dqn)$ on a q-dimensional mesh-connected computer and $O(\log^2 N + d \log N)$ on an N PE cube-connected or perfect shuffle computer, where d is the maximum number of data items written into any PE.* □

Nondeterministic data routing

Valiant [Val82] reported a nondeterministic routing algorithm that can perform any permutation on a hypercube of size $N = 2^n$ in $O(\log N)$ steps with overwhelming probability.

In describing the algorithm we identify each record to be routed by its starting node address. The name of each record is a number $s \in V = \{0, \ldots, N-1\}$.

The algorithm consists of two consecutive phases. Phase A sends each record $s \in V$ to a randomly chosen node $u(s) \in V$. For each s, every $u \in V$ has the same probability, $1/N$, of being chosen, and the choices for the different records are independent of each other. The second phase routes each record s from $u(s)$ to its destination $t = a(s)$.

At each instant, there is just one copy of each record, and this is either (a) being *transmitted* along an edge, or (b) waiting in a *queue* associated with such an edge, or (c) stored as *loose* at a node.

For simplicity, the algorithm is described in synchronized fashion, although this is inessential. In this form, the algorithm alternates between a transmitting mode and a bookkeeping mode. In the transmitting mode, the record at the head of each queue is transmitted along the edge associated with it and stored as loose at the recipient node. In the bookkeeping mode, each loose record is assigned to the queue of one of the outgoing edges according to some random choice, unless it has nowhere further to go in the current phase.

In phase A each record makes for itself a random ordering of the n dimensions. It considers each one in turn and, according to the toss of a coin, makes or refrains from making a *move* in that dimension from its current position. (By making a move we mean here that we add it to the appropriate queue. Actual transmission may be delayed by the presence of other records in the queue.) With this procedure, for each record, every node has the same probability of being its destination. Valiant proved that no record will have to wait in queues for more than $O(n)$ steps.

Phase B is similar except that now each record considers the set of dimensions in which its current location differs from its final destination, and moves along one randomly chosen such dimension in each step. Correctness is again immediate. Valiant proved that under the assumption that the records are initially at randomly chosen nodes (as guaranteed by Phase A), no record will wait in queues for more than $O(n)$ steps.

Valiant [Val82] proved that this distributed randomized algorithm can route every record to its destination without two records passing down the same link at any instant, and finishes within time $O(\log N)$ with overwhelming probability for all of such routing requests. ("Overwhelming" means here that given any constant S there is a corresponding parameter C of the algorithm such that the algorithm can successfully finish with probability greater than $1 - 2^{-Sn}$.) Each record carries with it $O(\log N)$ bits of bookkeeping information. No other communication among the nodes is needed.

Lower bounds for data routing

Gottlieb [GK84] proved the following interesting complexity results for data routing. The basic idea of the proofs is this: We first establish that most pairs of PE's are separated by a distance at least logarithmic in the number of PE's. A theorem of Dirac on the existence of Hamiltonian cycles is then applied to find for each PE(i) a distinct PE(j) at least a logarithmic distance away. This "processor permutation" is applied to the data items, and the minimum complexity needed to achieve the resulting data permutation is established using a Lagrange multiplier argument. The following are the main results in [GK84].

Theorem 2.4.2 *Let $Q = \{PE(0), \ldots, PE(P-1)\}$ be a degree K parallel processor of size P, and let N data items be distributed without replication among the PE's. Then there exist $h > 0$, which depends only on K, and a data permutation π such that at least $h(N/P)(\log P)$ cycles are required to achieve π, where we may choose $h = 1/3(\log K + \log 2)$.* $\square$

Corollary 2.4.1 *The permutation problem is not completely parallelizable on any degree $P^{O(1)}$ parallel processor.* $\square$

Corollary 2.4.2 *The permutation problem is not completely paralleliazable on any bounded degree parallel processor.* $\square$

Gottlieb [GK84] also showed that the bound for fixed K is sharp for evenly distributed data and a permutation given in advance by presenting an algorithm and providing that it attains the necessary speedup. Gottlieb also considered the dynamic permutation problem and presented an algorithm achieving the same speedup, but only in the supersaturation limit.

Comment on data routing

In this dissertation, we study data routing in systems with torus or mesh structures to simulate parallel inter-process communications in a task graph under various mapping schemes. In the context of mapping task graphs into system graphs, since all of the inter-process communication requirements are specified by the edges in the task graph, we need only to simulate in the system graph the parallel neighboring communications in the task graph. We show in Chapters 4 and 5 that the regularity of our mapping functions facilitates a very simple and efficient data routing approach. This approach is deterministic, and based on packet switching and distributed control strategies. After a task graph is mapped into a system graph, any permutation type or scatter type set of parallel neighboring communications in the task graph can be simulated in the system graph with data routing complexities either equal to the corresponding dilation costs, or less than four times the corresponding dilation costs. Since our mapping functions all have small dilation costs, our data routing approach has a performance better than that of sorting-based data routing in the program mapping context.

2.5 Mapping Parallel Programs onto Parallel Systems

We can view program mapping as a special form of binding parallel computations to system topologies. This binding can be performed at different stages of the program development cycle and in different forms. If this binding is performed early in the program development cycle, the resulting programs are clustered with low-level data routing steps and lack portability. On the other hand, if this binding is performed after the coding stage, the resulting programs have good abstraction of communication implementations, and are easily portable. Thus, this binding should be delayed to improve the parallel programming environment.

In this section, we review some typical mapping strategies in the literature.

The Poker System for the CHiP Computer

The Poker Parallel Programming Environment [Sny83, Sny84] is a graphics-based, interactive system for programming the Configurable, Highly Parallel (CHiP) Computer [Sny82]. Given a parallel algorithm with a known task graph, the conversion of the algorithm to an executable version involves the following steps:

(a) Embed the task graph into the switch lattice (system graph) on a screen.

(b) Program each process type in a sequential programming language.

(c) Assign one of the process types to each processor.

(d) Name the data path ports for each processor. In this step, each port in a processor used by the algorithm is identified with the corresponding edge in the task graph.

(e) Compile, assemble, coordinate, and load the program.

All of these steps are performed manually on the screen.

Program mapping in Prep-P project

Berman [JGD87] reported the program mapping strategy adopted by the ongoing Prep-P project. Prep-P is an automatic mapping software system designed for the CHiP Computer. It uses heuristic algorithms to automate the mapping steps used in the Poker environment. The input is an undirected graph using a graph description language. Each node in the graph is identified with a process (written in XX). The output of the system is Intel 8051 assembly code that, when run, executes the algorithm communication graph on a fixed parallel architecture simulator. The Prep-P system contracts, places, routes, and multiplexes the communication graph in the sequence followed in the Poker environment. In the Poker environment, these steps are performed manually. In the Prep-P system, these steps are performed automatically by heuristic algorithms. The system has been tested only for a small set of examples.

Mapping Crystal programs onto system

Saltz and Chen [SC87] reported an approach to map Crystal programs onto multi-processor systems. A Crystal program is a very high-level algorithm specification in which the detailed interactions among processes in space and time are suppressed. No explicit message passing is needed in the program specification, and task decomposition is done automatically by the Crystal compiler. The compiler generates as many logical processes as possible, and then combines clusters of logical processes to produce a problem decomposition that possesses a degree of granularity appropriate for the target machine. If the pattern of computations in a section of the program is known at compile time, a direct mapping of the algorithm may be performed. If the pattern of computations is fully determined only at runtime, the compiler constructs a symbolic representation of the data dependencies. This symbolic representation is used by a runtime system that aggregates the required computations. If enough regularity is present, the runtime system creates a parameterized mapping scheme. Different instances of the mapping scheme have a range of properties. Using information about the target machine characteristics, the runtime system chooses the appropriate instance of the mapping scheme and dynamically maps the computations onto the target architecture.

Comment on program mapping

In this research, programs are mapped onto systems at program loading time. All of the inter-process communications can be specified at the logical task graph level. These logical communications can be simulated automatically in the system graph at execution time with very low data routing complexity and system overhead. We use non-heuristic algorithms for task graph contraction, embedding, and data routing. Our approach can be supported in partitionable systems in which the partition for a task is unknown until execution time.

Chapter 3
Embeddings among Toruses and Meshes

3.1 Introduction

An embedding of a graph G (guest) into a graph H (host) is an injection (one-to-one mapping) of the nodes in G to the nodes in H. The graph embedding problem can be stated as follows: given a pair of graphs G and H, and a set of constraints and optimization measures, find an embedding of G into H that satisfies these constraints and optimizes these measures. Many variations of the graph embedding problem have been studied in the literature [AR82, BMS87, DEL78b, DJ86, Ell88, Har66, HMR83, HMR73, KA88, LED76, LW87, MS88, RS78, Ros79, Ros78, Wu85]. These variations differ mainly in the relative sizes of G and H, the constraints imposed on the embeddings, and the optimization measures used in the embeddings. Many important problems in parallel processing can be formulated as the graph embedding problem. They include the problem of mapping a parallel program onto a parallel processing system (by interpreting G as the task graph and H as the system graph) and the problem of evaluating the relative performance of a pair of interconnection networks (by interpreting G and H as interconnection networks).

This chapter studies embeddings among toruses and meshes of various dimensions. A d-dimensional torus is a graph in which each node has two neighbors in each of the d dimensions. A d-dimensional mesh is a graph in which each node, except those at the boundaries, has two neighbors in each of the d dimensions, while a boundary node in any dimension has only one neighbor in that dimension. (The terms *array* and *grid* have also been used for *mesh* in the literature.) Toruses and meshes are two families of graphs that are important in parallel processing. These two families include lines, rings, and hypercubes. Many of these graphs arise naturally as task graphs in parallel processing, particularly in the application areas of image processing, robotics, and scientific computation [Fox83, HKS*83, RK82, BB82]. Furthermore, because of their regularity and simplicity, many of these graphs have also been used widely as the topologies of large-scale interconnection networks [LM87a, Oru84, KWA82, PV79].

The most commonly used optimization measure in graph embeddings is *dilation cost.* The dilation cost of an embedding of G into H is the maximum distance in H between the images of any two adjacent nodes in G [HMR83]. This cost gives a measure of the *proximity* in H of the adjacent nodes in G under an embedding. In this chapter, we study embeddings for which G and H are of the *same* size, using dilation cost as the optimization measure. Based on the dimension of G, we divide the embeddings among toruses and meshes into two classes: (i) basic embeddings, those for which the dimension of G is 1, that is, G is either a ring or a line; and (ii) generalized embeddings, those for which the dimension of G is greater than 1. Based on the dimensions of G and H, we further divide generalized embeddings into two classes: (i) generalized embeddings for increasing dimension, those for which the dimension of G is lower than the dimension of H; and

(ii) generalized embeddings for lowering dimension, those for which the dimension of G is higher than the dimension of H. We study only those cases in generalized embeddings that satisfy some particular conditions: the condition of *expansion* for increasing dimension cases and the condition of *reduction* for lowering dimension cases.

All of our generalized embeddings are constructed from several optimal, basic embeddings, which are derived by generalizing the concept of Gray code for the radix-2 (binary) numbering system to similar sequences for mixed-radix numbering systems. For increasing dimension cases in which the shapes of G and H satisfy the condition of expansion, our embeddings have dilation costs of either 1 or 2, depending on the types of graphs of G and H. Except for the case in which G is a torus of even size and H is a mesh, these embeddings are all optimal. For lowering dimension cases in which the shapes of G and H satisfy the condition of reduction, the dilation costs of our embeddings depend on the shapes of G and H. These embeddings, however, are not optimal in general.

For the special cases in which both G and H are square, we can always construct an embedding of G into H using our results for generalized embeddings. For increasing dimension cases in which the dimension of G is divisible by the dimension of H, our embeddings have a dilation cost of 2 if G is a torus of odd size and H is a mesh, and have unit dilation cost otherwise. These embeddings are all optimal. For lowering dimension cases, our embeddings have dilation cost $2\ell^{(d-c)/c}$ if G is a torus and H is a mesh, and $\ell^{(d-c)/c}$ otherwise, where ℓ is the length of the dimensions of G, d the dimension of G, and c the dimension of H. For fixed values of d and c, these embeddings are all optimal to within a constant.

Using the sequential computation model, our basic embeddings and embeddings for increasing dimension have complexities proportional to n, and our embeddings for lowering dimension have complexities proportional to $(d-c)n$, where d is the dimension of G, c is the dimension of H, and n is the size of G and H.

3.2 Preliminaries

Unless stated otherwise, variables denote positive integers, logarithms refer to base 2, graphs are unweighted and undirected. Given an integer $n \geq 1$, we use $[n]$ to denote the set $\{0, 1, \ldots, n-1\}$, and $[n]^+$ to denote the set $\{1,\ 2,\ \ldots,\ n\}$. Given a list $(x_1, x_2, \ldots, x_p)$, we use $|(x_1, x_2, \ldots, x_p)|$ to denote the number of components in the list. Given a list $(x_1, x_2, \ldots, x_p)$ and a list $(y_1, y_2, \ldots, y_q)$, we use $(x_1, x_2, \ldots, x_p) \diamond (y_1, y_2, \ldots, y_q)$ to denote the concatenation of the two lists: $(x_1, x_2, \ldots, x_p) \diamond (y_1, y_2, \ldots, y_q) = (x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q)$. Given two functions f and g, we use $f \circ g$ to denote the composition of f and g: $(f \circ g)(x) = f(g(x))$ for all x in the domain of g. Given a positive integer k, a list $(i_1, i_2, \ldots, i_k)$, and a permutation $\pi : [k]^+ \to [k]^+$, we use $\pi((i_1, i_2, \ldots, i_k))$ to denote $(i_{\pi(1)}, \ldots, i_{\pi(k)})$. Given a rational number x, we use $\lfloor x \rfloor$ to denote the greatest integer less than or equal to x.

A *graph* $G = (V_G, E_G)$ is a pair consisting of a set V_G of nodes and a set E_G of edges. The size of G is $|V_G|$.

Definition 3.2.1 An *embedding* f of a graph $G = (V_G, E_G)$ into a graph $H = (V_H, E_H)$ is an injection $f : V_G \to V_H$. The *dilation cost* of f is $\max_{(i,j) \in E_G}\{\text{distance between nodes } f(i) \text{ and } f(j) \text{ in } H\}$. We call G the guest graph, H the host graph. □

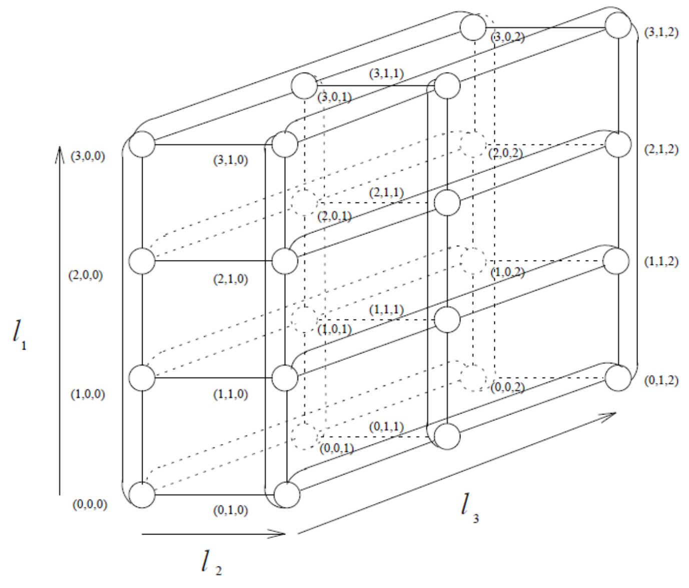


Figure 3.1: A (4, 2, 3)-torus

Definition 3.2.2 Let d be a positive integer, and $l_1, l_2, \ldots, l_d$ be integers greater than 1. An $(l_1, l_2, \ldots, l_d)$-*torus* is a connected graph with $\prod_{i\in[d]^+} l_i$ nodes. The nodes are all lists $(i_1, i_2, \ldots, i_d)$, where for all $j \in [d]^+$, $i_j \in [l_j]$. For each node $A = (i_1, i_2, \ldots, i_d)$ and each $j \in [d]^+$, A has in the j-th dimension a *left neighbor* $(i_1, i_2, \ldots, i_{j-1}, (i_j - 1) \bmod l_j, i_{j+1}, \ldots, i_d)$ and a *right neighbor* $(i_1, i_2, \ldots, i_{j-1}, (i_j + 1) \bmod l_j, i_{j+1}, \ldots, i_d)$. □

Given an $(l_1, l_2, \ldots, l_d)$-torus, $(l_1, l_2, \ldots, l_d)$ is the *shape* of the torus; d is the *dimension* of the torus; and for all $j \in [d]^+$, l_j is the *length* of the j-th dimension of the torus. If $l_1 = l_2 = \cdots = l_d$, we say that the torus is a *square graph*. A torus of dimension 1 is a *ring*. For convenience in notation, given a ring of size n, instead of using the lists (0), (1), ..., $(n-1)$ to denote its nodes, we simply use the integers 0, 1, ..., $n-1$. An example of a $(4, 2, 3)$-torus is given in Figure 3.1.

Definition 3.2.3 Let d be a positive integer, and $l_1, l_2, \ldots, l_d$ be integers greater than 1. An $(l_1, l_2, \ldots, l_d)$-*mesh* is a connected graph with $\prod_{i\in[d]^+} l_i$ nodes. The nodes are all lists $(i_1, i_2, \ldots, i_d)$, where for all $j \in [d]^+$, $i_j \in [l_j]$. For each node $A = (i_1, i_2, \ldots, i_d)$ and each $j \in [d]^+$, if $i_j \notin \{0,\ l_j - 1\}$, then A has in the j-th dimension a *left neighbor* $(i_1, i_2, \ldots, i_{j-1}, i_j - 1, i_{j+1}, \ldots, i_d)$ and a *right neighbor* $(i_1, i_2, \ldots, i_{j-1}, i_j + 1, i_{j+1}, \ldots, i_d)$. If $i_j = 0$, then A has no left neighbor in the j-th dimension, and if $i_j = l_j - 1$, then A has no right neighbor in the j-th dimension. □

The terms *shape*, *dimension*, *length* of a dimension, and *square* for meshes are defined in the same way as for toruses. A mesh of dimension 1 is a *line*. Given a line of size n, we use the integers 0, 1, ..., $n-1$ to denote its nodes. An example of a $(4, 2, 3)$-mesh is given in Figure 3.2.

Given a torus or a mesh G, the *type* of G refers to whether G is a torus or a mesh. Two graphs are of the *same type* if they are both toruses or both meshes.

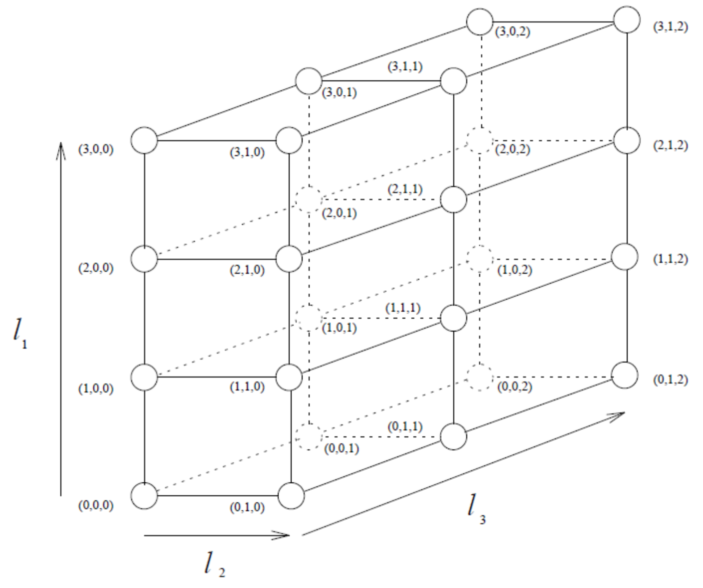


Figure 3.2: A (4, 2, 3)-mesh

Definition 3.2.4 Let $n = 2^d$, for some positive integer d. A *hypercube* of size n is a connected graph in which the nodes are all lists $(i_1, i_2, \ldots, i_d)$, where for all $i \in [d]^+$, $i_j \in \{0, 1\}$. A pair of nodes A and B are *neighbors* if the lists A and B differ in exactly one position. □

A graph G is a hypercube *if and only if* G is both a torus and a mesh: a hypercube of size n is both a $(\log n)$-dimensional torus and a $(\log n)$-dimensional mesh in which the length of each dimension is 2.

For every pair of nodes v and v' in a connected graph G, the *distance* between v and v' in G is the length of the shortest paths between v and v' in G. The following two lemmas follow directly from the definitions of toruses and meshes.

Lemma 3.2.1 *Let G be an $(l_1, l_2, \ldots, l_d)$-torus, and $A = (i_1, i_2, \ldots, i_d)$ and $B = (i'_1, i'_2, \ldots, i'_d)$ be a pair of nodes in G. The distance between A and B in G, denoted by $\delta_t(A, B)$, is $\sum_{k=1}^{d} \min\{|i_k - i'_k|, l_k - |i_k - i'_k|\}$.* □

Lemma 3.2.2 *Let G be an $(l_1, l_2, \ldots, l_d)$-mesh, and $A = (i_1, i_2, \ldots, i_d)$ and $B = (i'_1, i'_2, \ldots, i'_d)$ be a pair of nodes in G. The distance between A and B in G, denoted by $\delta_m(A, B)$, is $\sum_{k=1}^{d} |i_k - i'_k|$.* □

In the torus given in Figure 3.1, the distance between the nodes $(0, 0, 1)$ and $(3, 0, 0)$ is 2, and in the mesh given in Figure 3.2, the distance between the nodes $(0, 0, 1)$ and $(3, 0, 0)$ is 4.

Definition 3.2.5 Let d be a positive integer, and $l_1, l_2, \ldots, l_d$ be integers greater than 1. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and $n = \prod_{i=1}^{d} l_i$. For all $i \in [d+1]$, let $w_i = \prod_{j=i+1}^{d} l_j$. For every $x \in [n]$, the *radix-$\mathcal{L}$ representation* of x is the d-tuple $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)$ such that for all $j \in [d]^+$, $\hat{x}_j = \lfloor x/w_j \rfloor \bmod l_j$. $\mathcal{L}$ is a *radix-base*, and w_0, w_1, …, w_d are the *weights* in the radix-$\mathcal{L}$ representation. The set of all radix-$\mathcal{L}$ numbers, denoted by $\Omega_{\mathcal{L}}$, is the set of radix-$\mathcal{L}$ representation of x, for all $x \in [n]$. $\Omega_{\mathcal{L}}$ is a *mixed-radix numbering system.* Let $u_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ denote the bijection given above that maps each integer in $[n]$ to its radix-$\mathcal{L}$ representation in $\Omega_{\mathcal{L}}$. Let $u_{\mathcal{L}}^{-1} : \Omega_{\mathcal{L}} \to [n]$ denote the inverse of $u_{\mathcal{L}}$. For every integer $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d) \in \Omega_{\mathcal{L}}$, $u_{\mathcal{L}}^{-1}((\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)) = \sum_{k=1}^{d} \hat{x}_k w_k$. □

Every integer in $[n]$ has unique radix-$\mathcal{L}$ representation [TM75]. Note that the weight w_0 is not used in the definition of radix-$\mathcal{L}$ representation of numbers. This weight is included only for the simplification of our later definitions and analyses. Again, for convenience in presentation, when $d = 1$, instead of using the list (l_1) to denote a radix-base $\mathcal{L}$, and the lists (0), (1), …, $(l_1 - 1)$ to denote the numbers in $\Omega_{\mathcal{L}}$, we often use the integer l_1, and 0, 1, …, $l_1 - 1$, respectively. An example of the radix-$(4, 2, 3)$ numbering system is given in Figure 3.9 on page 39. In this example, $l_1 = 4$, $l_2 = 2$, $l_3 = 3$, $w_1 = 6$, $w_2 = 3$, and $w_3 = 1$.

Given a radix-base $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, we can view the radix-$\mathcal{L}$ numbers in $\Omega_{\mathcal{L}}$ as either the nodes in an $(l_1, l_2, \ldots, l_d)$-torus or the nodes in an $(l_1, l_2, \ldots, l_d)$-mesh using the obvious bijections. We can thus define the *δ_t-distance* and the *δ_m-distance* between a pair of radix-$\mathcal{L}$ numbers as the distances between the corresponding pair of nodes in a torus and in a mesh, respectively. By the definitions of δ_m-distance and δ_t-distance, the δ_m-distance between any two numbers in $\Omega_{\mathcal{L}}$ is always greater than or equal to their δ_t-distance.

Definition 3.2.6 Let n be a positive integer, $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ a radix-base, and $f : [n] \to \Omega_{\mathcal{L}}$ a bijection. Such a function f is often treated as an *acyclic sequence,* namely, $f(0)$, $f(1)$, …, $f(n-1)$. For all $i \in [n-1]$, $f(i)$ and $f(i+1)$ are *successive* elements in the acyclic sequence f. If the first and the last elements, $f(0)$ and $f(n-1)$, are also taken to be successive, then f is called a *cyclic sequence.* The *δ_m-spread* of the acyclic sequence f is the maximum of the δ_m-distances among all pairs of successive elements in f, and the *δ_t-spread* of the acyclic sequence f is the maximum of the δ_t-distances among all pairs of successive elements in f. The *δ_m-spread* and *δ_t-spread* of the cyclic sequence f are defined similarly. □

In the definition above, a function f can be treated as either an acyclic sequence or a cyclic sequence, depending on the way that successive elements are defined. Furthermore, whether f is viewed as cyclic or acyclic, we can always define a δ_m-distance and a δ_t-distance between pairs of elements of f. In the remainder of this chapter, we will simply call an acyclic sequence a *sequence.* Figure 3.3(a) gives an example of a function $f : [9] \to \Omega_{(3,3)}$, and Figure 3.3(b) shows the δ_m-distance and δ_t-distance between the pair $f(i)$ and $f((i+1) \bmod 9)$, for all $i \in [9]$. In this example, if we view f as an acyclic sequence, then the δ_m-spread of f is 2, and the δ_t-spread of f is 1. If we view f as a cyclic sequence, then the δ_m-spread of f is 3, and the δ_t-spread of f is 2.

As will be discussed in detail in the next section, given an embedding f of G into H, we often view f as an acyclic sequence if G is a line, and as a cyclic sequence if G is a ring. We use δ_m-distance measure on f if H is a mesh, and δ_t-distance measure if H is a torus.

i	0	1	2	3	4	5	6	7	8
$f(i)$	(0,0)	(0,1)	(0,2)	(2,2)	(2,1)	(2,0)	(1,0)	(1,1)	(1,2)

(a)

i	0	1	2	3	4	5	6	7	8
$\delta_m(f(i), f((i+1) \bmod 9))$	1	1	2	1	1	1	1	1	3
$\delta_t(f(i), f((i+1) \bmod 9))$	1	1	1	1	1	1	1	1	2

(b)

Figure 3.3: A function f with $n = 9$ and $\mathcal{L} = (3, 3)$

For the special case in which $n = 2^d$ and $\mathcal{L}$ is a list of d elements each equal to 2, if the function $f : [n] \to \Omega_{\mathcal{L}}$ has unit δ_t-spread (which is the same as the δ_m-spread in this case), then the sequence f is called a *Gray code* [RJD77].

3.3 Basic embeddings

In this section, we consider the embeddings of either a line or a ring into a mesh or a torus. The major results of this section are the following:

(a) A line can always be embedded into a mesh or a torus with unit dilation cost.

(b) A ring can always be embedded into a torus with unit dilation cost.

(c) A ring can be embedded into a mesh with unit dilation cost if the ring is of even size and the mesh has dimension greater than 1, and with an optimal dilation cost of 2 otherwise.

3.3.1 Embedding a line into a mesh or a torus

Let G be a line of size n, and H be either an $(l_1, l_2, \ldots, l_d)$-mesh or an $(l_1, l_2, \ldots, l_d)$-torus such that $n = \prod_{i=1}^{d} l_i$. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$. The problem of embedding G into H can be considered in terms of the radix-$\mathcal{L}$ numbers in $\Omega_{\mathcal{L}}$: the nodes in G are all numbers in $[n]$; the nodes in H are all radix-$\mathcal{L}$ numbers in $\Omega_{\mathcal{L}}$; and an embedding f of G into H is a bijection from $[n]$ to $\Omega_{\mathcal{L}}$. Since the neighbors in G correspond to the pairs of successive numbers in the sequence 0, 1, , . . . , $n-1$, the dilation cost of an embedding f is the δ_m-spread of the sequence f if H is a mesh, and the δ_t-spread if H is a torus. The problem of finding an embedding of G into H with minimum dilation cost thus corresponds to the problem of finding a sequence of all numbers in $\Omega_{\mathcal{L}}$ with minimum δ_m-spread if H is a mesh, and finding one with minimum δ_t-spread if H is a torus.

Since the δ_t-spread of a sequence is never greater than its δ_m-spread, to prove that a line can be embedded into a mesh and a torus with unit dilation cost, it suffices to prove that we can construct a sequence of all numbers in $\Omega_{\mathcal{L}}$ with unit δ_m-spread.

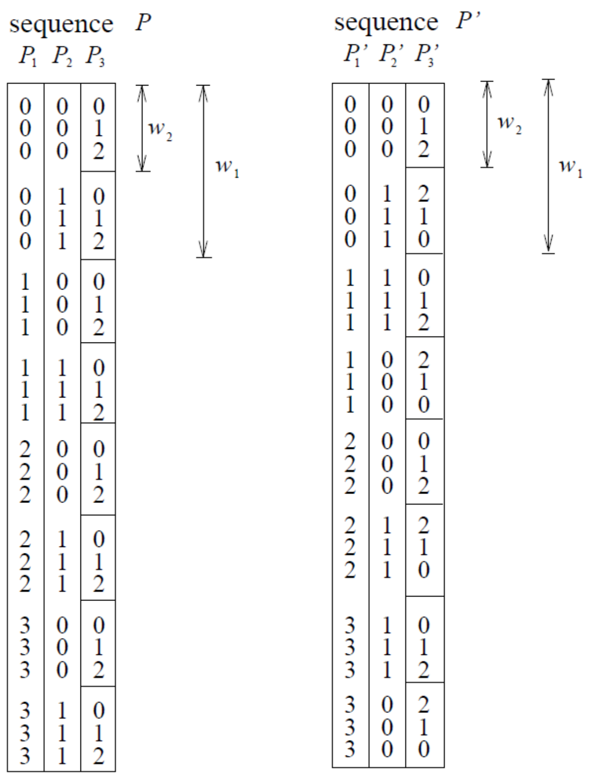


Figure 3.4: Sequences P and P' for $\mathcal{L} = (4,\ 2,\ 3)$

Let P be the sequence of numbers $0, 1, \ldots, n-1$ in their radix-$\mathcal{L}$ representations. In the following, we first show that the δ_m-spread of P is at least 2 for all $d > 1$, and then construct another sequence P' from P with unit δ_m-spread.

In the sequence P, every element a is of the form $(\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_d)$, where $\hat{a}_i \in [l_i]$, for all $i \in [d]^+$. Every element a in P thus consists of d components. The sequence P can be viewed as consisting of d separate sequences of natural numbers, namely p_1, p_2, …, p_d, all of length n, one for each of the d components of the elements in P. Let w_0, w_1, …, w_d be the weights in the radix-$\mathcal{L}$ representation. From the properties of the radix-$\mathcal{L}$ representation of numbers, for all $i \in [d]^+$, the sequence p_i can be partitioned into n/w_{i-1} segments, each with w_{i-1} elements and of the form $\underbrace{0 \cdots 0}_{w_i}\underbrace{1 \cdots 1}_{w_i}\underbrace{(l_i - 1) \cdots (l_i - 1)}_{w_i}$. We number these segments from 0 to $n/w_{i-1} - 1$ successively. For every pair of successive elements in p_i, for all $i \in [d]^+$, if they belong to the same segment in p_i, then their difference is at most 1; otherwise, their difference is $l_i - 1$. The sequence P has thus a δ_m-spread greater than 1 for all $d > 1$. An example of the sequence P for $\mathcal{L} = (4,\ 2,\ 3)$ and $n = 24$ is shown in Figure 3.4.

We next construct a sequence P' with unit δ_m-spread from P. The sequence P' can also be viewed as consisting of d sequences, p'_1, p'_2, …, p'_d. For all $i \in [d]^+$, p'_i is constructed from p_i by reversing all of the odd-numbered segments of p_i, which produces segments of the form $\underbrace{(l_i-1)\cdots(l_i-1)}_{w_i}\underbrace{1\cdots 1}_{w_i}\underbrace{0\cdots 0}_{w_i}$, and by leaving all of the even-numbered segments unchanged. As will be proved below, for every pair of successive elements in p'_i, if they belong to the same segment, their difference is at most 1; otherwise, their difference is 0. The sequence P' has unit δ_m-spread. An example of P' for $\mathcal{L} = (4,\ 2,\ 3)$ and $n = 24$ is shown in Figure 3.4.

We now define a function $f_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$. Lemma 3.3.1 shows that the sequence $f_{\mathcal{L}}$ is a sequence of all numbers in $\Omega_{\mathcal{L}}$, and Lemma 3.3.2 and Lemma 3.3.3 show respectively that the sequence $f_{\mathcal{L}}$ has unit δ_m-spread and unit δ_t-spread. The sequence $f_{\mathcal{L}}$ is P'.

Definition 3.3.1 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. Let w_0, w_1, …, w_d be the weights in the radix-$\mathcal{L}$ representation. For every $x \in [n]$, let $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)$ be the radix-$\mathcal{L}$ representation of x. The function $f_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows: for all $x \in [n]$, $f_{\mathcal{L}}(x) = (x_1, x_2, \ldots, x_d)$, where for all $i \in [d]^+$,

$$x_i = \begin{cases} \hat{x}_i, & \text{if } \lfloor x/w_{i-1} \rfloor \text{ is even;} \\ l_i - \hat{x}_i - 1, & \text{if } \lfloor x/w_{i-1} \rfloor \text{ is odd.} \end{cases}$$

□

In the definition above, for all $i \in [d]^+$, $\lfloor x/w_{i-1} \rfloor$ determines the segment in the sequence p_i to which $\hat{x}_i$ belongs. An example of the function $f_{\mathcal{L}}$ is given in Figure 3.9 on page 39.

We say that two numbers have the same *parity* if they are both even or both odd.

Lemma 3.3.1 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. The function $f_{\mathcal{L}}$ is bijective.* □

Proof. Since $|\Omega_{\mathcal{L}}| = n$, to show that $f_{\mathcal{L}}$ is bijective, it is sufficient to show that $f_{\mathcal{L}}$ is injective. Let x and y be an arbitrary pair of distinct integers in $[n]$. We want to show that $f_{\mathcal{L}}(x) \neq f_{\mathcal{L}}(y)$. Let $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)$ and $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_d)$ be the radix-$\mathcal{L}$ representations of x and y. Let $f_{\mathcal{L}}(x) = (x_1, x_2, \ldots, x_d)$, and $f_{\mathcal{L}}(y) = (y_1, y_2, \ldots, y_d)$. Since every integer in $[n]$ has a unique radix-$\mathcal{L}$ representation, there is at least one index $i \in [d]^+$ such that $\hat{x}_i \neq \hat{y}_i$. Let $k \in [d]^+$ be the smallest index such that $\hat{x}_k \neq \hat{y}_k$. We first show that $\lfloor x/w_{k-1} \rfloor$ and $\lfloor y/w_{k-1} \rfloor$ have the same parity. There are two cases:

Case 1. $k = 1$.

Since $w_0 = n$, $\lfloor x/w_0 \rfloor = \lfloor y/w_0 \rfloor = 0$. Thus, $\lfloor x/w_0 \rfloor$ and $\lfloor y/w_0 \rfloor$ have the same parity.

Case 2. $k > 1$.

Assume for contradiction that $\lfloor x/w_{k-1} \rfloor$ and $\lfloor y/w_{k-1} \rfloor$ have different parities. This implies that $\lfloor x/w_{k-1} \rfloor \neq \lfloor y/w_{k-1} \rfloor$. Since $\hat{x}_{k-1} = \hat{y}_{k-1}$, we also have $\lfloor x/w_{k-1} \rfloor \bmod l_{k-1} = \lfloor y/w_{k-1} \rfloor \bmod l_{k-1}$. It follows that $|\, \lfloor x/w_{k-1} \rfloor - \lfloor y/w_{k-1} \rfloor \,| = c\, l_{k-1}$, for some positive integer c. By the definition of radix-base, $l_{k-1} > 1$, and hence, $|\, \lfloor x/w_{k-1} \rfloor - \lfloor y/w_{k-1} \rfloor \,| > 1$. This implies that $|x - y| > w_{k-1}$. On the other hand, since k is the smallest index such that $\hat{x}_k \neq \hat{y}_k$, we have

$$|x - y| \leq \sum_{j=k}^{d} |\, \hat{x}_j w_j - \hat{y}_j w_j \,| \leq \sum_{j=k}^{d} (l_j - 1)\ w_j.$$

Since by definition, for all $j \in [d+1]$, $w_j = \prod_{i=j+1}^{d} l_i$, we have for all $j \in [d]^+$, $l_j w_j = w_{j-1}$. Thus,

$$|x - y| \leq \sum_{j=k-1}^{d-1} w_j - \sum_{j=k}^{d} w_j = w_{k-1} - w_d < w_{k-1},$$

which is a contradiction. Therefore, $\lfloor x/w_{k-1} \rfloor$ and $\lfloor y/w_{k-1} \rfloor$ have the same parity.

If $\lfloor x/w_{k-1} \rfloor$ and $\lfloor y/w_{k-1} \rfloor$ are both even, then we have $x_k = \hat{x}_k$ and $y_k = \hat{y}_k$. If they are both odd, then we have $x_k = l_k - \hat{x}_k - 1$ and $y_k = l_k - \hat{y}_k - 1$. In either case, the fact that $\hat{x}_k \neq \hat{y}_k$ implies that $x_k \neq y_k$. Thus, $f_{\mathcal{L}}(x) \neq f_{\mathcal{L}}(y)$. The function $f_{\mathcal{L}}$ is therefore bijective. □

Lemma 3.3.2 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. For all $x \in [n-1]$, $\delta_m(f_{\mathcal{L}}(x), f_{\mathcal{L}}(x+1)) = 1$.* □

Proof. Let x be an arbitrary number in $[n-1]$, and let $y = x+1$. Let $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)$ and $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_d)$ be the radix-$\mathcal{L}$ representations of x and y. Let $f_{\mathcal{L}}(x) = (x_1, x_2, \ldots, x_d)$, and $f_{\mathcal{L}}(y) = (y_1, y_2, \ldots, y_d)$. We want to show that $(x_1, x_2, \ldots, x_d)$ and $(y_1, y_2, \ldots, y_d)$ differ by 1 in exactly one position.

First we look at the relationship between the values of $\hat{x}_i$ and $\hat{y}_i$ for all $i \in [d]^+$. Since $x < n-1$, by the properties of the radix-$\mathcal{L}$ representation of numbers, there exists exactly one index $k \in [d]^+$ such that $\hat{x}_k < l_k - 1$ and for all $i \in \{k+1, \ \ldots, \ d\}$, $\hat{x}_i = l_i - 1$. Since $y = x+1$, for all $i \in \{k+1, \ \ldots, \ d\}$, $\hat{y}_i = 0$; $\hat{y}_k = \hat{x}_k + 1$; and for all $i \in \{1, \ \ldots, \ k-1\}$, $\hat{y}_i = \hat{x}_i$.

We now look at the relationship between x_i and y_i, for all $i \in [d]^+$. There are three cases:

Case 1. $i \in \{k+1, \ \ldots, \ d\}$.

First we show that $\lfloor x/w_{i-1} \rfloor$ and $\lfloor y/w_{i-1} \rfloor$ have different parities. Since $\hat{x}_{i-1} \neq \hat{y}_{i-1}$, we have $\lfloor x/w_{i-1} \rfloor \bmod l_{i-1} \neq \lfloor y/w_{i-1} \rfloor \bmod l_{i-1}$, and hence, $\lfloor x/w_{i-1} \rfloor \neq \lfloor y/w_{i-1} \rfloor$. Furthermore, since x and y differ only by 1, $\lfloor y/w_{i-1} \rfloor = \lfloor x/w_{i-1} \rfloor + 1$. Therefore, $\lfloor x/w_{i-1} \rfloor$ and $\lfloor y/w_{i-1} \rfloor$ have different parities. Since $\hat{x}_i = l_i - 1$ and $\hat{y}_i = 0$, we have $x_i = y_i$.

Case 2. $i \in [k-1]^+$.

First we show that $\lfloor x/w_{i-1} \rfloor$ and $\lfloor y/w_{i-1} \rfloor$ have the same parity. If $i = 1$, then since $w_0 = n$, we have $\lfloor x/w_0 \rfloor = \lfloor y/w_0 \rfloor = 0$. Therefore, $\lfloor x/w_{i-1} \rfloor$ and $\lfloor y/w_{i-1} \rfloor$ have the same parity. If $i \in \{2, 3, \ldots, k-1\}$, then since $\hat{x}_{i-1} = \hat{y}_{i-1}$, we have $\lfloor x/w_{i-1} \rfloor \bmod l_{i-1} = \lfloor y/w_{i-1} \rfloor \bmod l_{i-1}$, and since $l_{i-1} > 1$ and x and y differ only by 1, we have $\lfloor x/w_{i-1} \rfloor = \lfloor y/w_{i-1} \rfloor$. Therefore, $\lfloor x/w_{i-1} \rfloor$ and $\lfloor y/w_{i-1} \rfloor$ also have the same parity. Since $\hat{x}_i = \hat{y}_i$, we have $x_i = y_i$.

Case 3. $i = k$.

Using a proof as the one in Case 2, we can show that $\lfloor x/w_{k-1} \rfloor$ and $\lfloor y/w_{k-1} \rfloor$ have the same parity. Since $\hat{y}_k = \hat{x}_k + 1$, we have $|y_k - x_k| = 1$.

Since $\delta_m(f_{\mathcal{L}}(x), f_{\mathcal{L}}(x+1)) = \sum_{i=1}^{d} |x_i - y_i|$, we have $\delta_m(f_{\mathcal{L}}(x), f_{\mathcal{L}}(x+1)) = 1$. □

Lemma 3.3.3 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. For all $x \in [n-1]$, $\delta_t(f_{\mathcal{L}}(x), f_{\mathcal{L}}(x+1)) = 1$.* □

Proof. Since for any two numbers in $\Omega_{\mathcal{L}}$ their δ_m-distance is never less than their δ_t-distance, the claim follows from Lemma 3.3.2. □

Theorem 3.3.1 *Let G be a line, and H be either an $(l_1, l_2, \ldots, l_d)$-torus or an $(l_1, l_2, \ldots, l_d)$-mesh such that G and H are of the same size. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$. The line G can be embedded into H with unit dilation cost. The function $f_{\mathcal{L}}$ gives such an optimal embedding.* □

Proof. The theorem follows from Lemmas 3.3.1, 3.3.2, and 3.3.3 by interpreting the numbers in $[n]$ as the nodes in G, and the radix-$\mathcal{L}$ numbers in $\Omega_{\mathcal{L}}$ as the nodes in H. □

An example of embedding a line into a mesh using the function $f_{\mathcal{L}}$ is given in Figure 3.10 on page 40.

3.3.2 Embedding a ring into a mesh or a torus

Let G be a ring of size n, and H be either an $(l_1, l_2, \ldots, l_d)$-mesh or an $(l_1, l_2, \ldots, l_d)$-torus such that $n = \prod_{i=1}^{d} l_i$. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$. As with the problem of embedding a line into a mesh, we can consider this problem in terms of the radix-$\mathcal{L}$ numbers in $\Omega_{\mathcal{L}}$. The neighbors in a ring of size n correspond to the pairs of successive numbers in the *cyclic* sequence 0, 1, ,…, $n-1$. The problem of finding an embedding of G into H with minimum dilation cost thus corresponds to the problem of finding a cyclic sequence of all radix-$\mathcal{L}$ numbers in $\Omega_{\mathcal{L}}$ with minimum δ_m-spread if H is a mesh and finding one with minimum δ_t-spread if H is a torus.

In this section, we first show that the δ_m-spread of the cyclic sequence $f_{\mathcal{L}}$ is at least $l_1 - 1$. We then construct from $f_{\mathcal{L}}$ another cyclic sequence $g_{\mathcal{L}}$ with a δ_m-spread of 2. The function $g_{\mathcal{L}}$ provides an embedding of a ring into a mesh with a dilation cost of 2. We also prove that a ring of odd size cannot be embedded into a mesh of the same size with unit dilation cost. The embedding function $g_{\mathcal{L}}$ is therefore optimal for all rings and meshes of odd sizes. Finally, we construct a cyclic sequence $h_{\mathcal{L}}$ that has unit δ_m-spread if $\mathcal{L}$ consists of at least two components, and with the first component being an even number. The function $h_{\mathcal{L}}$ can be used to construct an embedding of a ring of even size into a higher-dimensional mesh with unit dilation cost. Furthermore, the cyclic sequence $h_{\mathcal{L}}$ has unit δ_t-spread. Thus, the function $h_{\mathcal{L}}$ also provides an optimal embedding of a ring into a torus with unit dilation cost.

Embedding a ring into a mesh

The embedding function $g_{\mathcal{L}}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. Let $f_{\mathcal{L}}(n-1) = (n_1, n_2, \ldots, n_d)$. The radix-$\mathcal{L}$ representation of $n-1$ is $(l_1 - 1, l_2 - 1, \ldots, l_d - 1)$. Since $w_0 = n$, we have $\lfloor (n-1)/w_0 \rfloor = 0$. It follows from the definition of $f_{\mathcal{L}}$ that $n_1 = l_1 - 1$. Hence, the δ_m-distance between $f_{\mathcal{L}}(0)$ and $f_{\mathcal{L}}(n-1)$ is at least $l_1 - 1$. The cyclic sequence $f_{\mathcal{L}}$ therefore has a δ_m-spread of at least $l_1 - 1$.

A cyclic sequence with a δ_m-spread of 2 can be constructed from $f_{\mathcal{L}}$ in the following way. We number all the elements in $f_{\mathcal{L}}$ successively from 0 to $n-1$. Let R' and R'' be the following two sequences: R' consists of all even-numbered elements in $f_{\mathcal{L}}$ in the same order as they appear in $f_{\mathcal{L}}$, and R'' consists of all odd-numbered elements in $f_{\mathcal{L}}$ in the reverse order. Since the sequence $f_{\mathcal{L}}$ has unit δ_m-spread, both R' and R'' have a δ_m-spread of 2. The cyclic sequence $R'R''$, the concatenation of R' and R'', has a δ_m-spread of 2: the first element in R' and the last element in R'' correspond to the first two elements in $f_{\mathcal{L}}$; the last element in R' and the first element in R'' correspond to the last two elements in $f_{\mathcal{L}}$; and the sequence $f_{\mathcal{L}}$ has a unit δ_m-spread.

We first define the function $t_n : [n] \to [n]$. This function defines a cyclic sequence of all numbers in $[n]$ with a δ_m-spread of 2. We then define the function $g_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ using $f_{\mathcal{L}}$ and t_n. The sequence $g_{\mathcal{L}}$ is $R'R''$.

Definition 3.3.2 Let n be any positive integer. The function $t_n : [n] \to [n]$ is defined as follows: for all $x \in [n]$,

if n is even, then

$$t_n(x) = \begin{cases} 2x, & \text{if } x < n/2; \\ n - 2(x - n/2) - 1, & \text{otherwise;} \end{cases}$$

if n is odd, then

$$t_n(x) = \begin{cases} 2x, & \text{if } x < (n+1)/2; \\ n - 2(x - (n+1)/2) - 2, & \text{otherwise.} \end{cases}$$

□

Definition 3.3.3 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. The function $g_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows: for all $x \in [n]$,

$$g_{\mathcal{L}}(x) = f_{\mathcal{L}}(t_n(x)).$$

□

An example of the function $g_{\mathcal{L}}$ for $\mathcal{L} = (4, 2, 3)$ is given in Figure 3.9 on page 39. It is clear that the function $g_{\mathcal{L}}$ is bijective. The next lemma follows directly from the definition of $g_{\mathcal{L}}$ and the properties of $f_{\mathcal{L}}$.

Lemma 3.3.4 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. For all $x \in [n]$, $\delta_m(g_{\mathcal{L}}(x), g_{\mathcal{L}}((x+1) \bmod n)) \leq 2$.* □

Theorem 3.3.2 *Let G be a ring, and H be an $(l_1, l_2, \ldots, l_d)$-mesh such that G and H are of the same size. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$. The ring G can always be embedded into H with a dilation cost of 2. The function $g_{\mathcal{L}}$ gives such an embedding. Furthermore, such an embedding is optimal if H is a line or has odd size.* □

Proof. We need only prove that a ring cannot be embedded into either a line or a mesh of odd size with unit dilation cost. The other part of the theorem follows from Lemma 3.3.4.

For the case in which H is a line, it suffices to notice that since each of the two boundary nodes of a line has only one neighbor, a ring cannot be embedded into a line with unit dilation cost. For the case in which H is of odd size and of dimension greater than 1, we prove the theorem by showing that there is no Hamiltonian circuit in such a mesh.

Assume for contradiction that a Hamiltonian circuit exists in an $(l_1, l_2, \ldots, l_d)$-mesh of odd size. Since the mesh has an odd number of nodes, the circuit also has an odd number of edges. By specifying a direction in the circuit, we can view all of the edges in the circuit as directed. Each node in the mesh is a list of d components, $(i_1, i_2, \ldots, i_d)$, where $i_j \in [l_j]$, for all $j \in [d]^+$. Since each edge (u, v) in the circuit connects a pair of neighboring nodes in the mesh, u and v differ in exactly one component by 1, that is, v can be obtained from u by either increasing or decreasing exactly one component of u by 1. Furthermore, for each edge (u, v) in the circuit, if v can be obtained from u by increasing the k-th component of u from a to $a+1$, where $k \in [d]^+$ and a, $a+1 \in [l_k]$, then there must exist an edge (s, t) in the circuit such that t can be obtained from s by decreasing the k-th component of s from $a+1$ to a; otherwise, if we traverse the circuit starting from the node u, we will not be able to return to u in the circuit. For a similar reason, the reverse of the above condition is also true: if v can be obtained from u by decreasing the k-th component of u from $a+1$ to a, then there must exist an edge (s, t) in the circuit such that t can be obtained from s by increasing the k-th component of s from a to $a+1$. It follows that every edge in the circuit has a unique *mate*. Therefore, the number of edges in the circuit is even. This contradicts the assumption that H is of odd size. □

An example of an embedding of a ring of size 24 into a $(4, 2, 3)$-mesh using the function $g_{\mathcal{L}}$ is given in Figure 3.10 on page 40.

The proof of the following corollary is contained in the proof of the theorem above.

Corollary 3.3.1 *There is no Hamiltonian circuit in any mesh of odd size.* □

For the special case where the mesh is of even size and of dimension at least 2, a ring can always be embedded into it with unit dilation cost. In the following, we first construct an embedding function $r_{\mathcal{L}}$ for the simple case where the dimension of the mesh is exactly 2, and then construct a function $h_{\mathcal{L}}$ for the case where the dimension of the mesh is at least 2.

The embedding function $r_{\mathcal{L}}$

The following lemma gives a property of $f_{\mathcal{L}}$ that will be used in the construction of the function $r_{\mathcal{L}}$.

Lemma 3.3.5 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. If l_1 is even, then $f_{\mathcal{L}}(n-1) = (l_1 - 1,\ 0, \ldots,\ 0)$.* □

Proof. By definition, the radix-$\mathcal{L}$ representation of $n-1$ is $(l_1 - 1, l_2 - 1, \ldots, l_d - 1)$. Since $w_0 = n$, $\lfloor (n-1)/w_0 \rfloor = 0$. We want to show that if l_1 is even, then for all $i \in \{2,\ \ldots, d\}$, $\lfloor (n-1)/w_{i-1} \rfloor$ is odd. These results together with the definition of the function $f_{\mathcal{L}}$ will then imply the lemma.

Since $n = \prod_{k=1}^{d} l_k$, and, by definition, for all $i \in \{2,\ \ldots, d\}$, $w_{i-1} = \prod_{j=i}^{d} l_j$, we can write $\lfloor (n-1)/w_{i-1} \rfloor$ as $\lfloor \prod_{j=1}^{i-1} l_j - (1/w_{i-1}) \rfloor$. Furthermore, since $0 < (1/w_{i-1}) \leq 1$, we have $\lfloor (n-1)/w_{i-1} \rfloor = \prod_{j=1}^{i-1} l_j - 1$. Therefore, for all $i \in \{2,\ \ldots, d\}$, $\lfloor (n-1)/w_{i-1} \rfloor$ is odd if l_1 is even. □

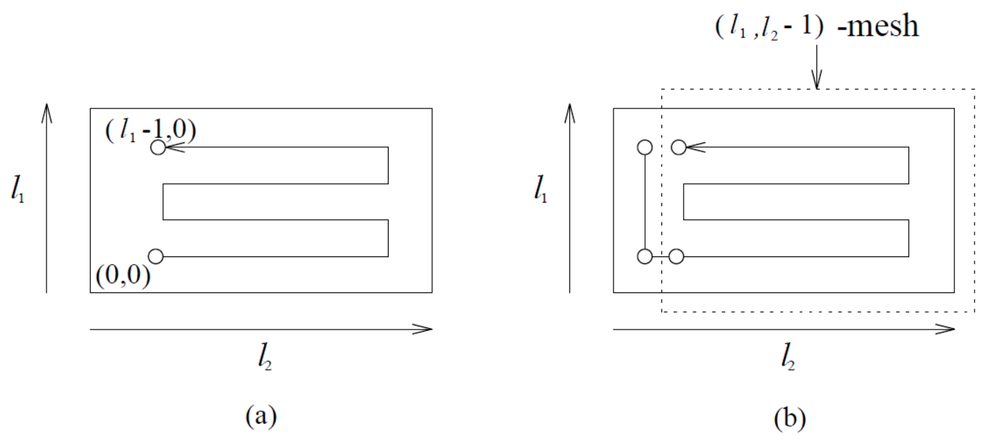


Figure 3.5: Embedding a ring into an (l_1, l_2)-mesh with $l_1 = 4$ and $l_2 > 2$

Let G be a ring, and H be an $(l_1,\ l_2)$-mesh such that l_1 is even, and G and H are of the same size. Let $\mathcal{L} = (l_1,\ l_2)$. We assume the following coordinates: the origin of the mesh H, $(0, 0)$, is at the lower left corner, the first dimension increases vertically upward, and the second dimension increases horizontally to the right. If we use the function $f_{\mathcal{L}}$ to embed the ring into the mesh, then by Lemma 3.3.5, both the first and the last nodes from the ring are embedded into the first column of the mesh, with node 0 at the bottom (node $(0,\ 0)$ in the mesh) and node $n-1$ at the top (node $(l_1-1,\ 0)$ in the mesh) (see Figure 3.5(a)). The δ_m-distance between $f_{\mathcal{L}}(0)$ and $f_{\mathcal{L}}(n-1)$ is thus l_1-1. For the case in which $l_2 > 2$, the following simple modification of $f_{\mathcal{L}}$ gives an embedding of G into H with unit dilation cost. We first embed the nodes from the ring successively into the first column of the mesh, from top to bottom, and then by treating the remaining nodes in the mesh as an (l_1, l_2-1)-mesh, we embed the remaining nodes from the ring using the function $f_{(l_1,l_2-1)}$. (See Figure 3.5(b).) In this embedding, all neighboring nodes in the ring are embedded into neighboring nodes in the mesh.

For the case in which $l_2 = 2$, the function $f_{(l_1,l_2-1)}$ is not defined because every component in a radix-base must be greater than 1. For this case, we simply embed the nodes from the ring successively into the first column of the mesh, from top to bottom, and then embed the remaining nodes from the ring into the second column of the mesh, from bottom to top. This embedding also has unit dilation cost.

We next define the function $r_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$. This function $r_{\mathcal{L}}$ gives the embedding above.

Definition 3.3.4 Let $\mathcal{L} = (l_1,\ l_2)$ be a radix-base, and let $n = l_1 l_2$. The function $r_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows: for all $x \in [n]$,

if $l_2 > 2$, then

$$r_{\mathcal{L}}(x) = \begin{cases} (l_1 - 1 - x, 0), & \text{if } x < l_1; \\ (x_1, x_2 + 1) \text{ where } (x_1, x_2) = f_{(l_1,l_2-1)}(x - l_1), & \text{if } x \geq l_1; \end{cases}$$

if $l_2 = 2$, then

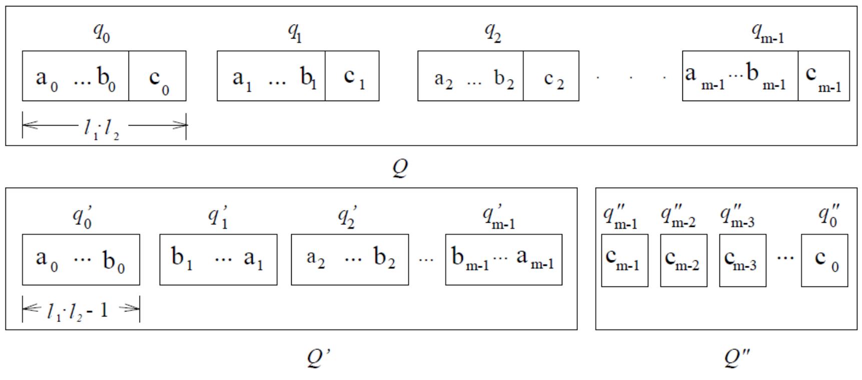


Figure 3.6: Q, Q' and Q'' for even m

$$r_{\mathcal{L}}(x) = \begin{cases} (l_1 - 1 - x, 0), & \text{if } x < l_1; \\ (x - l_1, 1), & \text{if } x \geq l_1. \end{cases}$$

□

The next lemma follows directly from the definition of $r_{\mathcal{L}}$ and the properties of the function $f_{\mathcal{L}}$.

Lemma 3.3.6 *Let $\mathcal{L} = (l_1,\ l_2)$ be a radix-base for which l_1 is even, and let $n = l_1 l_2$. For all $x \in [n]$, $\delta_m(r_{\mathcal{L}}(x), r_{\mathcal{L}}((x+1) \bmod n)) = 1$.* □

The embedding function $h_{\mathcal{L}}$

We next consider the case of embedding a ring of even size into a mesh of dimension at least 3. Given a mesh of even size, first we assume that the length of its first dimension is even.

Let $d \geq 3$, let $\mathcal{L} = (l_1, l_2, \dots, l_d)$ be a radix-base for which l_1 is even, and let $n = \prod_{i=1}^{d} l_i$. Let $\mathcal{L}' = (l_1,\ l_2)$, $\mathcal{L}'' = (l_3,\ l_4, \dots,\ l_d)$, and $m = \prod_{i=3}^{d} l_i$. We now construct a cyclic sequence of the numbers in $\Omega_{\mathcal{L}}$ with unit δ_m-spread. This sequence is defined in terms of $r_{\mathcal{L}'}$ and $f_{\mathcal{L}''}$. We first define m sequences $q_0, q_1, \dots, q_{m-1}$, each of which has length $l_1 l_2$. For all $i \in [m]$, let q_i be the sequence $r_{\mathcal{L}'}(0) \diamond f_{\mathcal{L}''}(i)$, $r_{\mathcal{L}'}(1) \diamond f_{\mathcal{L}''}(i)$, …, $r_{\mathcal{L}'}(l_1 l_2 - 1) \diamond f_{\mathcal{L}''}(i)$. ($\diamond$ is the operator for concatenating two lists, as defined in Section 3.2, page 22.) Since the function $r_{\mathcal{L}'} : [l_1 l_2] \to \Omega_{\mathcal{L}'}$ and the function $f_{\mathcal{L}''} : [m] \to \Omega_{\mathcal{L}''}$ are both bijective, each of these sequences consists of $l_1 l_2$ distinct numbers in $\Omega_{\mathcal{L}}$. Next we construct two disjoint segments from each of these sequences: for all $i \in [m]$, the segment q'_i consists of the first $l_1 l_2 - 1$ elements of q_i, with these elements in the same order as they appear in q_i if i is even and in the reverse order if i is odd; and the segment q''_i consists of the last element in q_i. Let $Q' = q'_0\ q'_1\ \cdots\ q'_{m-1}$, $Q'' = q''_{m-1}\ q''_{m-2}\ \cdots\ q''_0$, and $Q = Q'Q''$. An example of Q, Q', and Q'' is given in Figure 3.6 for even m. The sequence Q consists of all numbers in $\Omega_{\mathcal{L}}$, and each element in Q is a list of d components. We now show that the cyclic sequence Q has unit δ_m-spread by establishing the following claims.

Claim 1. The sequence Q' has unit δ_m-spread.

For every pair of successive elements in Q', if they belong to the same segment q'_i, for some $i \in [m]$, then they have the same rightmost $d-2$ components, which are the components of $f_{\mathcal{L}''}(i)$, and their leftmost two components correspond to successive elements in the sequence $r_{\mathcal{L}'}$. Therefore the δ_m-distance between them is 1. If they belong to different segments, then they have the same leftmost two components, which are either the components of $r_{\mathcal{L}'}(0)$ or the components of $r_{\mathcal{L}'}(l_1 l_2 - 2)$, and their rightmost $d-2$ components correspond to successive elements in the sequence $f_{\mathcal{L}''}$. Therefore the δ_m-distance between them is also 1. The sequence Q' thus has unit δ_m-spread.

Claim 2. The sequence Q'' has unit δ_m-spread.

All elements in Q'' have the same leftmost two components, which are the components of $r_{\mathcal{L}'}(l_1 l_2 - 1)$. Furthermore, for every pair of successive elements in Q'', their rightmost $d-2$ components correspond to successive elements, in reverse order, in $f_{\mathcal{L}''}$. The sequence Q'' therefore has unit δ_m-spread.

Claim 3. The cyclic sequence Q has unit δ_m-spread.

Let y' and z' be the first and last elements of Q', and y'' and z'' be the first and last elements of Q''. We show that the δ_m-distance between z' and y'' and the δ_m-distance between y' and z'' are both 1. Both z' and y'' come from the sequence q_{m-1}, with y'' being the last element in q_{m-1}, and depending on whether m is even or odd, z' being either the first or the second to last element in q_{m-1}. Since l_1 is even, and $l_2 \geq 2$, by Lemma 3.3.6, the cyclic sequence $r_{\mathcal{L}'}$ has unit δ_m-spread. The δ_m-distance between z' and y'' is therefore 1. For the pair y' and z'' , since they both come from the sequence q_0, with y' being the first element and z'' being the last element, again since the cyclic sequence $r_{\mathcal{L}'}$ has unit δ_m-spread, the δ_m-distance between y' and z'' is also 1. Using claims 1 and 2, we conclude that the cyclic sequence Q has unit δ_m-spread.

We next define the function $h_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$. When $d \geq 3$ and l_1 is an even number, the sequence $h_{\mathcal{L}}$ is $Q'Q''$. To simplify our presentation, we also define the function $h_{\mathcal{L}}$ for the special cases $d = 1$ and $d = 2$. For $d = 2$, we define $h_{\mathcal{L}}$ to be $r_{\mathcal{L}}$. For $d = 1$, we define $h_{\mathcal{L}}$ to be the identity function. (The function $h_{\mathcal{L}}$ with $d = 1$ appears only in the embedding of a ring into a torus, which will be discussed in the next subsection, but not in the embedding of a ring into a mesh.)

Definition 3.3.5 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. The function $h_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows: for all $x \in [n]$,

if $d \geq 3$, then let $\mathcal{L}' = (l_1,\ l_2)$, $\mathcal{L}'' = (l_3,\ l_4, \ldots,\ l_d)$, $m = \prod_{i=3}^{d} l_i$, $a = \lfloor x/(l_1 l_2 - 1) \rfloor$, $b = x \bmod (l_1 l_2 - 1)$, and

$$h_{\mathcal{L}}(x) = \begin{cases} r_{\mathcal{L}'}(b) \diamond f_{\mathcal{L}''}(a), & \text{if } x < m(l_1 l_2 - 1) \text{ and } a \text{ is even;} \\ r_{\mathcal{L}'}(l_1 l_2 - b - 2) \diamond f_{\mathcal{L}''}(a), & \text{if } x < m(l_1 l_2 - 1) \text{ and } a \text{ is odd;} \\ r_{\mathcal{L}'}(l_1 l_2 - 1) \diamond f_{\mathcal{L}''}(n - x - 1), & \text{otherwise;} \end{cases}$$

if $d = 2$, then $\quad h_{\mathcal{L}}(x) = r_{\mathcal{L}}(x)$; $\quad$ and
if $d = 1$, then $\quad h_{\mathcal{L}}(x) = x$. $\quad \square$

In the definition above, $l_1 l_2 - 1$ corresponds to the length of each segment in Q', $m(l_1 l_2 - 1)$ corresponds to the length of the sequence Q', a determines a particular segment in Q', and b determines a particular element inside the segment. An example of the function $h_{\mathcal{L}}$ for $\mathcal{L} = (4, 2, 3)$ is given in Figure 3.9 on page 39.

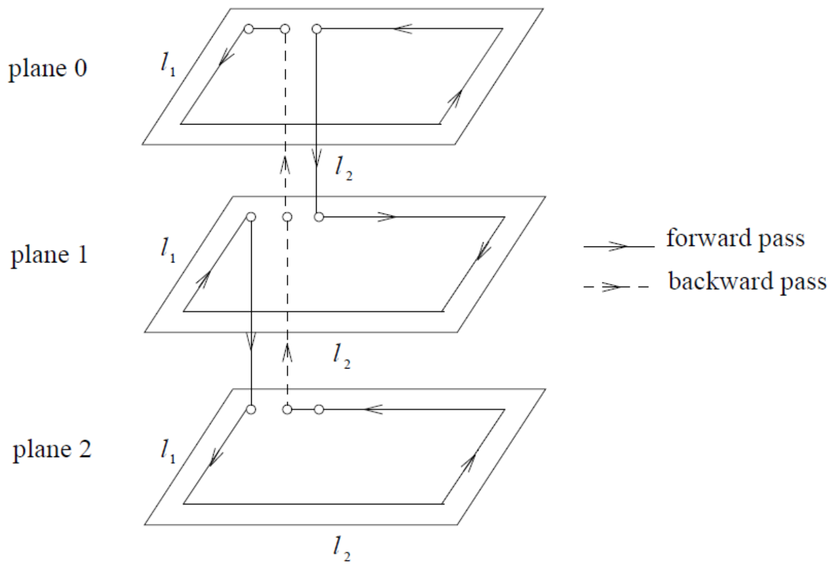


Figure 3.7: Embedding scheme of $h_{\mathcal{L}}$ with $\mathcal{L} = (l_1, l_2, l_3)$ and $l_3 = 3$

The function $h_{\mathcal{L}}$ is clearly bijective. The following lemma follows from the definition of $h_{\mathcal{L}}$ and the properties of $r_{\mathcal{L}'}$ and $f_{\mathcal{L}''}$.

Lemma 3.3.7 *Let $d > 1$, let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. If l_1 is even, then for all $x \in [n]$, $\delta_m(h_{\mathcal{L}}(x), h_{\mathcal{L}}((x+1) \bmod n)) = 1$.* $\square$

We can view the function $h_{\mathcal{L}}$ as embedding a ring into an $(l_1, l_2, \ldots, l_d)$-mesh for which $d \geq 2$ and l_1 is even in the following way. Let $m = \prod_{i=3}^{d} l_i$. We first divide the $(l_1, l_2, \ldots, l_d)$-mesh into m (l_1, l_2)-meshes, which we simply call *planes*. All nodes in each plane have the same rightmost $(d-2)$ components. The values of these components are used to order the planes from 0 to $m-1$ according to the sequence $f_{\mathcal{L}''}(0)$, $f_{\mathcal{L}''}(1)$, …, $f_{\mathcal{L}''}(m-1)$. We refer to the nodes in each plane only by their leftmost two components. The embedding function $h_{\mathcal{L}}$ *marches* through these planes in two passes: first a forward pass from plane 0 to plane $m-1$, and then a backward pass from plane $m-1$ to plane 0. In the forward pass, $h_{\mathcal{L}}$ fills up $l_1 l_2 - 1$ nodes in each plane according to the sequence $r_{\mathcal{L}'}(0)$, $r_{\mathcal{L}'}(1)$, …, $r_{\mathcal{L}'}(l_1 l_2 - 2)$ for even-numbered planes, and according to the sequence $r_{\mathcal{L}'}(l_1 l_2 - 2)$, $r_{\mathcal{L}'}(l_1 l_2 - 3)$, …, $r_{\mathcal{L}'}(0)$ for odd-numbered planes. In the backward pass, $h_{\mathcal{L}}$ fills up the last node $r_{\mathcal{L}'}(l_1 l_2 - 1)$ in each plane. (See Figure 3.7.) An example of an embedding of a ring of size 24 into a $(4, 2, 3)$-mesh using the function $h_{\mathcal{L}}$ is given in Figure 3.10 on page 40.

Given a ring G of even size and an $\mathcal{L}$-mesh H of the same size and of dimension greater than 1, the function $h_{\mathcal{L}}(x)$ gives a unit dilation cost embedding of G into H only if the first component of $\mathcal{L}$ is an even number. If this condition is not satisfied, we can define an $\mathcal{L}^*$-mesh H^* such that $\mathcal{L}^* = (l_1^*, l_2^*, \ldots, l_d^*)$, l_1^* is even, and $\pi(\mathcal{L}^*) = \mathcal{L}$, for some permutation $\pi : [d]^+ \to [d]^+$. (The application of a permutation to a list is defined in Section 3.2 on page 22.) Since H is of even size, $\mathcal{L}^*$ must exist. The ring G can be embedded into H by first embedding G into H^* using $h_{\mathcal{L}^*}$ and then embedding H^* into H using π. For any pair of neighboring nodes A and B in H^*, $\pi(A)$ and $\pi(B)$ remain neighbors in H because π is only a permutation of the lists A and B. Hence, the function $\pi \circ h_{\mathcal{L}^*}$ gives a unit dilation cost embedding of the ring G into the mesh H. ($\circ$ is the function composition operator defined in Section 3.2 on page 22.)

Theorem 3.3.3 *Let G be a ring of even size, and H be an $\mathcal{L}$-mesh of the same size and of dimension d, for $d \geq 2$. Let $\mathcal{L}^*$ be a list such that $\pi(\mathcal{L}^*) = \mathcal{L}$ for some permutation $\pi : [d]^+ \to [d]^+$, and the first component of $\mathcal{L}^*$ is even. The ring G can be embedded into H with unit dilation cost. The function $\pi \circ h_{\mathcal{L}^*}$ gives such an optimal embedding.* $\square$

The next corollary follows from Theorem 3.3.3.

Corollary 3.3.2 *Every mesh of even size and of dimension greater than 1 has a Hamiltonian circuit.* $\square$

Embedding a ring into a torus

By Lemma 3.3.5, if l_1 is even, then $f_{\mathcal{L}}(n) = (l_1 - 1, 0, \ldots, 0)$. In this case, while the δ_m-distance between $f_{\mathcal{L}}(0) = (0, 0, \ldots, 0)$ and $f_{\mathcal{L}}(n-1) = (l_1 - 1, 0, \ldots, 0)$ is $l_1 - 1$, the δ_t-distance between them is 1. On the other hand, if l_1 is odd, then $\lfloor (n-1)/w_1 \rfloor$ (which was shown to be $l_1 - 1$ in the proof of Lemma 3.3.5) is even. It follows that the sublist corresponding to the leftmost two components of $f_{\mathcal{L}}(n-1)$ is $(l_1 - 1, l_2 - 1)$, and thus the δ_t-distance between $f_{\mathcal{L}}(0)$ and $f_{\mathcal{L}}(n-1)$ is greater than 1.

Let G be a ring, and H be an $\mathcal{L}$-torus of the same size and of dimension d. If the size of G and H is even, we can define an $\mathcal{L}^*$-torus H^* such that the first component of $\mathcal{L}^*$ is an even number, and $\pi(\mathcal{L}^*) = \mathcal{L}$ for some permutation $\pi : [d]^* \to [d]^+$. The ring can be embedded into H^* using $f_{\mathcal{L}^*}$, and H^* can be embedded into H using π, both with unit dilation cost. The function $\pi \circ f_{\mathcal{L}^*}$ thus gives a unit dilation cost embedding of G into H. On the other hand, if the size of G and H is odd, then all the components in $\mathcal{L}$ are odd numbers. In this case, we cannot construct a unit dilation cost embedding of G into H in this way because the intermediate graph H^* does not exist.

We now show that the embedding function $h_{\mathcal{L}}$ always embeds a ring into an $\mathcal{L}$-torus of the same size with unit dilation cost, whether their size is even or odd.

Let $\mathcal{L} = (l_1, l_2)$ be a radix-base. While the cyclic sequence $r_{\mathcal{L}}$ has unit δ_m-spread only when l_1 is even, this cyclic sequence always has unit δ_t-spread. When l_1 is odd, $r_{\mathcal{L}}(n-1) = (l_1 - 1, l_2 - 1)$, which is the top node in the last column of a torus. (See Figure 3.8.) Since this node and $r_{\mathcal{L}}(0)$, which is the top node in the first column, are neighbors in a torus, $\delta_t(r_{\mathcal{L}}(0), r_{\mathcal{L}}(n-1)) = 1$. This property is summarized in the following lemma.

Lemma 3.3.8 *Let $\mathcal{L} = (l_1,\ l_2)$ be a radix-base, and let $n = l_1 l_2$. For all $x \in [n]$, $\delta_t(r_{\mathcal{L}}(x), r_{\mathcal{L}}((x+1) \bmod n)) = 1$.* $\square$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $\mathcal{L}' = (l_1, l_2)$. For the case in which $d \geq 2$, since the cyclic sequence $r_{\mathcal{L}'}$ in Definition 3.3.5 always has unit δ_t-spread, whether l_1 is odd or even, the cyclic sequence $h_{\mathcal{L}}$ has unit δ_t-spread. For the case in which $d = 1$,

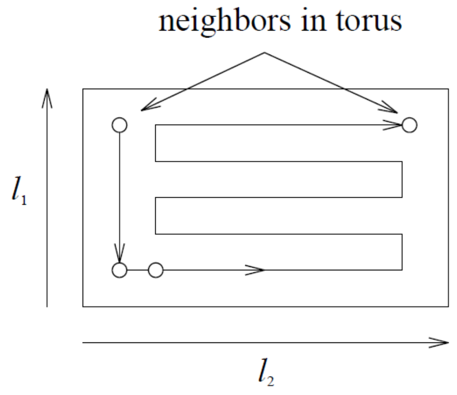


Figure 3.8: The function $r_{\mathcal{L}}$ for odd l_1

the cyclic sequence $h_{\mathcal{L}}$ is 0, 1, …, $n-1$, which also has unit δ_t-spread. The function $h_{\mathcal{L}}$ therefore always provides an optimal, unit dilation cost embedding of a ring into an $\mathcal{L}$-torus. We summarize these results in Lemma 3.3.9 and Theorem 3.3.4.

Lemma 3.3.9 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. For all $x \in [n]$, $\delta_t(h_{\mathcal{L}}(x), h_{\mathcal{L}}((x+1) \bmod n)) = 1$.* □

Theorem 3.3.4 *Let G be a ring, and H be an $\mathcal{L}$-torus of the same size and of dimension d. The ring G can be embedded into H with unit dilation cost. The function $h_{\mathcal{L}}$ gives such an optimal embedding.* □

The next corollary follows from the theorem above.

Corollary 3.3.3 *Every torus has a Hamiltonian circuit.* □

3.4 Generalized embeddings

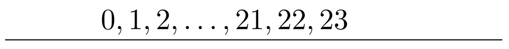
In this section, we study embeddings for which the dimensions of the two graphs are greater than 1. We analyze only the cases in which the shapes of the two graphs satisfy certain conditions: the condition of *expansion* for increasing dimension cases (G has lower dimension than H) and the condition of *reduction* for lowering dimension cases (G has higher dimension than H). The embedding functions for these cases are defined in terms of the basic embedding functions $f_{\mathcal{L}}$, $g_{\mathcal{L}}$, and $h_{\mathcal{L}}$.

Except when G is a torus of even size and H is a mesh, our embeddings for increasing dimension are all optimal. For the exception above, our embeddings can always achieve a dilation cost of 2, and when a certain condition on the shapes of G and H is satisfied, unit dilation cost is also achievable.

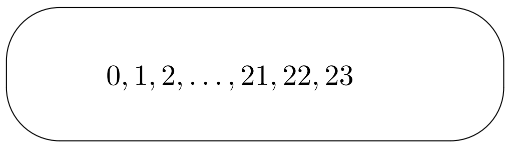
The dilation costs of our embeddings for lowering dimension depend on the shapes of G and H. They are not optimal in general.

x	radix-$\mathcal{L}$ rep. of x	$f_{\mathcal{L}}(x)$	$g_{\mathcal{L}}(x)$	$h_{\mathcal{L}}(x)$
0	(0,0,0)	(0,0,0)	(0,0,0)	(3,0,0)
1	(0,0,1)	(0,0,1)	(0,0,2)	(2,0,0)
2	(0,0,2)	(0,0,2)	(0,1,1)	(1,0,0)
3	(0,1,0)	(0,1,2)	(1,1,0)	(0,0,0)
4	(0,1,1)	(0,1,1)	(1,1,2)	(0,1,0)
5	(0,1,2)	(0,1,0)	(1,0,1)	(1,1,0)
6	(1,0,0)	(1,1,0)	(2,0,0)	(2,1,0)
7	(1,0,1)	(1,1,1)	(2,0,2)	(2,1,1)
8	(1,0,2)	(1,1,2)	(2,1,1)	(1,1,1)
9	(1,1,0)	(1,0,2)	(3,1,0)	(0,1,1)
10	(1,1,1)	(1,0,1)	(3,1,2)	(0,0,1)
11	(1,1,2)	(1,0,0)	(3,0,1)	(1,0,1)
12	(2,0,0)	(2,0,0)	(3,0,0)	(2,0,1)
13	(2,0,1)	(2,0,1)	(3,0,2)	(3,0,1)
14	(2,0,2)	(2,0,2)	(3,1,1)	(3,0,2)
15	(2,1,0)	(2,1,2)	(2,1,0)	(2,0,2)
16	(2,1,1)	(2,1,1)	(2,1,2)	(1,0,2)
17	(2,1,2)	(2,1,0)	(2,0,1)	(0,0,2)
18	(3,0,0)	(3,1,0)	(1,0,0)	(0,1,2)
19	(3,0,1)	(3,1,1)	(1,0,2)	(1,1,2)
20	(3,0,2)	(3,1,2)	(1,1,1)	(2,1,2)
21	(3,1,0)	(3,0,2)	(0,1,0)	(3,1,2)
22	(3,1,1)	(3,0,1)	(0,1,2)	(3,1,1)
23	(3,1,2)	(3,0,0)	(0,0,1)	(3,1,0)

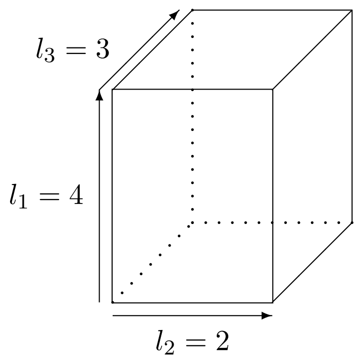
Figure 3.9: Embedding functions $f_{\mathcal{L}}$, $g_{\mathcal{L}}$, and $h_{\mathcal{L}}$ for $n = 24$ and $\mathcal{L} = (4, 2, 3)$



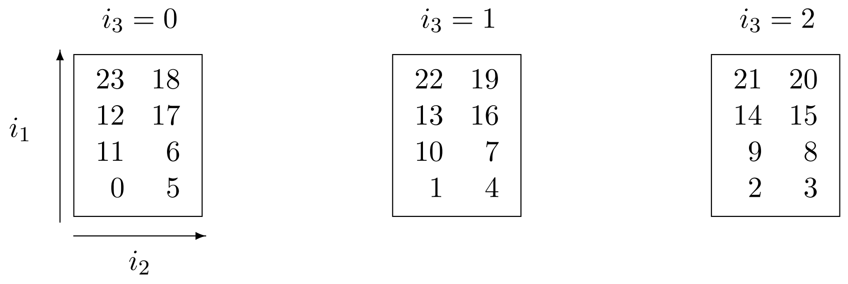
(a) A line of size 24



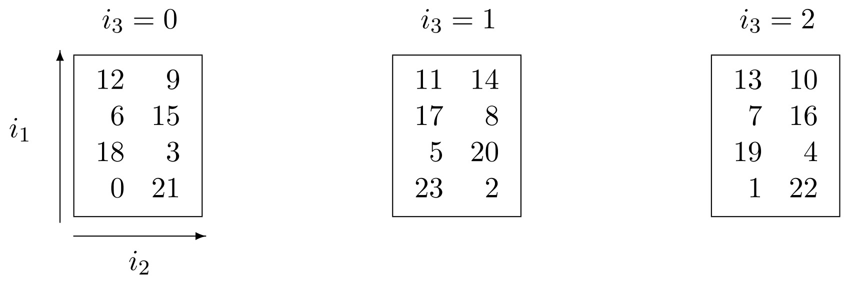
(b) A ring of size 24



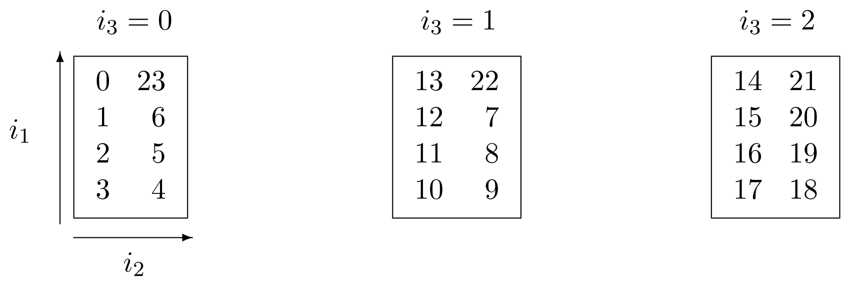
(c) A (4,2,3)-mesh



(d) Embedding the line into the mesh using $f_{(4,2,3)}$



(e) Embedding the ring into the mesh using $g_{(4,2,3)}$



(f) Embedding the ring into the mesh using $h_{(4,2,3)}$

Figure 3.10: Embedding a line or a ring of size 24 into a $(4, 2, 3)$-mesh

3.4.1 Embeddings for increasing dimension

Given a list $\mathcal{A} = (a_1, a_2, \ldots, a_k)$, we use $\prod \mathcal{A}$ to denote the product $a_1 a_2 \cdots a_k$.

Definition 3.4.1 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be lists of positive integers for which $d < c$. The list $\mathcal{M}$ is an *expansion* of the list $\mathcal{L}$ if there exist d lists of integers $\mathcal{V}_1$, $\mathcal{V}_2$, …, $\mathcal{V}_d$ such that (i) for all $i \in [d]^+$, $\prod \mathcal{V}_i = l_i$; and (ii) the list $\mathcal{M}$ is a permutation of the list $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \diamond \cdots \diamond \mathcal{V}_d$. We call $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$ an *expansion factor* of $\mathcal{L}$ into $\mathcal{M}$. □

For example, the list $\mathcal{M} = (2, 4, 3, 8, 5, 4)$ is an expansion of the list $\mathcal{L} = (6, 8, 80)$ because we can have $\mathcal{V}_1 = (2, 3)$, $\mathcal{V}_2 = (8)$, and $\mathcal{V}_3 = (4, 5, 4)$. The list $\mathcal{V} = ((2, 3), (8), (4, 5, 4))$ is an expansion factor of $\mathcal{L}$ into $\mathcal{M}$. Expansion factors may not be unique: the list $((3, 2), (8), (5, 4, 4))$ is also an expansion factor of $\mathcal{L}$ into $\mathcal{M}$.

Let G be either a torus or a mesh of shape $\mathcal{L}$, and let H be either a torus or a mesh of shape $\mathcal{M}$ such that $\mathcal{M}$ is an expansion of $\mathcal{L}$ with an expansion factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_d$, and let H' be a graph of shape $\tilde{\mathcal{V}}$ and of the same type as H. (*type* of a graph is defined in Section 3.2 on page 23.) We now construct an embedding of G into H in two steps: $G \to H' \to H$.

Let $\pi : [c]^+ \to [c]^+$ be a permutation such that $\pi(\tilde{\mathcal{V}}) = \mathcal{M}$. By the definition of expansion, such a permutation always exists. Since H' has shape $\tilde{\mathcal{V}}$ and H has shape $\mathcal{M}$, H' can be embedded into H with unit dilation cost using the permutation π. Next we construct an embedding of G into H'.

We first consider the case in which G and H' are both meshes. We map each node $(i_1, i_2, \ldots, i_d)$ in G to the node $f_{\mathcal{V}_1}(i_1) \diamond f_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond f_{\mathcal{V}_d}(i_d)$ in H'. Since the functions $f_{\mathcal{V}_1} : [l_1] \to \Omega_{\mathcal{V}_1}$, $f_{\mathcal{V}_2} : [l_2] \to \Omega_{\mathcal{V}_2}$, …, $f_{\mathcal{V}_d} : [l_d] \to \Omega_{\mathcal{V}_d}$ are all bijective, this mapping is an embedding of G into H'. For every pair of neighboring nodes $(i_1, i_2, \ldots, i_d)$ and $(i'_1, i'_2, \ldots, i'_d)$ in G, by definition, there exists exactly one index $k \in [d]^+$ such that $|i_k - i'_k| = 1$ and $i_j = i'_j$, for all $j \in [d]^+$ such that $j \neq k$. Since the sequences $f_{\mathcal{V}_1}$, $f_{\mathcal{V}_2}$, …, $f_{\mathcal{V}_d}$ all have unit δ_m-spread, we have $\delta_m(f_{\mathcal{V}_k}(i_k), f_{\mathcal{V}_k}(i'_k)) = 1$, and $\delta_m(f_{\mathcal{V}_j}(i_j), f_{\mathcal{V}_j}(i'_j)) = 0$, for all $j \in [d]^+$ such that $j \neq k$. The nodes $f_{\mathcal{V}_1}(i_1) \diamond f_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond f_{\mathcal{V}_d}(i_d)$ and $f_{\mathcal{V}_1}(i'_1) \diamond f_{\mathcal{V}_2}(i'_2) \diamond \cdots \diamond f_{\mathcal{V}_d}(i'_d)$ thus have unit δ_m-distance in H', and hence must be neighbors in H'. This embedding therefore has unit dilation cost. Furthermore, since the sequences $f_{\mathcal{V}_1}$, $f_{\mathcal{V}_2}$, …, $f_{\mathcal{V}_d}$ all have unit δ_t-spread. This embedding also has unit dilation cost when G is a mesh and H' is a torus.

When G is a torus and H' is a mesh, we can define a similar embedding by replacing the functions $f_{\mathcal{V}_1}$, $f_{\mathcal{V}_2}$, …, $f_{\mathcal{V}_d}$ with the functions $g_{\mathcal{V}_1}$, $g_{\mathcal{V}_2}$, …, $g_{\mathcal{V}_d}$. Since the cyclic sequences $g_{\mathcal{V}_1}$, $g_{\mathcal{V}_2}$, …, $g_{\mathcal{V}_d}$ all have a δ_m-spread of 2, by a similar argument, we can show that the embedding has a dilation cost of 2.

For the remaining case in which G and H' are both toruses, we can construct a similar embedding by replacing $f_{\mathcal{V}_1}$, $f_{\mathcal{V}_2}$, …, $f_{\mathcal{V}_d}$ with $h_{\mathcal{V}_1}$, $h_{\mathcal{V}_2}$, …, $h_{\mathcal{V}_d}$. Since the cyclic sequences $h_{\mathcal{V}_1}$, $h_{\mathcal{V}_2}$, …, $h_{\mathcal{V}_d}$ all have unit δ_t-spread, the embedding also has unit dilation cost.

The sequence of embeddings $G \to H' \to H$ described above gives an embedding of G into H with a dilation cost of 2 if G is a torus and H is a mesh, and with unit dilation cost otherwise.

As will be proved in Theorem 3.4.1, when G is a torus and H is a mesh, a dilation cost of 2 is optimal for all G of odd size. On the other hand, if each dimension of G

has even length and there is at least one expansion factor of $\mathcal{L}$ into $\mathcal{M}$ such that each list in the factor has at least two components, then we can choose an expansion factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$ of $\mathcal{L}$ into $\mathcal{M}$ such that for all $i \in [d]^+$, $\mathcal{V}_i$ has length at least 2, and its first component is an even number. If we use such an expansion factor $\mathcal{V}$ to define the shape of H', then by Lemma 3.3.7, G can be embedded into H' with unit dilation cost by mapping each node $(i_1, i_2, \ldots, i_d)$ in G to the node $h_{\mathcal{V}_1}(i_1) \diamond h_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond h_{\mathcal{V}_d}(i_d)$ in H'. Such an embedding sequence $G \to H' \to H$ gives a unit dilatoin cost embedding of G into H.

For example, if $\mathcal{L} = (6, 12)$ and $\mathcal{M} = (6, 3, 2, 2)$, then both $((6), (3, 2, 2))$ and $((2, 3), (6, 2))$ are expansion factors of $\mathcal{L}$ into $\mathcal{M}$. If we choose the expansion factor $((2, 3), (6, 2))$ to define the shape of H', then we get a unit dilation cost embedding of a $(6, 12)$-torus G into a $(6, 3, 2, 2)$-mesh H. On the other hand, if we choose $((6), (3, 2, 2))$ to define the shape of H', then we get an embedding with a dilation cost of 2.

We formalize the above results in the following definition and theorems.

Definition 3.4.2 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases such that $\mathcal{M}$ is an expansion of $\mathcal{L}$ with an expansion factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \diamond \cdots \diamond \mathcal{V}_d$. The functions $\mathcal{F}_{\mathcal{V}}\colon \Omega_{\mathcal{L}} \to \Omega_{\tilde{\mathcal{V}}}$, $\mathcal{G}_{\mathcal{V}}\colon \Omega_{\mathcal{L}} \to \Omega_{\tilde{\mathcal{V}}}$ and $\mathcal{H}_{\mathcal{V}}\colon \Omega_{\mathcal{L}} \to \Omega_{\tilde{\mathcal{V}}}$ are defined as follows: for all $(i_1,\ i_2,\ \ldots,\ i_d) \in \Omega_{\mathcal{L}}$,

$$\mathcal{F}_{\mathcal{V}}((i_1,\ i_2,\ \ldots,\ i_d)) = f_{\mathcal{V}_1}(i_1) \diamond f_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond\ f_{\mathcal{V}_d}(i_d),$$

$$\mathcal{G}_{\mathcal{V}}((i_1,\ i_2,\ \ldots,\ i_d)) = g_{\mathcal{V}_1}(i_1) \diamond g_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond\ g_{\mathcal{V}_d}(i_d),$$

$$\mathcal{H}_{\mathcal{V}}((i_1,\ i_2,\ \ldots,\ i_d)) = h_{\mathcal{V}_1}(i_1) \diamond h_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond\ h_{\mathcal{V}_d}(i_d).$$

Furthermore, let $\pi : [c]^+ \to [c]^+$ be a permutation such that $\pi(\tilde{\mathcal{V}}) = \mathcal{M}$. Then we have the functions $\pi \circ \mathcal{F}_{\mathcal{V}} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$, $\pi \circ \mathcal{G}_{\mathcal{V}} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$, and $\pi \circ \mathcal{H}_{\mathcal{V}} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$. □

Examples of the functions $\mathcal{F}_{\mathcal{V}}$, $\mathcal{G}_{\mathcal{V}}$, and $\mathcal{H}_{\mathcal{V}}$ for $\mathcal{L} = (4, 6)$, $\mathcal{M} = (2, 2, 2, 3)$, and $\mathcal{V} = ((2, 2), (2, 3))$ are given in Figure 3.11. In this example, we have $\mathcal{M} = \mathcal{V}_1 \diamond \mathcal{V}_2$.

Theorem 3.4.1 *Let G be either an $(l_1, l_2, \ldots, l_d)$-torus or an $(l_1, l_2, \ldots, l_d)$-mesh, and let H be either an $(m_1, m_2, \ldots, m_c)$-torus or an $(m_1, m_2, \ldots, m_c)$-mesh. Assume that $(m_1, m_2, \ldots, m_c)$ is an expansion of $(l_1, l_2, \ldots, l_d)$ with an expansion factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$. Let $\pi : [c]^+ \to [c]^+$ be a permutation such that $\pi(\mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_d) = (m_1, m_2, \ldots, m_c)$. Then*

(a) *If G is a mesh, then G can be embedded into H with unit dilation cost. The function $\pi \circ \mathcal{F}_{\mathcal{V}}$ gives such an optimal embedding.*

(b) *If G and H are both toruses, then G can be embedded into H with unit dilation cost. The function $\pi \circ \mathcal{H}_{\mathcal{V}}$ gives such an optimal embedding.*

(c) *If G is a torus and H is a mesh, then G can be embedded into H with a dilation cost of 2. The function $\pi \circ \mathcal{G}_{\mathcal{V}}$ gives such an embedding. Furthermore, such an embedding is optimal for all G of odd size. If G is of even size, and for all $i \in [d]^+$, $\mathcal{V}_i$ consists of at least two components such that the first component is an even number, then G can be embedded into H with unit dilation cost. The function $\pi \circ \mathcal{H}_{\mathcal{V}}$ gives such an optimal embedding.*

(i_1, i_2)	$\Omega_{(2,2)} \diamond \Omega_{(2,3)}$	$\mathcal{F}_{\mathcal{V}} = f_{(2,2)} \diamond f_{(2,3)}$	$\mathcal{G}_{\mathcal{V}} = g_{(2,2)} \diamond g_{(2,3)}$	$\mathcal{H}_{\mathcal{V}} = h_{(2,2)} \diamond h_{(2,3)}$
(0,0)	(0,0,0,0)	(0,0,0,0)	(0,0,0,0)	(0,0,1,0)
(0,1)	(0,0,0,1)	(0,0,0,1)	(0,0,0,2)	(0,0,0,0)
(0,2)	(0,0,0,2)	(0,0,0,2)	(0,0,1,1)	(0,0,0,1)
(0,3)	(0,0,1,0)	(0,0,1,2)	(0,0,1,0)	(0,0,0,2)
(0,4)	(0,0,1,1)	(0,0,1,1)	(0,0,1,2)	(0,0,1,2)
(0,5)	(0,0,1,2)	(0,0,1,0)	(0,0,0,1)	(0,0,1,1)
(1,0)	(0,1,0,0)	(0,1,1,0)	(1,1,0,0)	(1,0,1,0)
(1,1)	(0,1,0,1)	(0,1,1,1)	(1,1,0,2)	(1,0,0,0)
(1,2)	(0,1,0,2)	(0,1,1,2)	(1,1,1,1)	(1,0,0,1)
(1,3)	(0,1,1,0)	(0,1,0,2)	(1,1,1,0)	(1,0,0,2)
(1,4)	(0,1,1,1)	(0,1,0,1)	(1,1,1,2)	(1,0,1,2)
(1,5)	(0,1,1,2)	(0,1,0,0)	(1,1,0,1)	(1,0,1,1)
(2,0)	(1,0,0,0)	(1,1,0,0)	(1,0,0,0)	(1,1,1,0)
(2,1)	(1,0,0,1)	(1,1,0,1)	(1,0,0,2)	(1,1,0,0)
(2,2)	(1,0,0,2)	(1,1,0,2)	(1,0,1,1)	(1,1,0,1)
(2,3)	(1,0,1,0)	(1,1,1,2)	(1,0,1,0)	(1,1,0,2)
(2,4)	(1,0,1,1)	(1,1,1,1)	(1,0,1,2)	(1,1,1,2)
(2,5)	(1,0,1,2)	(1,1,1,0)	(1,0,0,1)	(1,1,1,1)
(3,0)	(1,1,0,0)	(1,0,1,0)	(0,1,0,0)	(0,1,1,0)
(3,1)	(1,1,0,1)	(1,0,1,1)	(0,1,0,2)	(0,1,0,0)
(3,2)	(1,1,0,2)	(1,0,1,2)	(0,1,1,1)	(0,1,0,1)
(3,3)	(1,1,1,0)	(1,0,0,2)	(0,1,1,0)	(0,1,0,2)
(3,4)	(1,1,1,1)	(1,0,0,1)	(0,1,1,2)	(0,1,1,2)
(3,5)	(1,1,1,2)	(1,0,0,0)	(0,1,0,1)	(0,1,1,1)

Figure 3.11: Embedding functions $\mathcal{F}_{\mathcal{V}}$, $\mathcal{G}_{\mathcal{V}}$, $\mathcal{H}_{\mathcal{V}}$ for $\mathcal{L} = (4,6)$, $\mathcal{M} = (2,2,2,3)$, and $\mathcal{V} = ((2,2),(2,3))$

□

Proof. We prove only the claim in (iii) that $\mathcal{G}_{\mathcal{V}}$ is optimal for all toruses of odd sizes. We prove this by showing that such a torus cannot be embedded into a mesh with unit dilation cost. The other parts of the theorem follow from the definitions of $\mathcal{F}_{\mathcal{V}}$, $\mathcal{G}_{\mathcal{V}}$, and $\mathcal{H}_{\mathcal{V}}$.

Assume for contradiction that a torus G of odd size can be embedded into a mesh H with unit dilation cost. Let p be such an embedding. Since G is a torus, by Corollary 3.3.3, there exists at least one Hamiltonian circuit $v_0 - v_1 - \cdots - v_{n-1} - v_n$ $(= v_0)$ in G. By the definition of a Hamiltonian circuit, for all $i \in \{0, \ldots, n-1\}$, v_i and v_{i+1} are neighbors in G. Since the embedding p has unit dilation cost, $p(v_i)$ and $p(v_{i+1})$ must also be neighbors in H. This implies that the path $p(v_0)-p(v_1)-\cdots-p(v_{n-1})-p(v_n)$ $(= p(v_0))$ is a Hamiltonian circuit in H, contradicting the fact that no mesh of odd size has a Hamiltonian circuit (Corollary 3.3.1). □

The embeddings for increasing dimension given in this subsection can be applied only if the shapes of the two graphs satisfy the condition of expansion. The next theorem states that if H is a hypercube, then the shapes of G and H always satisfy the condition of expansion.

Theorem 3.4.2 *Let G be either a torus or a mesh, and let H be a hypercube of the same size. Then the shape of H is an expansion of the shape of G.* □

Proof. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be the shape of G, and $\mathcal{M}$ be the shape of H. By the definition of hypercube, G and H must both be of size some power of 2. Hence, for all $k \in [d]^+$, $l_k = 2^{q_k}$, for some positive integer q_k. Since G and H are of the same size, $2^{q_1}2^{q_2}\cdots2^{q_d}$ is the size of H. The list $\mathcal{M}$ is thus an expansion of the list $\mathcal{L}$ with an expansion factor

$$((\underbrace{2,2,\ldots,2}_{q_1}), (\underbrace{2,2,\ldots,2}_{q_2}), \ldots, (\underbrace{2,2,\ldots,2}_{q_d})).$$

□

By viewing a hypercube as a special case of a torus, the next corollary follows directly from Theorems 3.4.1 and 3.4.2. This corollary was proved in [CS86].

Corollary 3.4.1 *A torus or a mesh can be embedded into a hypercube of the same size with unit dilation cost.* □

3.4.2 Embeddings for lowering dimension

Our embeddings for lowering dimension are defined using two types of embeddings: embeddings for increasing dimension (from preceding subsection) and embeddings among toruses and meshes of the same *shape*.

Given a torus or a mesh G and a torus or a mesh H of the same shape $(l_1, l_2, \ldots, l_d)$, G can be embedded into H with unit dilation cost using the identity function, except when G is a torus, H is a mesh, and neither is a hypercube. In this exceptional case, G clearly cannot be embedded into H with unit dilation cost because each boundary node in H has degree less than that of any node in G. An optimal embedding of G into H with a dilation cost of 2 can be constructed by embedding each node $(i_1, i_2, \ldots, i_d)$ of G into the node

$(t_{l_1}(i_1), t_{l_2}(i_2), \ldots, t_{l_d}(i_d))$ of H. Since for all $i \in [d]^+$, the function $t_{l_i} : [l_i] \rightarrow [l_i]$ defines a cyclic sequence of all numbers in $[l_i]$ with a δ_m-spread of 2 (Definition 3.3.2), every two neighboring nodes in G are mapped to nodes in H at a distance no greater than 2. This embedding thus has a dilation cost of 2. The following definition and lemma summarize these results.

Definition 3.4.3 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base. The function $\mathcal{T}_{\mathcal{L}} : \Omega_{\mathcal{L}} \rightarrow \Omega_{\mathcal{L}}$ is defined as follows: for all $(x_1, x_2, \ldots, x_d) \in \Omega_{\mathcal{L}}$,

$$\mathcal{T}_{\mathcal{L}}((x_1, x_2, \ldots, x_d)) = (t_{l_1}(x_1), t_{l_2}(x_2), \ldots, t_{l_d}(x_d)).$$

$\square$

Lemma 3.4.1 *Let G be a torus or a mesh of shape $\mathcal{L}$, and let H be a torus or a mesh of the same shape. If G is a torus, H is a mesh, and neither G nor H is a hypercube, then G can be embedded into H with an optimal dilation cost of 2 using the embedding function $\mathcal{T}_{\mathcal{L}}$. Otherwise, G can be embedded into H with unit dilation cost using the identity function.* $\square$

For lowering dimension, we consider only those cases where the shapes of G and H satisfy the condition of reduction. We define two types of reduction: (i) simple reduction and (ii) general reduction.

Simple reduction

Definition 3.4.4 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be lists of positive integers for which $d > c$. The list $\mathcal{M}$ is a *simple reduction* of the list $\mathcal{L}$ with a reduction factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_c)$ if $\mathcal{L}$ is an expansion of $\mathcal{M}$ with an expansion factor $\mathcal{V}$. $\square$

Let $\mathcal{L}$ be a radix-base. We next define a function that will be used to construct our embeddings. This function is defined in terms of the function $u_{\mathcal{L}}^{-1}$, which maps each mixed-radix number in $\Omega_{\mathcal{L}}$ to the corresponding natural number in $[|\Omega_{\mathcal{L}}|]$, defined on page 25, Section 3.2.

Definition 3.4.5 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases such that $\mathcal{M}$ is a simple reduction of $\mathcal{L}$ with a reduction factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_c)$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_c$. For all $k \in [c]^+$, let $u_{\mathcal{V}_k}^{-1} : \Omega_{\mathcal{V}_k} \rightarrow [m_k]$. The function $\mathcal{U}_{\mathcal{V}} : \Omega_{\tilde{\mathcal{V}}} \rightarrow \Omega_{\mathcal{M}}$ is defined as follows: for all $(i_1,\ i_2,\ \ldots,\ i_d) \in \Omega_{\tilde{\mathcal{V}}}$,

$$\mathcal{U}_{\mathcal{V}}((i_1,\ i_2,\ \ldots,\ i_d)) = u_{\mathcal{V}_1}^{-1}(I_1) \diamond u_{\mathcal{V}_2}^{-1}(I_2) \diamond \cdots \diamond\ u_{\mathcal{V}_c}^{-1}(I_c),$$

where I_1, I_2, ..., I_c are partitions of $(i_1, i_2, \ldots, i_d)$ such that for all $k \in [c]^+$, $|I_k| = |\mathcal{V}_k|$, and $I_1 \diamond I_2 \diamond \cdots \diamond I_k = (i_1, i_2, \ldots, i_d)$. Furthermore, let $\pi : [d]^+ \rightarrow [d]^+$ be a permutation such that $\pi(\mathcal{L}) = (\tilde{\mathcal{V}})$. Then we have the function $\mathcal{U}_{\mathcal{V}} \circ \pi : \Omega_{\mathcal{L}} \rightarrow \Omega_{\mathcal{M}}$. $\square$

Let G be either a torus or a mesh with shape $\mathcal{L}$, and let H be either a torus or a mesh with shape $\mathcal{M}$ such that $\mathcal{M}$ is a simple reduction of $\mathcal{L}$. Let $\mathcal{V}$ be a reduction factor of $\mathcal{L}$ into $\mathcal{M}$ such that for all $i \in [c]^+$, the elements in the list $\mathcal{V}_i$ are in non-increasing order. Let v_i denote the index in $[d]^+$ such that l_{v_i} is the first element in $\mathcal{V}_i$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_c$, and let G' be a graph with shape $\tilde{\mathcal{V}}$ and of the same type of graph as G. Let $\pi : [d]^+ \to [d]^+$ be a permutation such that $\pi(\mathcal{L}) = \tilde{\mathcal{V}}$. The graph G can be embedded into G' using the permutation π with unit dilation cost. We next construct an embedding of G' into H.

Let $A = I_1 \diamond I_2 \diamond \cdots \diamond I_k \diamond \cdots \diamond I_c$ and $B = I'_1 \diamond I'_2 \diamond \cdots \diamond I'_k \diamond \cdots \diamond I'_c$ be an arbitrary pair of neighboring nodes in G', where for all $i \in [c]^+$, $|I_i| = |I'_i| = |\mathcal{V}_i|$. Let $q = |\mathcal{V}_k|$, and $(l'_1, l'_2, \ldots, l'_q) = \mathcal{V}_k$. Here $l'_1 = l_{v_k}$. Without loss of generality, assume that A and B differ at the r-th position in I_k, for some $r \in [q]^+$. Let i_r and i'_r denote respectively the components of A and B at this position.

We first consider the case in which both G' and H are meshes. We use the function $\mathcal{U}_{\mathcal{V}}$ to embed G' into H. The distance between the images of A and B in H is $\delta_m(\mathcal{U}_{\mathcal{V}}(A), \mathcal{U}_{\mathcal{V}}(B)) = |u^{-1}_{\mathcal{V}_k}(I_k) - u^{-1}_{\mathcal{V}_k}(I'_k)| = |i_r - i'_r| \prod_{j=r+1}^{q} l'_j$. Since G' is a mesh, $|i_r - i'_r| = 1$, and since $m_k = \prod_{j=1}^{q} l'_j$, we have $\delta_m(\mathcal{U}_{\mathcal{V}}(A), \mathcal{U}_{\mathcal{V}}(B)) = m_k / \prod_{j=1}^{r} l'_j \le m_k / l'_1$. Therefore, the function $\mathcal{U}_{\mathcal{V}}$ gives an embedding of G' into H with a dilation cost of $\max_{1 \le i \le c}\{m_i / l_{v_i}\}$.

For the cases in which either (i) G' is a mesh and H is a torus or (ii) both G' and H are toruses, we use the same embedding function $\mathcal{U}_{\mathcal{V}}$ to embed G' into H. The distance between the images of A and B in H is $\delta_t(\mathcal{U}_{\mathcal{V}}(A), \mathcal{U}_{\mathcal{V}}(B)) = \min\{|i_r - i'_r| \prod_{j=r+1}^{q} l'_j, m_k - |i_r - i'_r| \prod_{j=r+1}^{q} l'_j\}$. For case (i), $|i_r - i'_r|$ is 1 and for case (ii) $|i_r - i'_r|$ is either 1 or $l'_r - 1$. In either case, using the fact that for all $j \in [q]^+$, $l'_j \ge 2$, we can show that the embedding also has a dilation cost of $\max_{1 \le i \le c}\{m_i / l_{v_i}\}$.

For the remaining case in which G' is a torus and H is a mesh, using the embedding function $\mathcal{T}_{\tilde{\mathcal{V}}}$, we first embed G' into an intermediate mesh G'' that has the same shape as G'. Such an embedding has a dilatin cost of 2. We then embed the mesh G'' into the mesh H using the function $\mathcal{U}_{\mathcal{V}}$. This sequence gives an embedding of G' into H with a dilation cost of $2 \max_{1 \le i \le c}\{m_i / l_{v_i}\}$.

Theorem 3.4.3 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases, G be either an $\mathcal{L}$-mesh or an $\mathcal{L}$-torus, and H be either an $\mathcal{M}$-mesh or an $\mathcal{M}$-torus such that $\mathcal{M}$ is a simple reduction of $\mathcal{L}$. Let $\mathcal{V}$ be a reduction factor of $\mathcal{L}$ into $\mathcal{M}$ such that for all $i \in [c]^+$, the elements in the list $\mathcal{V}_i$ are in non-increasing order. Let v_i denote the index in $[d]^+$ such that l_{v_i} is the first element in $\mathcal{V}_i$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \diamond \cdots \diamond \mathcal{V}_c$. Let $\pi : [d]^+ \to [d]^+$ be a permutation such that $\pi(\mathcal{L}) = \tilde{\mathcal{V}}$. If G is a torus and H is a mesh, then G can be embedded into H with a dilation cost of $2 \max_{1 \le i \le c}\{m_i / l_{v_i}\}$, and the function $\mathcal{U}_{\mathcal{V}} \circ \mathcal{T}_{\tilde{\mathcal{V}}} \circ \pi$ gives such an embedding; otherwise, G can be embedded into H with a dilation cost of $\max_{1 \le i \le c}\{m_i / l_{v_i}\}$, and the function $\mathcal{U}_{\mathcal{V}} \circ \pi$ gives such an embedding.* $\square$

The next corollary follows from the property that for the special case in which G is a hypercube, the shapes of G and H always satisfy the condition of simple reduction.

Corollary 3.4.2 *A hypercube can be embedded into an $(m_1, m_2, \ldots, m_c)$-torus or an $(m_1, m_2, \ldots, m_c)$-mesh of the same size with a dilation cost of* $\max\{m_1, m_2, \ldots, m_c\}/2$. $\square$

Proof. Let G be a hypercube of size 2^d, for some positive integer d. Let H be a $(m_1, m_2, \ldots, m_c)$-torus or a $(m_1, m_2, \ldots, m_c)$-mesh of the same size as G. Since G and H have the same size, we have $\prod_{i=1}^{c} m_i = 2^d$, and hence for all $i \in [c]^+$, $m_i = 2^{b_i}$ for some positive integer b_i. Therefore, H is a simple reduction of G with a reduction factor

$((\underbrace{2,\ldots,2}_{b_1}), (\underbrace{2,\ldots,2}_{b_2}), \cdots (\underbrace{2,\ldots,2}_{b_c}))$. Since a hypercube is a special case of a mesh, by Theorem 3.4.3, G can be embedded into H with a dilation cost of $\max\{m_1, m_2, \ldots, m_c\}/2$. □

General reduction

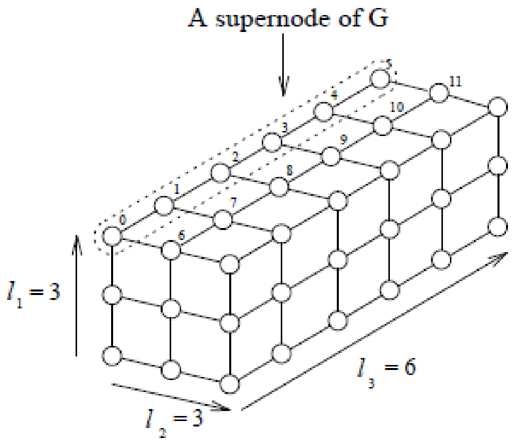
We first illustrate through a simple example the embeddings to be constructed under general reduction. Let G be a $(3,3,6)$-mesh, and H be a $(6,9)$-mesh. We can view G as a $(3,3)$-mesh of *supernodes*, each of which is a line of length 6, and view H as a $(3,3)$-mesh of *supernodes*, each of which is a $(2,3)$-mesh. (See Figure 3.12.) With respect to supernodes, G and H have the same shape: a $(3,3)$-mesh. With the identity function, neighboring supernodes of G can be embedded into neighboring supernodes of H. Since the supernodes of G are lines of length 6, and the supernodes of H are $(2,3)$-meshes, the nodes belonging to a single supernode of G can be embedded into the nodes belonging to the corresponding supernode of H by using the embedding function $f_{(2,3)}$. This embedding of G into H is achieved by embedding nine separate lines of length 6 into nine separate $(2,3)$-meshes, with neighboring lines embedded into neighboring meshes. Such an embedding gives a dilation cost of 3.

In general, given a torus or a mesh G and a torus or a mesh H whose dimension is at least half of the dimension of G and whose shape is a *general reduction* (to be defined below) of the shape of G, G and H can be viewed as graphs of some supernodes such that (i) with respect to supernodes, G and H have the same shape; and (ii) the shape of the supernodes of H is an expansion of the shape of the supernodes of G. An embedding of G into H can be achieved as follows: first establish a one-to-one correspondence between the supernodes of G and the supernodes of H, and then by using the embedding functions for increasing dimension defined in the last subsection, embed the nodes belonging to a single supernode of G into the nodes belonging to the corresponding supernode of H.

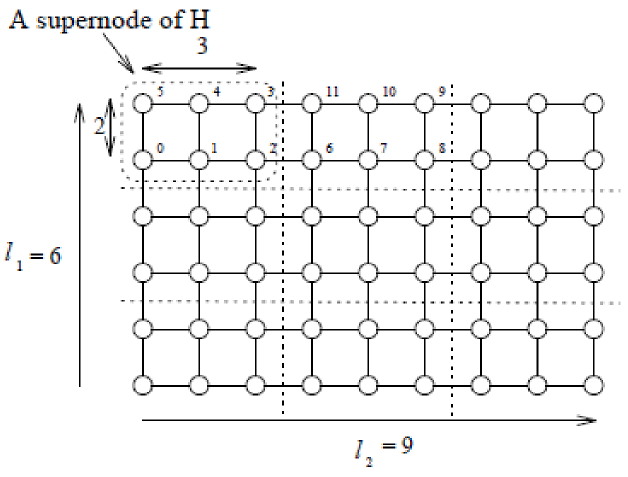
We now define the relation *general reduction* between two lists of different lengths where the length of the shorter list is at least half of the length of the longer list. Given a list $\mathcal{A} = (a_1, a_2, \ldots, a_k)$ and a list $\mathcal{B} = (b_1, b_2, \ldots, b_k)$, we use $\mathcal{A} \times \mathcal{B}$ to denote the list $(a_1b_1, a_2b_2, \ldots, a_kb_k)$ and $\mathcal{A} + \mathcal{B}$ to denote the list $(a_1 + b_1, a_2 + b_2, \ldots, a_k + b_k)$. We use [ ] for grouping.

Definition 3.4.6 Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be lists of positive integers for which $c < d < 2c$. The list $\mathcal{M}$ is a *general reduction* of the list $\mathcal{L}$ if (i) there exist a list $\mathcal{L}'$ of length c and a list $\mathcal{L}''$ of length $d - c$ such that $\mathcal{L}$ is a permutation of the list $\mathcal{L}' \diamond \mathcal{L}''$; (ii) there exist $d - c$ lists $\mathcal{S}_1$, $\mathcal{S}_2$, …, $\mathcal{S}_{d-c}$, the components of each of which are integers all greater than 1, such that the list $\mathcal{L}''$ is $(\prod \mathcal{S}_1, \prod \mathcal{S}_2, \ldots, \prod \mathcal{S}_{d-c})$ and the list $\tilde{\mathcal{S}} = \mathcal{S}_1 \diamond \mathcal{S}_2 \ldots \diamond \mathcal{S}_{d-c}$ has length b, where $d - c < b \leq c$; and (iii) $\mathcal{M}$ is a permutation of the list $[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'$, where $\mathcal{I} = (\underbrace{1, 1, \ldots, 1}_{c-b})$. We call $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$ a *reduction factor* of $\mathcal{L}$ into $\mathcal{M}$, $\mathcal{L}'$ the *multiplicant sublist*, and $\mathcal{L}''$ the *multiplier sublist.* □

For example, the list $\mathcal{M} = (4,3,5,28,10,18)$ is a general reduction of the list $\mathcal{L} = (2,3,2,10,6,21,5,4)$ because we can choose $\mathcal{L}' = (2,2,6,4,3,5)$, $\mathcal{L}'' = (10,21)$, $\mathcal{S}_1 = (5,2)$, and $\mathcal{S}_2 = (3,7)$. The list $[\tilde{\mathcal{S}} \diamond (1,1)] \times \mathcal{L}' = (10,4,18,28,3,5)$ is a permutation of $\mathcal{M}$. The



(a) (3, 3, 6)-mesh G



(b) (6, 9)-mesh H

Figure 3.12: Supernode view

list $\mathcal{S} = ((5,2),(3,7))$ is a reduction factor of $\mathcal{L}$ into $\mathcal{M}$. Reduction factors may not be unique: the list $((2,5),(3,7))$ is also a reduction factor of $\mathcal{L}$ into $\mathcal{M}$.

By the definition, if $\mathcal{M}$ is a general reduction of $\mathcal{L}$ with a reduction factor $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$, then the list $\tilde{\mathcal{S}} = \mathcal{S}_1 \diamond \mathcal{S}_2 \diamond \cdots \diamond \mathcal{S}_{d-c}$ is an expansion of $\mathcal{L}''$ with an expansion factor $\mathcal{S}$.

Note that if $\mathcal{M}$ is a simple reduction of $\mathcal{L}$, then each component in $\mathcal{M}$ is the product of one or more components of $\mathcal{L}$. On the other hand, if $\mathcal{M}$ is a general reduction of $\mathcal{L}$, then each component in $\mathcal{M}$ is either (i) a component in the multiplicant sublist $\mathcal{L}'$ or (ii) the product of a component in $\mathcal{L}'$ and a factor of one of the components in the multiplier sublist $\mathcal{L}''$.

Let G be a torus or a mesh of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and let H be a torus or a mesh of shape $\mathcal{M} = (m_1, m_2, \ldots, m_c)$. Assume that $\mathcal{M}$ is a general reduction of $\mathcal{L}$ with a reduction factor $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$. Let $\tilde{\mathcal{S}} = (s_1, s_2, \ldots, s_b) = \mathcal{S}_1 \diamond \mathcal{S}_2 \diamond \cdots \diamond \mathcal{S}_{d-c}$, where $d - c < b \le c$, and let $\mathcal{I} = (\underbrace{1, \ldots, 1}_{c-b})$. Let G' be a graph of shape $\mathcal{L}' \diamond \mathcal{L}''$ and of the same type as G, and let H' be a graph of shape $[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'$ and of the same type as H. We now construct an embedding of G into H in three steps: $G \to G' \to H' \to H$. Let $\alpha : [d]^+ \to [d]^+$ be a permutation such that $\alpha(\mathcal{L}) = \mathcal{L}' \diamond \mathcal{L}''$, and let $\beta : [c]^+ \to [c]^+$ be a permutation such that $\beta([\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}') = \mathcal{M}$. By the definition of general reduction, such permutations always exist. The graph G can be embedded into G' with unit dilation cost using the permutation α, and H' can be embedded into H with unit dilation cost using the permutation β. Next we construct an embedding of G' into H'.

The graph G' has shape $\mathcal{L}' \diamond \mathcal{L}'' = (l_{\alpha(1)}, l_{\alpha(2)}, \ldots, l_{\alpha(c)}) \diamond (l_{\alpha(c+1)}, l_{\alpha(c+2)}, \ldots l_{\alpha(d)})$. If G' is a mesh, we can think of G' as an $\mathcal{L}'$-mesh of *supernodes* with each supernode being an $\mathcal{L}''$-mesh, that is, the supernode $(i_1, i_2, \ldots, i_c)$ consists of all nodes $(i_1, i_2, \ldots, i_c) \diamond (*, *, \ldots, *)$ in G', where for all $j \in [c]^+$, $i_j \in [l_{\alpha(j)}]$, and $(*, *, \ldots, *)$ denotes all lists in $\Omega_{\mathcal{L}''}$. For example, if we view the $(3,3,6)$-mesh given in Figure 3.12(a) as a $(3,3)$-mesh of supernodes, then the supernode $(2,0)$ consists of the nodes $(2,0,0)$, $(2,0,1)$, $(2,0,2)$, $(2,0,3)$, $(2,0,4)$, and $(2,0,5)$. These nodes are labeled 0, 1, 2, 3, 4, and 5 in the figure. Similarly, if G' is a torus, we can think of G' as an $\mathcal{L}'$-torus of supernodes with each supernode being an $\mathcal{L}''$-torus.

The graph H' has shape $[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}' = (s_1 l_{\alpha(1)}, s_2 l_{\alpha(2)}, \ldots, s_b l_{\alpha(b)}, l_{\alpha(b+1)} \ldots, l_{\alpha(c)})$. If H' is a mesh, we can think of H' as an $\mathcal{L}'$-mesh of supernodes with each supernode being an $\tilde{\mathcal{S}}$-mesh, that is, the supernode $(i_1, i_2, \ldots, i_c)$ consists of all nodes $[(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + (*, *, \ldots, *)] \diamond (i_{b+1}, i_{b+2}, \ldots, i_c)$ in H', where for all $j \in [c]^+$, $i_j \in [l_{\alpha(j)}]$, and $(*, *, \ldots, *)$ denotes all lists in $\Omega_{\tilde{\mathcal{S}}}$. For example, if we view the $(6,9)$-mesh in Figure 3.12(b) as a $(3,3)$-mesh of supernodes, then the supernode $(2,0)$ consists of the nodes $(4,0)$, $(4,1)$, $(4,2)$, $(5,0)$, $(5,1)$, and $(5,2)$. These nodes are labeled 0, 1, 2, 5, 4, and 3 in the figure. If H' is a torus, we can also think of H' as an $\mathcal{L}'$-torus of supernodes. Each supernode in H' is now an $\tilde{\mathcal{S}}$-mesh instead of an $\tilde{\mathcal{S}}$-torus. Notice that we cannot divide a torus into toruses of the same dimension and of smaller sizes because the neighborship required at the boundary nodes of the smaller toruses cannot be satisfied.

In summary, the supernodes of G' are formed by partitioning the shape of G' into two parts, with one part forming the shape of the supernodes, and the other the shape of the graph consisting of these supernodes. On the other hand, the supernodes of H' are formed by factoring the length of each dimension of H' into one or two factors, with one factor forming the length of a dimension of the graph consisting of the supernodes, and the other factor, if present, forming the length of a dimension of the supernodes. The dimensions of the supernodes of G' and the graph consisting of these supernodes are both lower than the dimension of G'. On the other hand, the dimension of the supernodes of H' may be lower than the dimension of H', while the dimension of the graph consisting of these supernodes is always the same as the dimension of H'. With respect to supernodes, G' and H' have the same shape $\mathcal{L}'$. The shape of the supernodes of H' ($\tilde{\mathcal{S}}$) is an expansion of the shape of the supernodes of G' ($\mathcal{L}''$) with an expansion factor of $\mathcal{S}$.

We consider the following four cases for constructing an embedding of G' into H'.

Case 1. G' and H' are meshes.

In this case, G' and H' are $\mathcal{L}'$-meshes of supernodes. Neighboring supernodes in G' can be mapped to neighboring supernodes in H' using the identity function. The $\mathcal{L}''$-meshes (supernodes of G') can then be embedded into the $\tilde{\mathcal{S}}$-meshes (supernodes of H') using the embedding function $\mathcal{F}_{\mathcal{S}} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$ defined in the preceding subsection. Hence, we map each node $(i_1, i_2, \ldots, i_d)$ in G' to the node

$$\mathcal{F}'_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{F}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2}, \ldots, i_c)$$

in H'. We call $(s_1 i_1, s_2 i_2, \ldots, s_b i_b)$ and $(i_{b+1}, \ldots, i_c)$ the *base*, and $\mathcal{F}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))$ the *offset*.

Let $\mathcal{F}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d)) = (e_1, e_2, \ldots, e_b)$. We can write

$$\mathcal{F}'_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) \text{ as } (s_1 i_1 + e_1, s_2 i_2 + e_2, \ldots, s_b i_b + e_b, i_{b+1}, \ldots, i_c).$$

Since $\mathcal{F}_{\mathcal{S}}\colon \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$ is bijective, and for all $i \in [b]^+$, $0 \le e_i < s_i$, the function $\mathcal{F}'_{\mathcal{S}}\colon \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$ is injective. Since $|\Omega_{\mathcal{L}' \diamond \mathcal{L}''}| = |\Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}|$, $\mathcal{F}'_{\mathcal{S}}$ is bijective. Therefore, the function $\mathcal{F}'_{\mathcal{S}}$ is an embedding of G' into H'.

This embedding has a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$. Let $A = (i_1, i_2, \ldots, i_d)$ and $B = (i'_1, i'_2, \ldots, i'_d)$ be an arbitrary pair of neighboring nodes in G', and let $k = [d]^+$ be the index at which $i_k \neq i'_k$. Let $A' = \mathcal{F}'_{\mathcal{S}}(A)$ and $B' = \mathcal{F}'_{\mathcal{S}}(B)$. If $k \in [c]^+$, then A' and B' have the same offset but different bases. Since H' is a mesh, the distance between A' and B' is $|s_k i_k - s_k i'_k|$ if $k \in [b]^+$, and $|i_k - i'_k|$ if $k \in \{b+1, \ldots, c\}$. Since G' is also a mesh, we have $|i_k - i'_k| = 1$. Therefore, the distance between A' and B' in H' is s_k if $k \in [b]^+$, and 1 if $k \in \{b+1, \ldots, c\}$. If $k \in \{c+1, \ldots, d\}$, then A' and B' have the same base but different offsets. Since the function $\mathcal{F}_{\mathcal{S}}$ embeds an $\mathcal{L}''$-mesh into an $\tilde{\mathcal{S}}$-mesh with unit dilation cost, the distance between A' and B' in H' is 1.

Case 2. G' is a mesh and H' is a torus.

We use the embedding function $\mathcal{F}'_{\mathcal{S}}$ from Case 1 but modifying the analysis slightly. We change the distance measure between A' and B' from δ_m-distance to δ_t-distance, and use the relation that for all $k \in [b]^+$, $m_k = s_k l_k$ and $l_k > 1$. In this way, we can show that this embedding also gives a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$.

Case 3. G' and H' are both toruses.

Since G' and H' are both $\mathcal{L}'$-toruses of supernodes, neighboring supernodes in G' can be mapped to neighboring supernodes in H' using the identity function. The $\mathcal{L}''$-toruses (supernodes of G') can then be embedded into the $\tilde{\mathcal{S}}$-meshes (supernodes of H') using the function $\mathcal{G}_{\mathcal{S}} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$ defined in the preceding subsection. Hence, we map each node $(i_1, i_2, \ldots, i_d)$ in G' to the node

$$\mathcal{G}'_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{G}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2}, \ldots, i_c)$$

in H'. This mapping is also bijective, and is therefore an embedding of G' into H'.

This embedding also has a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$. Let A, B, and k be defined as in Case 1; and let $A' = \mathcal{G}'_{\mathcal{S}}(A)$, and $B' = \mathcal{G}'_{\mathcal{S}}(B)$. Since G' is a torus, $|i_k - i'_k|$ is either 1 or $l_k - 1$. If $k \in [b]^+$, then the distance between A' and B' is $\min\{|s_k i_k - s_k i'_k|,\ m_k - |s_k i_k - s_k i'_k|\} = \min\{s_k |i_k - i'_k|,\ s_k(l_k - |i_k - i'_k|)\}$; since $m_k = s_k l_k$, this distance is s_k. If $k \in \{b+1, \ldots, c\}$, the distance between A' and B' is $\min\{|i_k - i'_k|,\ m_k - |i_k - i'_k|\}$; since $m_k = l_k$, this distance is 1. If $k \in \{c+1, \ldots, d\}$, then the distance between A' and B' in H' is at most 2 because the function $\mathcal{G}_{\mathcal{S}}$ embeds an $\mathcal{L}''$-torus into an $\tilde{\mathcal{S}}$-mesh with a dilation cost of 2. Finally, since for all $i \in [d]^+$, $l_i > 1$, we have $\max\{s_1, s_2, \ldots, s_b\} \geq 2$. Therefore, the embedding has a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$.

Case 4. G' is a torus and H' is a mesh.

By Lemma 3.4.1, neighboring supernodes of G' can be mapped to some supernodes in H' at a distance no greater than 2 by embedding each supernode $(i_1, i_2, \ldots, i_c)$ in G' into the supernode $(t_{l_{\alpha(1)}}(i_1), t_{l_{\alpha(2)}}(i_2), \ldots, t_{l_{\alpha(c)}}(i_c))$ in H'. The $\mathcal{L}''$-toruses in G' are then embedded into the $\tilde{\mathcal{S}}$-meshes using the function $\mathcal{G}_{\mathcal{S}}$. Hence, we can map each node $(i_1, i_2, \ldots, i_c)$ in G' to the node

$$\begin{aligned}\mathcal{G}''_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) \quad = \quad & [(s_1 t_{l_{\alpha(1)}}(i_1), s_2 t_{l_{\alpha(2)}}(i_2), \ldots, s_b t_{l_{\alpha(b)}}(i_b)) + \mathcal{G}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \\ & \diamond (t_{l_{\alpha(b+1)}}(i_{b+1}), t_{l_{\alpha(b+2)}}(i_{b+2}) \ldots, t_{l_{\alpha(c)}}(i_c))\end{aligned}$$

in H'. This mapping is also bijective, and is therefore an embedding of G' into H'.

Let A, B, and k be defined as in Case 1, and let $A' = \mathcal{G}''_{\mathcal{S}}(A)$ and $B' = \mathcal{G}''_{\mathcal{S}}(B)$. The distance between A' and B' is $|s_k t_{l_{\alpha(k)}}(i_k) - s_k t_{l_{\alpha(k)}}(i'_k)|$ if $k \in [b]^+$, and $|t_{l_{\alpha(k)}}(i_k) - t_{l_{\alpha(k)}}(i'_k)|$ if $k \in \{b+1, \ldots, c\}$. Since for all $j \in [c]^+$, the cyclic sequence $t_{l_{\alpha(j)}}$ has a δ_m-spread of 2 if $l_{\alpha(j)} > 2$, and 1 otherwise, this distance is at most $2s_k$ if $k \in [b]^+$, and at most 2 if $k \in \{b+1, \ldots, c\}$. If $k \in \{c+1, \ldots, d\}$, then as in Case 3, the distance between A' and B' in H' is at most 2. Hence, the embedding has a dilation cost at most $2\max\{s_1, s_2, \ldots, s_b\}$.

In summary, the sequence of embeddings $G \to G' \to H' \to H$ defined above has a dilation cost at most $2\max\{s_1, s_2, \ldots, s_b\}$ if G is a torus, H is a mesh, and a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$ otherwise.

We refine and formalize the above results in the following definition and theorem.

Definition 3.4.7 Let d and c be positive integers such that $c < d < 2c$. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases. Assume that $\mathcal{M}$ is a general reduction of $\mathcal{L}$ with a reduction factor $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$, multiplicant sublist $\mathcal{L}'$, and multiplier sublist $\mathcal{L}''$. Let $\alpha : [d]^+ \to [d]^+$ be a permutation such that $\alpha(\mathcal{L}) = \mathcal{L}' \diamond \mathcal{L}''$. Let $\tilde{\mathcal{S}} = (s_1, s_2, \ldots, s_b) = \mathcal{S}_1 \diamond \mathcal{S}_2 \diamond \cdots \diamond \mathcal{S}_{d-c}$, and let $\mathcal{I} = (\underbrace{1, 1, \ldots, 1}_{c-b})$. Let $\mathcal{F}_{\mathcal{S}} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$, $\mathcal{H}_{\mathcal{S}} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$, and $\mathcal{G}_{\mathcal{S}} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$ be defined as in Definition 3.4.2. The functions $\mathcal{F}'_{\mathcal{S}} : \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$, $\mathcal{H}'_{\mathcal{S}} : \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$, $\mathcal{G}'_{\mathcal{S}} : \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$,

$\mathcal{H}''_{\mathcal{S}}: \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$, and $\mathcal{G}''_{\mathcal{S}}: \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$ are defined as follows: for all $(i_1, i_2, \ldots, i_d) \in \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$,

$$\mathcal{F}'_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{F}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2} \ldots, i_c),$$

$$\mathcal{H}'_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{H}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2} \ldots, i_c),$$

$$\mathcal{G}'_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{G}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2} \ldots, i_c),$$

$$\begin{aligned} \mathcal{H}''_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) &= [(s_1 t_{l_{\alpha(1)}}(i_1), s_2 t_{l_{\alpha(2)}}(i_2), \ldots, s_b t_{l_{\alpha(b)}}(i_b)) + \mathcal{H}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \\ &\quad \diamond (t_{l_{\alpha(b+1)}}(i_{b+1}), t_{l_{\alpha(b+2)}}(i_{b+2}) \ldots, t_{l_{\alpha(c)}}(i_c)). \end{aligned}$$

$$\begin{aligned} \mathcal{G}''_{\mathcal{S}}((i_1, i_2, \ldots, i_d)) &= [(s_1 t_{l_{\alpha(1)}}(i_1), s_2 t_{l_{\alpha(2)}}(i_2), \ldots, s_b t_{l_{\alpha(b)}}(i_b)) + \mathcal{G}_{\mathcal{S}}((i_{c+1}, i_{c+2}, \ldots, i_d))] \\ &\quad \diamond (t_{l_{\alpha(b+1)}}(i_{b+1}), t_{l_{\alpha(b+2)}}(i_{b+2}) \ldots, t_{l_{\alpha(c)}}(i_c)). \end{aligned}$$

Furthermore, let $\beta : [c]^+ \to [c]^+$ be a permutation such that $\beta([\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}') = \mathcal{M}$. Then we have the functions $\beta \circ \mathcal{F}'_{\mathcal{S}} \circ \alpha: \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$, $\beta \circ \mathcal{H}'_{\mathcal{S}} \circ \alpha: \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$, $\beta \circ \mathcal{G}'_{\mathcal{S}} \circ \alpha: \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$, $\beta \circ \mathcal{H}''_{\mathcal{S}} \circ \alpha: \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$, and $\beta \circ \mathcal{G}''_{\mathcal{S}} \circ \alpha: \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$. $\square$

Theorem 3.4.4 *Let d and c be positive integers such that $c < d < 2c$, and $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases. Let G be a torus or a mesh of shape $\mathcal{L}$, and let H be a torus or a mesh of shape $\mathcal{M}$. Assume that $\mathcal{M}$ is a general reduction of $\mathcal{L}$ with a reduction factor $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$, multiplicant sublist $\mathcal{L}'$, and multiplier sublist $\mathcal{L}''$. Let $\tilde{\mathcal{S}} = (s_1, s_2, \ldots, s_b) = \mathcal{S}_1 \diamond \mathcal{S}_2 \diamond \cdots \diamond \mathcal{S}_{d-c}$, and let $\mathcal{I} = (\underbrace{1, 1, \ldots, 1}_{c-b})$. Let $\alpha : [d]^+ \to [d]^+$ be a permutation such that $\alpha(\mathcal{L}) = \mathcal{L}' \diamond \mathcal{L}''$, and let $\beta : [c]^+ \to [c]^+$ be a permutation such that $\beta([\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}') = \mathcal{M}$.*

(1) *If G is a mesh, then G can be embedded into H with a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$; the function $\beta \circ \mathcal{F}'_{\mathcal{S}} \circ \alpha$ gives such an embedding.*

(2) *If both G and H are toruses, and for all $i \in [d-c]^+$, $\mathcal{S}_i$ consists of at least two components such that the first component is an even number, then G can be embedded into H with a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$; the function $\beta \circ \mathcal{H}'_{\mathcal{S}} \circ \alpha$ gives such an embedding.*

(3) *If both G and H are toruses, and the condition on the lists in $\mathcal{S}$ stated in (2) is not satisfied, then G can be embedded into H with a dilation cost of $\max\{s_1, s_2, \ldots, s_b\}$; the function $\beta \circ \mathcal{G}'_{\mathcal{S}} \circ \alpha$ gives such an embedding.*

(4) *If G is a torus, H is a mesh, and for all $i \in [d-c]^+$, $\mathcal{S}_i$ consists of at least two components such that the first component is an even number, then G can be embedded into H with a dilation cost of $2\max\{s_1, s_2, \ldots, s_b\}$; the function $\beta \circ \mathcal{H}''_{\mathcal{S}} \circ \alpha$ gives such an embedding.*

(5) *If G is a torus, H is a mesh, and the condition on the lists in $\mathcal{S}$ stated in (4) is not satisfied, then G can be embedded into H with a dilation cost of $2\max\{s_1, s_2, \ldots, s_b\}$, the function $\beta \circ \mathcal{G}''_{\mathcal{S}} \circ \alpha$ gives such an embedding.*

□

Theorem 3.4.4 refines the embeddings in our informal discussion for lowering dimension. In case (2), the function $\mathcal{H}'_{\mathcal{S}}$ has the same dilation cost as the function $\mathcal{G}'_{\mathcal{S}}$, but the neighborship in the dimensions of G corresponding to $\mathcal{L}''$ is maintained by $\mathcal{H}'_{\mathcal{S}}$, not by $\mathcal{G}'_{\mathcal{S}}$. Similarly, in case (4), the function $\mathcal{H}''_{\mathcal{S}}$ has the same dilation cost as the function $\mathcal{G}''_{\mathcal{S}}$, but the neighborship in the dimensions of G corresponding to $\mathcal{L}''$ is maintained by $\mathcal{H}'_{\mathcal{S}}$, not by $\mathcal{G}'_{\mathcal{S}}$. These extra properties of $\mathcal{H}'_{\mathcal{S}}$ and $\mathcal{H}''_{\mathcal{S}}$ can improve data routing complexities in Chapter 4.

The condition of general reduction requires that the dimension of H be at least half of the dimension of G. If this condition is not satisfied, an embedding of G into H can still be constructed using the results in this subsection provided that there exists a sequence of intermediate graphs in which every pair of successive graphs have shapes satisfying the condition of general reduction.

As will be shown in Section 3.5, if G and H are both square, then one of the following two conditions must be true: (i) their shapes satisfy the condition of simple reduction, and (ii) the sequence of graphs described above exists.

3.5 Generalized embeddings among square toruses and square meshes

The results for generalized embeddings developed in the last section can be applied only if the shapes of G and H satisfy either the condition of expansion (for increasing dimension) or the condition of reduction (for lowering dimension). In this section, we study the cases in which G and H are both square. For these cases, we can always construct an embedding of G into H through a sequence of one or more embeddings using the embedding functions defined in Section 3.4.

Let d be the dimension of G, c be the dimension of H, a be the greatest common denominator of d and c, and ℓ be the length of the dimensions of G. The major results of this section are the following:

For the case of lowering dimension ($c < d$), G can be embedded into H with a dilation cost of $2\ell^{(d-c)/c}$ if G is a torus and H is a mesh, and with a dilation cost of $\ell^{(d-c)/c}$ otherwise. For fixed values of d and c, these dilation costs are optimal to within a constant.

For the case of increasing dimension ($d < c$), if c is divisible by d, then G can be embedded into H with an optimal dilation cost of 2 if G is a torus of odd size and H is a mesh, and with unit dilation cost otherwise. If c is not divisible by d, then G can be embedded into H with a dilation cost of $2\ell^{(d-a)/c}$ if G is a torus of odd size and H is a mesh, and with a dilation cost of $\ell^{(d-a)/c}$ otherwise.

A lower bound on dilation cost for lowering dimension

In [Ros75], Rosenberg studied the problem of embedding finite arrays (meshes), prism arrays, and orthant arrays into lines to minimize proximity in various local and global senses. Let t be an embedding of a d-dimensional mesh G into a line. For any positive integer k, *the diameter of preservation* σ_k is the smallest positive integer i such that for every node v in G, and for every pair of nodes u and w in G whose distances from v are no greater than k, $\delta_m(t(u), t(w)) < i$. Rosenberg proved that $\sigma_k > bk\mu^{d-1}$, where μ is the length of the shortest dimension of G, and b depends only on d and is a constant with respect to μ.

Let G be a d-dimensional torus or a d-dimensional mesh, and H be a c-dimensional torus or a c-dimensional mesh such that $c < d$ and G and H are of the same size. In the following, using a straightforward modification of Rosenberg's proof for the lower bound on the diameter of preservation [Ros75], we show that the dilation cost of any embedding of G into H is bounded from below by $b\mu^{(d-c)/c}$, where μ is the length of the shortest dimension of G, and b is a constant with respect to μ and depends only on d and c. This lower bound on dilation cost will be used to prove the optimality properties of our embeddings among square toruses and square meshes in the lowering dimension case.

Given an $(l_1, l_2, \ldots, l_d)$-mesh G, a node v in G, and a positive integer k, let $Q(v,k)$ denote the set of nodes in G whose distances from v are no greater than k.

Lemma 3.5.1 *[Ros75] Let G be a d-dimensional mesh. Let μ be the length of the shortest dimension of G. For any positive integer k such that $k < \mu$, $\max_{v \in G}|Q(v,k)| \geq \binom{k+d}{d} > bk^d$, where $b > 0$ is a constant with respect to k, and depends only on d.* $\Box$

Lemma 3.5.2 *Let G be a d-dimensional mesh, and H be a c-dimensional mesh such that $c < d$ and G and H are of the same size. Let t be an embedding of G into H with a dilation cost of ρ. Then for any node v in G and any positive integer k, $|Q(v,k)| \leq (2k\rho + 1)^c$.* $\Box$

Proof. Let p_1, p_2, $\ldots$, p_c be nonnegative integers. A set of lists is said to *lie within* a c-dimensional interval $[p_1, p_2, \ldots, p_c]$ if the lists are all of the form $(i_1+e_1, i_2+e_2, \ldots, i_c+e_c)$, and for all $j \in [c]^+$, i_j is some fixed integer and $e_j \in [p_j]$. For v an arbitrary node in G and k an arbitrary positive integer, let $t(Q(v,k))$ be the set of images of all the nodes in $Q(v,k)$ under the embedding t. We first show by induction on k that $t(Q(v,k))$ lies within a c-dimensional interval $[2k\rho+1, 2k\rho+1, \ldots, 2k\rho+1]$.

Induction basis: $k = 1$.

Let $q = |Q(v,1)|$. Let $(a_1^1, a_2^1, \ldots, a_c^1)$, $(a_1^2, a_2^2, \ldots, a_c^2)$, $\ldots$, $(a_1^q, a_2^q, \ldots, a_c^q)$ denote the nodes in $t(Q(v,1))$. For all $j \in [c]^+$, let $\alpha_j = \min\{a_j^1, a_j^2, \ldots, a_j^q\}$, and let $\beta_j = \max\{a_j^1, a_j^2, \ldots, a_j^q\}$. Since for all $u, w \in Q(v,1)$, $\delta_m(t(u), t(w)) \leq \delta_m(t(u), t(v)) + \delta_m(t(v), t(w)) \leq 2\rho$, we have for all $j \in [c]^+$, $|\alpha_j - \beta_j| \leq 2\rho$. Therefore, $t(Q(v,1))$ must lie within a c-dimensional interval $[2\rho+1, 2\rho+1, \ldots, 2\rho+1]$.

Induction hypothesis: Assume that for all $k \leq k'$, $t(Q(v,k'))$ lies within a c-dimensional interval $[2k'\rho+1, 2k'\rho+1, \ldots, 2k'\rho+1]$.

Induction step: $k = k'+1$.

Since every node u in $Q(v,k'+1)$ must either belong to $Q(v,k')$ or be a neighbor of some node w in $Q(v,k')$, the smallest c-dimensional interval containing $t(Q(v,k'+1))$ contains at most 2ρ elements more in each of the c dimensions than the corresponding interval for $t(Q(v,k'))$. Therefore, by our induction hypothesis, $t(Q(v,k'+1))$ must lie within a c-dimensional interval $[2\rho+2k'\rho+1, 2\rho+2k'\rho+1, \ldots, 2\rho+2k'\rho+1] = [2(k'+1)\rho+1, 2(k'+1)\rho+1, \ldots, 2(k'+1)\rho+1]$.

For any positive integer k, the maximum number of lists that can lie within a c-dimensional interval $[2k\rho+1, 2k\rho+1, \ldots, 2k\rho+1]$ is $(2k\rho+1)^c$. Since t is bijective, we have $|Q(v,k)| \leq (2k\rho+1)^c$. $\Box$

Lemma 3.5.3 *Let G and H be meshes of the same size. Let G' be a torus of the same shape as G, and H' be a torus of the same shape as H. Assume that the dilation cost of any embedding of G into H is bounded from below by x. Then the dilation cost of any* *embedding of G' into H, G into H', or G' into H' is bounded from below by bx, for some constant b.* □

Proof. Let ζ be the dilation cost of an arbitrary embedding of the torus G' into the torus H'. By Lemma 3.4.1, the mesh G can be embedded into the torus G' with unit dilation cost, and the torus H' can be embedded into the mesh H with a dilation cost of 2. Since the sequence of embeddings $G \xrightarrow{1} G' \xrightarrow{\zeta} H' \xrightarrow{2} H$ provides an embedding of G into H with a dilation cost of 2ζ, we have $\zeta \geq x/2$.

Similarly, let λ be the dilation cost of an arbitrary embedding of G' into H, and γ the dilation cost of an arbitrary embedding of G into H'. Since the sequence $G \xrightarrow{1} G' \xrightarrow{\lambda} H$ and the sequence $G \xrightarrow{\gamma} H' \xrightarrow{2} H$ also provide embeddings of G into H with dilation costs of λ and 2γ respectively, we have $\lambda \geq x$ and $\gamma \geq x/2$. □

Theorem 3.5.1 *Let G be a d-dimensional torus or a d-dimensional mesh, and let H be a c-dimensional torus or a c-dimensional mesh such that $c < d$ and G and H are of the same size. Let μ be the length of the shortest dimension of G. Then the dilation cost of any embedding of G into H is bounded from below by $b\mu^{(d-c)/c}$, for some positive number b that is a constant with respect to μ and depends only on d and c.* □

Proof. We first assume that G and H are both meshes. Let ρ be the dilation cost of an arbitrary embedding of G into H. By Lemmas 3.5.1 and 3.5.2, for any positive integer k such that $k < \mu$, $(2k\rho + 1)^c > bk^d$, for some positive number b that depends only on d. We thus have $\rho > (\frac{b^{1/c}}{2})k^{(d-c)/c} - \frac{1}{2k} \geq (\frac{b^{1/c}}{2})k^{(d-c)/c}$. By letting $k = \mu - 1$, we have $\rho \geq (\frac{b^{1/c}}{2})(\mu - 1)^{(d-c)/c}$. Since $\mu \geq 2$, $\mu - 1 \geq \frac{\mu}{2}$. Therefore $\rho \geq b'\mu^{(d-c)/c}$, for some b' that is a constant with respect to μ and depends only on d and c. The other cases follow from Lemma 3.5.3. □

Embeddings for lowering dimension

Theorem 3.5.2 *Let G be a square torus or a square mesh of dimension d, and H be a square torus or a square mesh of dimension c such that $c < d$ and G and H are of the same size. Let ℓ be the length of the dimensions of G. Assume that d is divisible by c. Then the shapes of G and H always satisfy the condition of simple reduction. Furthermore, G can be embedded into H with a dilation cost of $2\ell^{(d-c)/c}$ if G is a torus and H is a mesh, and with a dilation cost of $\ell^{(d-c)/c}$ otherwise; for fixed vaules of d and c, such dilation costs are optimal to within a constant.* □

Proof. Let $b = d/c$. Since d is divisible by c, b is an integer. Let m be the length of the dimensions of H. Since G and H are of the same size, we have $m^c = \ell^d$, and $m = \ell^b$. Hence, H is a simple reduction of G with a reduction factor of $((\underbrace{\ell, \ldots, \ell}_{b}), \ldots, (\underbrace{\ell, \ldots, \ell}_{b}))$. Therefore, by Theorem 3.4.3, G can be embedded into H with a dilation cost of $2m/\ell = 2\ell^{(d-c)/c}$ if G is a torus and H is a mesh, and with a dilation cost of $\ell^{(d-c)/c}$ otherwise.

By Theorem 3.5.1, the optimal dilation cost of embedding G into H is bounded from below by $b\ell^{(d-c)/c}$, for some positive number $b > 0$ that is a constant with respect to ℓ and depends only on d and c. Since the dilation costs of our embeddings are either $2\ell^{(d-c)/c}$ or $\ell^{(d-c)/c}$, they are optimal to within a constant for fixed values of d and c. □

The following lemma states a property of integers that will be used in Theorem 3.5.3 to construct our embedding for lowering dimension case in which d is not divisible by c. This lemma in turn uses the following properties of integers [Bun72]:

(∗) Any positive integer $N > 1$ can be written uniquely in a *standard form* $N = p_1^{b_1} p_2^{b_2} \cdots p_r^{b_r}$ such that for all $i \in \{1, 2, \ldots, r\}$, b_i is a positive integer and each p_i is a prime with $1 < p_1 < \cdots < p_r$.

Lemma 3.5.4 *Let x be any integer greater than 1, and let u and v be any integers that are relatively prime. Assume that $x^{u/v}$ is an integer. Then $x^{1/v}$ is also an integer.* □

Proof. Let $y = x^{u/v}$. By assumption, y is an integer. Furthermore, since x is an integer greater than 1, y must also be an integer greater than 1. By property (∗) of integers, x can be rewritten in its unique standard form $p_1^{b_1} p_2^{b_2} \cdots p_r^{b_r}$ in which $r, b_1, b_2, \ldots, b_r$ are positive integers and p_1, p_2, …, p_r are distinct primes with $p_1 < p_2 < \cdots < p_r$. Similarly, y can be written in its unique standard form $q_1^{c_1} q_2^{c_2} \cdots q_s^{c_s}$ in which $s, c_1, c_2, \ldots, c_s$ are positive integers and q_1, q_2, …, q_s are distinct primes with $q_1 < q_2 < \cdots < q_s$.

Since $y^v = x^u$, we have $q_1^{vc_1} q_2^{vc_2} \cdots q_s^{vc_s} = p_1^{ub_1} p_2^{ub_2} p_r^{ub_r}$. Since $q_1, q_2, \ldots, q_s$ are distinct primes with $q_1 < q_2 < \ldots < q_s$ and $p_1, p_2, \ldots, p_r$ are also distinct primes with $p_1 < p_2 < \ldots < p_r$, we have $r = s$ and for all $i \in [r]^+$, $q_i = p_i$ and $vc_i = ub_i$. Hence, for all $i \in [r]^+$, we have $ub_i/v = c_i$. Since c_i is an integer, and u and v are relatively prime, b_i must be divisible by v. It follows that $p_1^{b_1/v} p_2^{b_2/v} \cdots p_r^{b_r/v}$, which is $x^{1/v}$, must be an integer. □

Theorem 3.5.3 *Let G be a square torus or a square mesh of dimension d, and H a square torus or a square mesh of dimension c such that $c < d$ and G and H are of the same size. Let ℓ be the length of the dimensions of G. Assume that d is not divisible by c. Then there always exists a sequence of intermediate graphs in which the shapes of every pair of successive graphs satisfy the condition of general reduction. Furthermore, G can be embedded into H with a dilation cost of $2\ell^{(d-c)/c}$ if G is a torus and H is a mesh, and with a dilation cost of $\ell^{(d-c)/c}$ otherwise. For fixed vaules of d and c, these dilation costs are optimal to within a constant.* □

Proof. We first treat the case in which G is a mesh, and H is a mesh or a torus. Let m be the length of the dimensions of H. Since G and H are of the same size, we have $m^c = \ell^d$, and $m = \ell^{d/c}$. Since m is an integer, $\ell^{d/c}$ must also be an integer.

We first consider the simple case in which d and c are relatively prime. By the definition of meshes, $\ell > 1$, and hence by Lemma 3.5.4, $\ell^{1/c}$ is an integer. Let I_0, I_1, …, I_{d-c-1} be all meshes, and I_{d-c} be of the same type as H. For all $k \in [d-c+1]$, I_k has dimension $d-k$ and shape

$$(\underbrace{\ell^{(c+k)/c}, \ldots, \ell^{(c+k)/c}}_{c}, \underbrace{\ell \ldots, \ell}_{d-c-k}).$$

We have $I_0 = G$; $I_{d-c} = H$; I_0, I_1, …, I_{d-c} all have the same size ℓ^d; and, except for I_0 and I_{d-c}, none of the meshes I_1, I_2, …, I_{d-c-1} is square. For all $k \in [d-c]$, the dimension of

I_k is greater than the dimension of I_{k+1} by 1, and the shape of I_{k+1} is a general reduction of the shape of I_k with a reduction factor

$$\mathcal{S}_{k+1} = ((\underbrace{\ell^{1/c}, \ldots, \ell^{1/c}}_{c})).$$

By Theorem 3.4.4, I_k can be embedded into I_{k+1} using the function $\mathcal{F}'_{\mathcal{S}_{k+1}}$ with a dilation cost of $\ell^{1/c}$. The sequence of embeddings $G = I_0 \to I_1 \to \cdots \to I_{d-c-1} \to I_{d-c} = H$ has a total of $d - c$ steps, with a dilation cost of $\ell^{1/c}$ in each step. This embedding of G into H therefore has a dilation cost of $\ell^{(d-c)/c}$.

Next we consider the case in which d and c are not relatively prime. Let a be the greatest common denominator of d and c, and let $u = d/a$ and $v = c/a$. Since d is not divisible by c, u and v are integers and relatively prime. We can write $\ell^{d/c}$ as $\ell^{u/v}$. Since $\ell^{u/v}$ is an integer and u and v are relatively prime, by Lemma 3.5.4, $\ell^{1/v}$ is an integer.

As in the preceding case, we can define a sequence of embeddings from G to H. This sequence consists of $u - v$ embedding steps, in each step of which the dimensions of the two corresponding graphs differ by a. Let I_0, I_1, …, I_{u-v-1} be all meshes, and I_{d-c} be of the same type as H. For all $k \in [u - v + 1]$, I_k has dimension $a(u - k)$ and shape

$$\mathcal{L}_k = (\underbrace{\ell^{(v+k)/v}, \ldots, \ell^{(v+k)/v}}_{av}, \underbrace{\ell, \ldots, \ell}_{a(u-v-k)}).$$

We have $I_0 = G$; $I_{u-v} = H$; I_0, I_1, …, I_{u-v} all have the same size $\ell^{au} = \ell^d$; and, except for I_0 and I_{u-v}, none of the meshes I_1, I_2, …, I_{u-v-1} is square.

For all $k \in [u - v]$, let $\mathcal{L}'_k$ be a list of length $a(u - k - 1)$, and $\mathcal{L}''_k$ be a list of length a such that

$$\mathcal{L}'_k = (\underbrace{\ell^{(v+k)/v}, \ldots, \ell^{(v+k)/v}}_{av}, \underbrace{\ell, \ldots, \ell}_{a(u-v-k-1)}) \quad \text{and} \quad \mathcal{L}''_k = (\underbrace{\ell, \ldots, \ell}_{a}).$$

$\mathcal{L}'_k \diamond \mathcal{L}''_k$ is a permutation of $\mathcal{L}_k$. Let

$$\mathcal{R}_k = (\underbrace{\ell^{1/v}, \ldots, \ell^{1/v}}_{v}) \quad \text{and} \quad \mathcal{R}'_k = \underbrace{\mathcal{R}_k \diamond \cdots \diamond \mathcal{R}_k}_{a}.$$

The list $\mathcal{R}'_k$ has length av. We have

$$\mathcal{L}''_k = (\underbrace{\prod \mathcal{R}_k, \ldots, \prod \mathcal{R}_k}_{a}).$$

The list $[\mathcal{R}'_k \diamond (\underbrace{1, \ldots, 1}_{a(u-v-k-1)})] \times \mathcal{L}'_k$ is $\mathcal{L}_{k+1}$. Therefore, the list $\mathcal{L}_{k+1}$ is a general reduction of the list $\mathcal{L}_k$ with a reduction factor of

$$\mathcal{S}_{k+1} = (\underbrace{\mathcal{R}_k, \ldots, \mathcal{R}_k}_{a}).$$

By Theorem 3.4.4, I_k can be embedded into I_{k+1} using the function $\mathcal{F}'_{\mathcal{S}_{k+1}}$ with a dilation cost of $\ell^{1/v}$.

In the sequence of embeddings $G = I_0 \to I_1 \to \cdots \to I_{u-v-1} \to I_{u-v} = H$, each embedding step has a dilation cost of $\ell^{1/v}$. Since there are a total of $u - v$ steps, this embedding of G into H has a dilation cost of $\ell^{(u-v)/v} = \ell^{(d-c)/c}$.

For the case in which G and H are both toruses, we modify the embedding procedure for the case in which G is a mesh as follows. For all $i \in [u - v + 1]$, let I_i be a torus. By Theorem 3.4.4, for all $k \in [u - v]$, I_k can be embedded into I_{k+1} using the function $\mathcal{H}'_{\mathcal{S}_{k+1}}$ with a dilation cost of $\ell^{1/v}$ if G is of even size, and using the function $\mathcal{G}'_{\mathcal{S}_{k+1}}$ with a dilation cost of $\ell^{1/v}$ otherwise. Therefore, the sequence of embeddings $G = I_0 \to I_1 \to \cdots \to I_{u-v} = H$ has a total dilation cost of $\ell^{(d-c)/c}$.

For the case in which G is a torus of odd size, and H is a mesh, we modify the embedding procedure for the case in which G is a mesh as follows. For all $i \in [u - v + 1]$, let I_i be a mesh. Let $\mathcal{L}$ be the shape of G. The torus G can be first embedded into the mesh I_0 using the function $\mathcal{T}_{\mathcal{L}}$ with a dilation cost of 2. By Theorem 3.4.4, for all $k \in [u - v]$, I_k can be embedded into I_{k+1} using the function $\mathcal{F}'_{\mathcal{S}_{k+1}}$ with a dilation cost of $\ell^{1/v}$. Therefore, the sequence of embeddings $G \to I_0 \to I_1 \to \cdots \to I_{u-v} = H$ has a total dilation cost of $2\ell^{(d-c)/c}$.

For the case in which G is a torus of even size, and H is a mesh, we modify the embedding procedure for the case in which G is a mesh as follows. For all $i \in [u - v + 1]$, let I_i be a torus. By Theorem 3.4.4, for all $k \in [u - v]$, I_k can be embedded into I_{k+1} using the function $\mathcal{H}'_{\mathcal{S}_{k+1}}$ with a dilation cost of $\ell^{1/v}$. Let $\mathcal{M}$ be the shape of H. The torus I_{u-v} can be embedded into the mesh H using the function $\mathcal{T}_{\mathcal{M}}$ with a dilation cost of 2. Therefore, the sequence of embeddings $G = I_0 \to I_1 \to \cdots \to I_{u-v} \to H$ has a total dilation cost of $2\ell^{(d-c)/c}$.

The optimality condition of these dilation costs follows from Theorem 3.5.1. $\square$

The next corollary follows directly from Theorem 3.5.3 by treating a hypercube as a mesh. This corollary also follows as a special case of Theorem 3.4.2.

Corollary 3.5.1 *A hypercube can be embedded into a square torus or a square mesh of the same size with a dilation cost of $m/2$, where m is the length of the dimensions of the given torus or mesh.* $\square$

Notice that in Theorem 3.5.3 and Corollary 3.5.1, the ratio of our dilation cost to the optimal dilation cost is bounded from above by $1/b$, for some positive number b that depends only on d and c. For fixed values of d and c, this upper bound on the ratio is a constant. Since in Theorem 3.5.3, an instance of G and H depends on d, c, and ℓ (or equivalently, on b, c, and m, since $\ell^d = m^c$), we can fix the values of d and c without fixing an instance of G and H. Therefore, in Theorem 3.5.3, for all problem instances in which d and c are fixed but ℓ is any integer greater than 1, the ratio of our dilation cost to the optimal dilation cost is bounded from above by a constant. On the other hand, in Corollary 3.5.1, an instance of G and H depends only on d and c. Fixing d and c fixes such an instance. Therefore, in this case, the upper bound $1/b$ on the ratio of our dilation cost to the optimal dilation cost varies with each problem instance.

A few special cases of embeddings among toruses and meshes of the same size for lowering dimension have been solved optimally in the literature: optimal embedding of an (ℓ, ℓ, ℓ)-mesh into a line of the same size with a dilation cost of $\lfloor 3\ell^2/4 + \ell/2 \rfloor$ [Fit74], optimal embedding of an (ℓ, ℓ)-mesh into a line of the same size with a dilation cost of ℓ

[Fit74], optimal embedding of an (ℓ, ℓ)-torus into a ring of the same size with a dilation cost of ℓ [MN86], and optimal embedding of a hypercube of size 2^d into a line of the same size with a dilation cost of $\sum_{k=0}^{d-1} \binom{k}{\lfloor k/2 \rfloor}$ [Har66].

For the cases of embedding an (ℓ, ℓ)-mesh into a line and embedding an (ℓ, ℓ)-torus into a ring, our embeddings also give a dilation cost of ℓ. Thus, both are truly optimal. For the case of embedding an (ℓ, ℓ, ℓ)-mesh into a line, our embedding gives a dilation cost of ℓ^2. Thus, it is optimal to within a constant $4/3$.

For the case of embedding a hypercube of size 2^d into a line, our embedding gives a dilation cost of 2^{d-1}. The optimal dilation cost $\sum_{k=0}^{d-1} \binom{k}{\lfloor k/2 \rfloor}$ can be written as $\varepsilon_{d-1} 2^{d-1}$, where $\varepsilon_0 = \varepsilon_1 = \varepsilon_2 = 1$, and for all $d \geq 3$, $\varepsilon_{d-1} > \varepsilon_d$ (see Appendix). Hence, our embedding is truly optimal for $1 \leq d \leq 3$. However, for all $d \geq 3$, the ratio of our dilation cost to the optimal dilation cost, which is $1/\varepsilon_{d-1}$, is strictly greater than 1. Furthermore, for all $d \geq 3$, this ratio is an increasing function of d, and hence cannot be bounded from above by a constant.

Embeddings for increasing dimension

Theorem 3.5.4 *Let G be a square torus or a square mesh of dimension d, and let H be a square torus or a square mesh of dimension c such that $d < c$ and G and H are of the same size. Assume that c is divisible by d. Then G can be embedded into H with an optimal dilation cost of 2 if G is a torus of odd size and H is a mesh, and with unit dilation cost otherwise.* □

Proof. Let $a = c/d$. By the assumption of the theorem, a is an integer. Let ℓ be the length of the dimensions of G, and m be the length of the dimensions of H. Let $\mathcal{L}$ be the shape of G, and $\mathcal{M}$ be the shape of H. We have

$$\mathcal{L} = (\underbrace{\ell, \ldots, \ell}_{d}) \quad \text{and} \quad \mathcal{M} = (\underbrace{m, \ldots, m}_{c})$$

Since G and H are of the same size, we have $\ell^d = m^c$, and $\ell = m^a$. Let

$$\mathcal{R} = (\underbrace{m, \ldots, m}_{a}).$$

Since $\prod \mathcal{R} = \ell$, and

$$\mathcal{M} = \underbrace{\mathcal{R} \diamond \cdots \diamond \mathcal{R}}_{d},$$

the list $\mathcal{M}$ is an expansion of the list $\mathcal{L}$, with an expansion factor of

$$(\underbrace{\mathcal{R}, \mathcal{R}, \ldots, \mathcal{R}}_{d}).$$

Assume that G is a torus of even size and H is a mesh of the same size. Since $d < c$, we have $a \geq 2$. Hence, the list $\mathcal{R}$ consists of at least two components. Furthermore, since the size of H is even, m must also be even, and hence, all of the components of $\mathcal{R}$ are even. Therefore, by Theorem 3.4.1, G can be embedded into H with unit dilation cost. The other cases of G and H also follow from Theorem 3.4.1. □

Theorem 3.5.5 *Let G be a square torus or a square mesh of dimension d, and let H be a square torus or a square mesh of dimension c such that $d < c$ and G and H are of the same size. Let ℓ be the length of the dimensions of G, and a be the greatest common divisor of c and d. Assume that c is not divisible by d. Then G can be embedded into H with a dilation cost of $2\ell^{(d-a)/c}$ if G is a torus of odd size and H is a mesh, and with a dilation cost of $\ell^{(d-a)/c}$ otherwise.* □

Proof. We construct an embedding of G into H through an intermediate graph G' for which the shape of G' is an expansion of that of G and the shape of H is a general reduction of that of G'. We first consider the case in which G is either a mesh or a torus of even size. Let m be the length of the dimensions of H. Let $u = d/a$, and $v = c/a$. Since u and v are relatively prime, and $\ell^{u/v}$ is an integer, by Lemma 3.5.4, $\ell^{1/v}$ is also an integer. Let G' be a mesh of dimension vd and with the length of the dimensions equal to $\ell^{1/v}$. The mesh G' has the same size as G, and the shape of G' is an expansion of the shape of G with an expansion factor of

$$\mathcal{V} = (\underbrace{\mathcal{R}, \ldots, \mathcal{R}}_{d}) \quad \text{where} \quad \mathcal{R} = (\underbrace{\ell^{1/v}, \ldots, \ell^{1/v}}_{v}).$$

By Theorem 3.4.1, the mesh G can be embedded into G' using the function $\mathcal{F}_{\mathcal{V}}$ with unit dilation cost if G is a mesh, and using the function $\mathcal{H}_{\mathcal{V}}$ with unit dilation cost otherwise.

Next we construct an embedding of G' into H. The dimension of G', which is vd, can be written as $(c/a)d = cu$. By definitions of u and v, we have $d = au$ and $c = av$. Since a is the greatest common divisor of d and c, and since by the assumption of the theorem, c is not divisible by d, we have $u > 1$. The dimension of G' is thus greater than the dimension of H. Since G' and H are both square and of the same size, and G' is a mesh, by Theorem 3.5.3, G' can be embedded into H with a dilation cost of $(\ell^{1/v})^{(vd-c)/c} = \ell^{(d-a)/c}$. Therefore, the embedding sequence $G \to G' \to H$ gives an embedding of G into H with a dilation cost of $\ell^{(d-a)/c}$.

For the case in which G is a torus of odd size, and H is a torus or a mesh, we modify the embedding procedure for the preceding case as follows. Let G' be a torus of dimension vd and with the length of the dimensions equal to $\ell^{1/v}$. By Theorem 3.4.1, the torus G can be embedded into the torus G' using the function $\mathcal{H}_{\mathcal{V}}$ with unit dilation cost. Since G' and H are both square and of the same size, $vd > c$, and G' is a torus, by Theorem 3.5.3, G' can be embedded into H with a dilation cost of $(\ell^{1/v})^{(vd-c)/c} = \ell^{(d-a)/c}$ if H is a torus, and G' can be embedded into H with a dilation cost of $2(\ell^{1/v})^{(vd-c)/c} = 2\ell^{(d-a)/c}$ otherwise. Therefore, the embedding sequence $G \to G' \to H$ gives an embedding of G into H with a dilation cost of $\ell^{(d-a)/c}$ if H is a torus, and with a dilation cost of $2\ell^{(d-a)/c}$ otherwise. □

In summary, our embeddings for square toruses and square meshes are all defined using the generalized embeddings defined in Section 3.4. For lowering dimension, if the dimension of G is divisible by the dimension of H, then the shape of H is a simple reduction of the shape of G. Otherwise, G can be embedded into H through a sequence of intermediate graphs in which every pair of successive graphs have shapes satisfying the condition of general reduction. In either case, our embeddings have dilation costs optimal to within a constant for fixed values of d and c. For increasing dimension, if the dimension of H is divisible by the dimension of G, then H is always an expansion of G, and an embedding of G into H can be immediately constructed by applying the results from

Section 3.4. Furthermore, this embedding is always optimal. If the dimension of H is not divisible by the dimension of G, then an embedding of G into H is constructed through an intermediate graph G' such that the shape of G' is an expansion of the shape of G and the shape of H is a general reduction of the shape of G'. This embedding, however, may not be optimal in general

3.6 Conclusion

In this chapter we studied embeddings among toruses and meshes of the same size. All the results are based on several basic embeddings of either a line or a ring into a torus or a mesh. The results for basic embeddings are all optimal. Among generalized embeddings for which at least one of the two graphs is not square, our results are restricted only to those special cases in which the shapes of the two graphs satisfy the condition of expansion for increasing dimension and the condition of reduction for lowering dimension. The results for lowering dimension are not optimal in general. On the other hand, the results for increasing dimension are all optimal except for the case when G is a torus of even size and H is a mesh; for this case, we provide an embedding with a dilation cost of 2 and under certain condition, an embedding with optimal unit dilation cost.

For increasing dimension, if the graph H is a hypercube, the condition of expansion can always be satisfied; similarly, for lowering dimension, if the graph G is a hypercube, the condition of simple reduction can always be satisfied. Consequently, our results for generalized embeddings can always be applied if one of the two graphs is a hypercube.

Furthermore, our results can always be applied if both graphs are square. For increasing dimension, these embeddings are optimal when the dimension of H is divisible by that of G. For lowering dimension, the embeddings are all optimal to within a constant for fixed values of d and c; by comparing with the several known optimal results in the literature, we have further shown that some of these embeddings are truly optimal.

A few special cases of the embedding problem studied in this thesis have been solved optimally in the literature: embedding a mesh (of size some power of 2) into a hypercube [CS86], embedding a 2-dimensional square torus into a ring [MN86], embedding a 2-dimensional square mesh into a line [Fit74], embedding a 3-dimensional square mesh into a line [Fit74], and embedding a hypercube into a line [Har66]. For these cases, our dilation cost is either optimal or optimal to within a constant. In addition to having minimum dilation cost, the embeddings of meshes into hypercubes given in [CS86] also satisfy other proximity properties, and they are derived based on binary reflected Gray codes. Our basic embeddings and generalized embeddings for increasing dimension are derived using a generalization of the technique used in [CS86].

Other closely related results in the literature include the following: embeddings of 2-dimensional square meshes into lines to minimize *average proximity* [DEL78b], embeddings of finite arrays (meshes), prism arrays, and orthant arrays into lines to minimize proximity in various local and global senses [Ros75], embeddings of 2-dimensional rectangular meshes into 2-dimensional square meshes to minimize the dilation costs while satisfying the constraints on expansion costs [AR82, Ell88], embeddings of meshes into hypercubes with various expansion costs and dilation costs [S87, HJ87, BMS87], simulations of rectangular meshes in square meshes [Ata85], and simulations among rectangular meshes [KA85]. (In a *simulation* of G in H, a constant number of nodes in G can be mapped into a single node in H; thus, a simulation is not an injection but a many-to-one mapping.) With the exceptions of [Ata85, KA85], in which the *costs* are expressed in terms of big O notation (referring to the asymptotic behavior of an embedding), the *costs* in the papers cited above and in this thesis are all exact.

Based on the sequential computation model, our basic embeddings, generalized embeddings for increasing dimension, and generalized embeddings for lowering dimension all have complexity $O(cn)$; our embeddings through simple reduction have complexity $O(dn)$, where d is the dimension of G, c is the dimension of H, and n is the size of G and H.

Chapter 4
Program Loading and Data Routing

4.1 Introduction

In the preceding chapter, we designed efficient graph embedding schemes to minimize dilation cost. In the corresponding program mapping problem, this dilation cost is the number of system cycles required for a single process in the task graph to send a message to one of its neighboring processes in the worst case.

But graph embedding is not an adequate model of the program mapping problem. If more than one process in the task graph attempts to send a message to one of its neighboring processes at the same time, some link may be required by more than one message at the same time. Since each unidirectional link can support transmission of only one message at any instant, extra delay may be introduced. We call the problem of incurring extra delay because of link conflicts the *link contention problem.*

In this chapter we investigate the mapping of parallel programs onto parallel processing systems. We use a task graph to represent a parallel program, and a system graph to represent a parallel processing system. The nodes in the task graph represent the processes in the parallel program, and the edges in the task graph represent the communication requirements between pairs of processes. The nodes in the system graph represent the processors in the parallel processing system, and the edges in the system graph represent the physical links in the parallel processing system. We do not distinguish a node from its address unless ambiguity might occur. The emphasis of this chapter will be on (1) the parallel loading of the code for each process into the corresponding processor under particular embedding schemes, and (2) the conflict-free data routing in the system graph to simulate a large class of parallel neighboring communications in the task graph after the program is mapped into the system. The first topic addresses the implementation of the embedding of the task graph into the system graph. The second topic addresses the extension of the dilation cost analysis to the data routing complexity analysis and the data routing implementation. The related topics discussed in this chapter include:

1. How can the physical nodes in the system graph calculate the addresses of their guest nodes in the task graph in parallel?

2. How to establish translation tables in the physical nodes in parallel to support at execution time parallel neighboring communications on the task graph level?

3. For each graph embedding function, how to design conflict-free data routing schemes to support parallel neighboring communications in the task graph, and what are the complexities of these schemes under different link communication models?

4.2 Assumptions and Definitions

Parallel logical neighboring communications

By definition, all communication requirements are represented directly as edges in the task graph. Therefore, we need only to consider data routing in the system graph to support *neighboring* communications in the task graph. We call the communications with addresses in the task graph *communications on the logical level,* and the communications with addresses in the system graph *communications on the physical level.* When we say to *simulate* in the system graph a set of parallel neighboring communications in the task graph, we mean to satisfy all of the communication requirements in the set through the paths in the system graph that connect images of neighboring nodes in the task graph.

We define the following two types of sets of parallel neighboring communications in the task graph:

Permutation type: At any instant, each node in the task graph can send out at most one message to one of its neighbors and receive at most one message from one of its neighbors.

Scatter type: At any instant, each node in the task graph can send out one message to each of its neighbors and receive one message from each of its neighbors.

By definition, permutation type sets of parallel neighboring communications are more restrictive than scatter type sets of parallel neighboring communications. If the nodes in the system graph have smaller degree than the corresponding nodes in the task graph, the link conflicts in the simulation of scatter type sets of parallel neighboring communications are unavoidable. In this chapter, we study the simulation in the system graph of any scatter type set of parallel neighboring communications in the task graph if the task graph is embedded into the system graph with unit dilation cost, and the simulation in the system graph of any permutation type set of parallel neighboring communications in the task graph otherwise.

In this thesis, we assume that the communication requirements of a parallel program are always in the form of either permutation type sets or scatter type sets of parallel neighboring communications in the task graph.

Assumptions about the system

In this thesis, we do not distinguish a system graph and a physical parallel processing system. We use the term *physical nodes* to denote the nodes in the system graph, and the term *logical nodes* to denote the nodes in the task graph. We make the following assumptions about the system architecture:

1. Each physical node has two message buffers associated with each bidirectional physical link, and one message buffer associated with each unidirectional physical link. Each of the buffers is capable of accommodating a message for either input or output.

2. Each physical node knows its address in the system graph. We also call this address the *physical address.*

3. Each physical node has a *translation table* in which each entry corresponds to a logical neighbor of the logical node embedded into this physical node. Suppose that an entry corresponds to the logical neighboring node v. This entry will consist of a pair of addresses: the first is the logical address of v in the task graph, and the second is the physical address of the physical node accommodating logical node v. This table is used to translate at execution time the communications on the logical task graph level to the communications on the physical system graph level. Upon receiving a request to send a message to one of the logical neighbors, the physical node will look up the translation table and automatically translate the logical address for this logical neighbor into the corresponding physical address. We will show how to construct these translation tables in Section 4.4.

4. We use packet switching for data routing. Each message carries the physical address of its destination.

5. We assume that the host of the parallel processing system can broadcast a message to all of the physical nodes under its control. No other communication networks between the host and the physical nodes are assumed.

6. We assume three possible communication modes for the links of the parallel processing system:

   Mode 1: Each link can independently support communications in both directions, and two messages can be sent in opposite directions over the same link at the same time.

   Mode 2: Each link can independently support communications in both directions, but two messages cannot be sent in opposite directions over the same link at the same time.

   Mode 3: At any instant, only those links along a single dimension can support concurrent communications in a single direction.

In the remainder of this section and in Chapter 5, we assume that all of the links work in communication mode 1 unless stated otherwise.

Coordinated parallel data movements

Here we define two kinds of *coordinated parallel data movements* on the system graph to be used later for data routing.

When we say to perform a *unidirectional coordinated parallel data movement along the i-th dimension to the left (right) for k steps,* we mean that (i) at the beginning of the operation, each physical node may identify a message for moving towards the left (right) in the i-th dimension; (ii) all of the identified messages may move along the specified single direction in the i-th dimension in parallel, with each message participating in at most k such parallel movement steps; (iii) an identified message will participate in a parallel movement step *if and only if* this movement step will make it approach its destination along a shortest path connecting its source and destination.

Note that in a coordinated parallel data movement, the movement of every message is consecutive. If it stops, it will not move further in its specified direction in that coordinated parallel data movement.

For example, let us assume that the system graph H is a torus of shape $(8, 8)$; for all $i \in [4]$ and $j \in [8]$, node (i, j) has a message $A(i, j)$ with node $((i-2) \bmod 8, (j+1) \bmod 8)$ as its destination; and for all $i \in \{4, 5, 6, 7\}$ and $j \in [8]$, node (i, j) has a message $A(i, j)$ with node $((i-3) \bmod 8, j)$ as its destination. Assume that we perform a unidirectional coordinated parallel data movement along the first dimension to the left for three steps, then perform another unidirectional coordinated parallel data movement along the second dimension to the right for one step. In the first coordinated parallel data movement, we assume that for all $i, j \in [8]$, node (i, j) identifies message $A(i, j)$ in it to participate in the data movement. Upon the completion of this first coordinated parallel data movement, for all $i \in [4]$ and $j \in [8]$, message $A(i, j)$ will be in node $((i-2) \bmod 8, j)$, and for all $i \in \{4, 5, 6, 7\}$ and $j \in [8]$, message $A(i, j)$ will be in node $((i-3) \bmod 8, j)$. In the second coordinated parallel data movement, we assume that for all $i \in [4]$ and $j \in [8]$, node $((i-2) \bmod 8, j)$ identifies message $A(i, j)$ in it to participate in the data movement. Upon the completion of this second coordinated parallel data movement, for all $i \in [4]$ and $j \in [8]$, message $A(i, j)$ will be in node $((i-2) \bmod 8, (j+1) \bmod 8)$, and for all $i \in \{4, 5, 6, 7\}$ and $j \in [8]$, message $A(i, j)$ will still be in node $((i-3) \bmod 8, j)$.

When we say to perform a *bidirectional coordinated parallel data movement along the i-th dimension for k steps,* we mean that (i) at the beginning of the operation, each physical node may identify a message in it for moving towards the left in the i-th dimension, and also identify a message in it for moving towards the right in the i-th dimension; (ii) all of the identified messages may move along their specified directions in the i-th dimension in parallel, with each message participating in at most k such parallel movement steps; (iii) an identified message will participate in a parallel movement step *if and only if* that movement will make it approach its destination along a shortest path connecting its source and destination.

For example, let us again assume that the system graph H is a torus of shape $(8, 8)$, and for all $i, j \in [8]$, node (i, j) has a message $A(i, j)$ with node $((i-2) \bmod 8, (j+1) \bmod 8)$ as its destination, and also a message $B(i, j)$ with node $((i+3) \bmod 8, (j-1) \bmod 8)$ as its destination. Assume that we perform a bidirectional coordinated parallel data movement along the first dimension for three steps. For all $i, j \in [8]$, we assume that node (i, j) identifies message $A(i, j)$ in it to participate in the data movement towards the left and message $B(i, j)$ in it to participate in the data movement towards the right. Upon the completion of the coordinated parallel data movement, message $A(i, j)$ will be in node $((i-2) \bmod 8, j)$ for all $i, j \in [8]$, and message $B(i, j)$ will be in node $((i+3) \bmod 8, j)$ for all $i, j \in [8]$.

By the definitions, in any unidirectional or bidirectional coordinated parallel data movement, different messages use different links at different times. Therefore, no link conflicts will occur. For the same reason, more than one unidirectional or bidirectional coordinated parallel data movement can be performed simultaneously without link conflicts as long as they are along distinct dimensions. In this dissertation, all data routing algorithms consist of one or more consecutive phases, and within each phase, one or more coordinated parallel data movements are performed simultaneously, all in distinct dimensions of the system graph. Therefore, to prove that these algorithms are link conflict free, we need only to verify that at the beginning of each phase, there is only one message that needs to be sent out in any direction.

We now establish two lemmas about coordinated parallel data movements, which will be used in the proofs for our data routing algorithms.

Lemma 4.2.1 *Let G be a task graph, H be a system graph, and S be a permutation type set of parallel neighboring communications in G. Let ρ be any positive integer. Assume that after a one-to-one embedding of G into H, for every message in S, its source and destination lie along one of the dimensions of H and are at a distance of at most ρ. Then S can be simulated in H by performing simultaneously along each dimension of H a bidirectional coordinated parallel data movement for ρ steps.* □

Proof. We need only to prove that there are no link conflicts in the parallel data movements above. By the definition of permutation type sets of parallel neighboring communications, each node in H can be the source of at most one message in S. We also know that simultaneous bidirectional coordinated parallel data movements are link conflict free as long as they are along distinct dimensions. Therefore the lemma is true. □

Lemma 4.2.2 *Let G be a task graph, H be a system graph, and S be a permutation type set of parallel neighboring communications in G. Let ρ be any positive integer. Assume that after a one-to-one embedding of G into H, for every message s in S, its source u and destination v satisfy the following conditions:*

(a) *there is at least one shortest path connecting u and v in H that follows at most two dimensions;*

(b) *any shortest path connecting u and v in H that follows one dimension has length either ρ or 2ρ along that dimension;*

(c) *any shortest path connecting u and v in H that follows two dimensions has length ρ along either of the two dimensions.*

Then S can be simulated in H by first performing simultaneously along each dimension of H a bidirectional coordinated parallel data movement for ρ steps, then performing simultaneously along each dimension of H another bidirectional coordinated parallel data movement for ρ steps. □

Proof. Using an argument similar to the proof for Lemma 4.2.1, we know that at the end of the first coordinated parallel data movement, every message in S with source and destination at a distance of ρ has reached its destination, and each of the other messages in S has reached its mid-point node, which is along the same dimension as its destination and at a distance of exactly ρ from both its source and destination.

Since S is of permutation type, every node in H can be the destination of at most one message in S. Furthermore, since each message that has not reached its destination is now in a mid-point node at a distance of exactly ρ from its destination, every mid-point node has at most one message to deliver in each direction along every dimension of H. Therefore, all of the remaining messages can reach their destinations by performing simultaneously along each dimension of H another bidirectional coordinated parallel data movement for ρ steps. □

4.3 Logical Address Identification and Program Loading

We use the term *code* to denote the piece of program generating a process. Given a parallel program in the form of a task graph, each node has a logical address and a code type. The code type determines which code will be executed to generate the process represented by the corresponding logical node. If two nodes have the same code type, they will execute different copies of the same code. In the SIMD environment, all logical nodes execute different copies of the same code and thus have the same code type. In the MIMD environment, each logical node may execute different code and thus may have different code types.

When we want to execute a parallel program on a parallel processing system, we first have to allocate the logical nodes to the physical nodes according to some embedding function. Since different systems usually have different system support for communication between the host and the physical nodes, and different parallel programs usually have different code allocation patterns, the code loading problem is not trivial and has not yet been well treated in the literature.

One possible approach is to have the host calculate for each logical node the physical node to which the logical node will be assigned, and then use some network to load the appropriate codes, together with their associated logical addresses, into the physical nodes to which the logical nodes are assigned. For large task graphs, this approach imposes a heavy computational load on the system host because the host must sequentially evaluate the embedding function for each logical node. The delay introduced by this sequential computation can prohibit the application of our embedding schemes at execution time.

In this research, since all of the embedding functions are well-defined mappings, we propose a parallel approach to solve the code loading problem above. Our approach is based on the inverses of the embedding functions. We assume that broadcast is the only means for the host to send messages to the physical nodes. Since a broadcast network is available or simulated in all SIMD, MSIMD, and MIMD systems, this approach is applicable to a wide range of parallel processing systems.

In the first step, the host broadcasts the inverse of the embedding function as well as the parameters of the subsystem for the current task to all of the physical nodes in the subsystem. Using the inverse function with its own unique physical address as argument, each processor then computes the logical address of the node in the task graph to be embedded into this processor. Such computations are performed by all processors in parallel. Since all of our embedding functions have low complexity and these computations are performed in parallel, the delay introduced by this step is small.

In the next step, the host first broadcasts a special code called *loader*, which is basically a table specifying the code type for each logical node, to all of the physical nodes in the subsystem, and then broadcasts sequentially the codes of different types used in the parallel program to all of the physical nodes in the subsystem. Each physical node decides independently whether it should ignore or accept the incoming code based on the information in the loader. The delay introduced by this step is proportional to the number of code types and the lengths of these codes used by the parallel program. As a special case, for SIMD programs, only one piece of code needs to be broadcast. This approach is especially usefull when either the parallel program is large but has only a limited number of different code types, or broadcasting is the only means for the host to send messages to all of the physical nodes.

4.3.1 Inverses of Embedding Functions

In this subsection, we show that all of our embedding functions have simple inverse functions. Since the permutations applied before or after an embedding function are trivially invertible, we ignore these permutations to simplify our exposition.

Inverse function for $f_{\mathcal{L}}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and $n = l_1 l_2 \cdots l_d$. We can redefine the transformation in Definition 3.3.1 as a function $\tau_{\mathcal{L}} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{L}}$. For all $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d) \in \Omega_{\mathcal{L}}$, let x be $u_{\mathcal{L}}^{-1}((\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d))$, $\tau_{\mathcal{L}}((\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)) = (x_1, x_2, \ldots, x_d)$, where for all $i \in [d]^+$,

$$x_i = \begin{cases} \hat{x}_i, & \text{if } \lfloor x/w_{i-1} \rfloor \text{ is even;} \\ l_i - \hat{x}_i - 1, & \text{if } \lfloor x/w_{i-1} \rfloor \text{ is odd.} \end{cases}$$

By this definition, we can write the embedding function $f_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ as a composition of the function $u_{\mathcal{L}}$ and the function $\tau_{\mathcal{L}}$: for all $x \in [n]$,

$$f_{\mathcal{L}}(x) = \tau_{\mathcal{L}}(u_{\mathcal{L}}(x)).$$

Since $u_{\mathcal{L}}$ and $\tau_{\mathcal{L}}$ are both bijections, the inverse function of $f_{\mathcal{L}}$, $f_{\mathcal{L}}^{-1} : \Omega_{\mathcal{L}} \to [n]$, can be expressed as

$$f_{\mathcal{L}}^{-1}((x_1, x_2, \ldots, x_d)) = u_{\mathcal{L}}^{-1}(\tau_{\mathcal{L}}^{-1}((x_1, x_2, \ldots, x_d))),$$

where $u_{\mathcal{L}}^{-1}$ is the inverse function of $u_{\mathcal{L}}$, $\tau_{\mathcal{L}}^{-1}$ is the inverse function of $\tau_{\mathcal{L}}$, and $(x_1, x_2, \ldots, x_d)$ is any number in $\Omega_{\mathcal{L}}$.

As we noted in Section 3.2, for all $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d) \in \Omega_{\mathcal{L}}$,

$$u_{\mathcal{L}}^{-1}((\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)) = \sum_{k=1}^{d} \hat{x}_k w_k.$$

Let $\pi_{i,j} = \prod_{p=i}^{j} l_p$. The function $\tau_{\mathcal{L}}^{-1} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{L}}$ can be specified in the following way: for all $(x_1, x_2, \ldots, x_d) \in \Omega_{\mathcal{L}}$, $\tau_{\mathcal{L}}^{-1}((x_1, x_2, \ldots, x_d)) = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_d)$, where for all $i \in [d]^+$,

$$\hat{x}_i = \begin{cases} x_i, & \text{if } \sum_{k=1}^{i-1} \hat{x}_k \pi_{k+1,i-1} \text{ is even;} \\ l_i - x_i - 1, & \text{otherwise.} \end{cases}$$

To prove that $\tau_{\mathcal{L}}^{-1}$ is the inverse of $\tau_{\mathcal{L}}$, we need only to show that $\lfloor x/w_{i-1} \rfloor = \sum_{k=1}^{i-1} \hat{x}_k \pi_{k+1,i-1}$. We can rewrite $\lfloor x/w_{i-1} \rfloor$ as

$$\begin{aligned} \left\lfloor \frac{\sum_{k=1}^{d} \hat{x}_k \pi_{k+1,d}}{\pi_{i,d}} \right\rfloor &= \left\lfloor \frac{\sum_{k=1}^{i-1} \hat{x}_k \pi_{k+1,d} + \sum_{k=i}^{d} \hat{x}_k \pi_{k+1,d}}{\pi_{i,d}} \right\rfloor \\ &= \left\lfloor \sum_{k=1}^{i-1} \hat{x}_k \pi_{k+1,i-1} + \sum_{k=i}^{d} \frac{\hat{x}_k}{\pi_{i,k}} \right\rfloor . \end{aligned}$$

Since $\sum_{k=1}^{i-1} \hat{x}_k \pi_{k+1,i-1}$ is an integer, and

$$\begin{aligned}
\sum_{k=i}^{d} \frac{\hat{x}_k}{\pi_{i,k}} &\leq \sum_{k=i}^{d} \frac{l_k - 1}{\pi_{i,k}} \\
&= \left(1 - \frac{1}{l_i}\right) + \left(\frac{1}{l_i} - \frac{1}{l_i l_{i+1}}\right) + \cdots + \left(\frac{1}{l_i l_{i+1} \cdots l_{d-1}} - \frac{1}{l_i l_{i+1} \cdots l_d}\right) \\
&= 1 - \frac{1}{l_i l_{i+1} \cdots l_d} \\
&< 1,
\end{aligned}$$

we have

$$\lfloor x/w_{i-1} \rfloor = \sum_{k=1}^{i-1} \hat{x}_k \pi_{k+1,i-1}.$$

Therefore, the inverse function is correct.

Note that in the specification of $\tau_{\mathcal{L}}^{-1}$, we have $\hat{x}_1 = x_1$, and for all $i \in \{2, 3, \ldots, d\}$, the value of $\hat{x}_i$ depends on the values of $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{i-1}$. By computing the $\hat{x}_i$'s in increasing order of i, all of the values needed in the computation of $\hat{x}_i$, for all $i \in \{2, 3, \ldots, d\}$, will be available before its computation.

Inverse function for $g_{\mathcal{L}}$

The embedding function $g_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows:

$$g_{\mathcal{L}}(x) = f_{\mathcal{L}}(t_n(x)),$$

where function $t_n : [n] \to [n]$ is defined as follows: for all $x \in [n]$,

if n is even, then

$$t_n(x) = \begin{cases} 2x, & \text{if } x < n/2; \\ n - 2(x - n/2) - 1, & \text{otherwise;} \end{cases}$$

if n is odd, then

$$t_n(x) = \begin{cases} 2x, & \text{if } x < (n+1)/2; \\ n - 2(x - (n+1)/2) - 2, & \text{otherwise.} \end{cases}$$

Since $f_{\mathcal{L}}$ and t_n are both bijections, we can write the inverse function of $g_{\mathcal{L}}$, $g_{\mathcal{L}}^{-1} : \Omega_{\mathcal{L}} \to [n]$, as

$$g_{\mathcal{L}}^{-1}((y_1, y_2, \ldots, y_d)) = t_n^{-1}(f_{\mathcal{L}}^{-1}((y_1, y_2, \ldots, y_d)))$$

for all $(y_1, y_2, \ldots, y_d) \in \Omega_{\mathcal{L}}$.

By the definition of t_n, we know that for all $y \in [n]$,

$$t_n^{-1}(y) = \begin{cases} y/2, & \text{if } y \text{ is even;} \\ n - (y+1)/2, & \text{otherwise.} \end{cases}$$

Inverse function for $r_{\mathcal{L}}$

Let $\mathcal{L} = (l_1,\ l_2)$ be a radix-base, and let $n = l_1 l_2$. The function $r_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows: for all $x \in [n]$,

if $l_2 > 2$, then

$$r_{\mathcal{L}}(x) = \begin{cases} (l_1 - 1 - x, 0), & \text{if } x < l_1; \\ (x_1, x_2 + 1) \text{ where } (x_1, x_2) = f_{(l_1, l_2 - 1)}(x - l_1), & \text{if } x \geq l_1; \end{cases}$$

if $l_2 = 2$, then

$$r_{\mathcal{L}}(x) = \begin{cases} (l_1 - 1 - x, 0), & \text{if } x < l_1; \\ (x - l_1, 1), & \text{if } x \geq l_1. \end{cases}$$

The inverse function of $r_{\mathcal{L}}$, $r_{\mathcal{L}}^{-1} : \Omega_{\mathcal{L}} \to [n]$, can be specified as follows: for all $(y_1, y_2) \in \Omega_{\mathcal{L}}$,

if $l_2 > 2$, then

$$r_{\mathcal{L}}^{-1}((y_1, y_2)) = \begin{cases} l_1 - 1 - y_1, & \text{if } y_2 = 0; \\ f_{(l_1, l_2 - 1)}^{-1}((y_1, y_2 - 1)) + l_1, & \text{otherwise}; \end{cases}$$

if $l_2 = 2$, then

$$r_{\mathcal{L}}^{-1}((y_1, y_2)) = \begin{cases} l_1 - 1 - y_1, & \text{if } y_2 = 0; \\ y_1 + l_1, & \text{otherwise.} \end{cases}$$

Inverse function for $h_{\mathcal{L}}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base, and let $n = \prod_{i=1}^{d} l_i$. The function $h_{\mathcal{L}} : [n] \to \Omega_{\mathcal{L}}$ is defined as follows: for all $x \in [n]$,

if $d \geq 3$, then let $\mathcal{L}' = (l_1,\ l_2)$, $\mathcal{L}'' = (l_3,\ l_4, \ldots,\ l_d)$, $m = \prod_{i=3}^{d} l_i$, $a = \lfloor x/(l_1 l_2 - 1) \rfloor$, $b = x \bmod (l_1 l_2 - 1)$, and

$$h_{\mathcal{L}}(x) = \begin{cases} r_{\mathcal{L}'}(b) \diamond f_{\mathcal{L}''}(a), & \text{if } x < m(l_1 l_2 - 1) \text{ and } a \text{ is even}; \\ r_{\mathcal{L}'}(l_1 l_2 - b - 2) \diamond f_{\mathcal{L}''}(a), & \text{if } x < m(l_1 l_2 - 1) \text{ and } a \text{ is odd}; \\ r_{\mathcal{L}'}(l_1 l_2 - 1) \diamond f_{\mathcal{L}''}(n - x - 1), & \text{otherwise}; \end{cases}$$

if $d = 2$, then $\quad h_{\mathcal{L}}(x) = r_{\mathcal{L}}(x)$; $\quad$ and
if $d = 1$, then $\quad h_{\mathcal{L}}(x) = x$.

The inverse function of $h_{\mathcal{L}}$, $h_{\mathcal{L}}^{-1} : \Omega_{\mathcal{L}} \to [n]$, can be specified as follows: for all $(y_1, y_2, y_3, \ldots, y_d) \in \Omega_{\mathcal{L}}$,

if $d \geq 2$, let $\mathcal{L}' = (l_1, l_2)$, $\mathcal{L}'' = (l_3, l_4, \ldots, l_d)$, $a = f_{\mathcal{L}''}^{-1}((y_3, \ldots, y_d))$, and

$$b = \begin{cases} r_{\mathcal{L}'}^{-1}((y_1, y_2)), & \text{if } a \text{ is even}; \\ l_1 l_2 - r_{\mathcal{L}'}^{-1}((y_1, y_2)) - 2, & \text{otherwise}; \end{cases}$$

$$h_{\mathcal{L}}^{-1}((y_1, y_2, y_3, \ldots, y_d)) = \begin{cases} n - f_{\mathcal{L}''}^{-1}((y_3, \ldots, y_d)) - 1, & \text{if } (y_1, y_2) = r_{\mathcal{L}'}(l_1 l_2 - 1); \\ a(l_1 l_2 - 1) + b, & \text{otherwise}; \end{cases}$$

if $d = 2$, $\quad h_{\mathcal{L}}^{-1}((y_1, y_2)) = r_{\mathcal{L}}^{-1}((y_1, y_2))$; $\quad$ and
if $d = 1$, $\quad h_{\mathcal{L}}^{-1}(y_1) = y_1$.

Inverse function for $\mathcal{T}_\mathcal{L}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base. The function $\mathcal{T}_\mathcal{L} : \Omega_\mathcal{L} \to \Omega_\mathcal{L}$ is defined as follows: for all $(x_1, x_2, \ldots, x_d) \in \Omega_\mathcal{L}$,

$$\mathcal{T}_\mathcal{L}((x_1, x_2, \ldots, x_d)) = (t_{l_1}(x_1), t_{l_2}(x_2), \ldots, t_{l_d}(x_d)).$$

The inverse function of $\mathcal{T}_\mathcal{L}$, $\mathcal{T}_\mathcal{L}^{-1} : \Omega_\mathcal{L} \to \Omega_\mathcal{L}$, can be specified as follows: for all $(y_1, y_2, \ldots, y_d) \in \Omega_\mathcal{L}$,

$$\mathcal{T}_\mathcal{L}^{-1}((y_1, y_2, \ldots, y_d)) = (t_{l_1}^{-1}(y_1), t_{l_2}^{-1}(y_2), \ldots, t_{l_d}^{-1}(x_d)).$$

Inverse functions for $\mathcal{F}_\mathcal{V}$, $\mathcal{G}_\mathcal{V}$, and $\mathcal{H}_\mathcal{V}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases such that $\mathcal{M}$ is an expansion of $\mathcal{L}$ with an expansion factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \diamond \cdots \diamond \mathcal{V}_d$. The functions $\mathcal{F}_\mathcal{V}\colon \Omega_\mathcal{L} \to \Omega_{\tilde{\mathcal{V}}}$, $\mathcal{G}_\mathcal{V}\colon \Omega_\mathcal{L} \to \Omega_{\tilde{\mathcal{V}}}$, and $\mathcal{H}_\mathcal{V}\colon \Omega_\mathcal{L} \to \Omega_{\tilde{\mathcal{V}}}$ are defined as follows: for all $(i_1, i_2, \ldots, i_d) \in \Omega_\mathcal{L}$,

$$\mathcal{F}_\mathcal{V}((i_1, i_2, \ldots, i_d)) = f_{\mathcal{V}_1}(i_1) \diamond f_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond f_{\mathcal{V}_d}(i_d),$$

$$\mathcal{G}_\mathcal{V}((i_1, i_2, \ldots, i_d)) = g_{\mathcal{V}_1}(i_1) \diamond g_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond g_{\mathcal{V}_d}(i_d),$$

$$\mathcal{H}_\mathcal{V}((i_1, i_2, \ldots, i_d)) = h_{\mathcal{V}_1}(i_1) \diamond h_{\mathcal{V}_2}(i_2) \diamond \cdots \diamond h_{\mathcal{V}_d}(i_d).$$

Their inverse functions $\mathcal{F}_\mathcal{V}^{-1}$, $\mathcal{G}_\mathcal{V}^{-1}$, and $\mathcal{H}_\mathcal{V}^{-1}$ are all bijections from $\Omega_{\tilde{\mathcal{V}}}$ to $\Omega_\mathcal{L}$ and can be specified in terms of $f_\mathcal{L}^{-1}$, $g_\mathcal{L}^{-1}$, and $h_\mathcal{L}^{-1}$. For all $Y = (y_1, y_2, \ldots, y_c) \in \Omega_{\tilde{\mathcal{V}}}$, we decompose Y into d segments $Y_1, Y_2, \ldots, Y_d$ such that $Y = Y_1 \diamond Y_2 \diamond \cdots \diamond Y_d$ and $|Y_i| = |\mathcal{V}_i|$ for all $i \in [d]^+$. The inverse functions can be expressed as

$$\mathcal{F}_\mathcal{V}^{-1}(Y) = f_{\mathcal{V}_1}^{-1}(Y_1) \diamond f_{\mathcal{V}_2}^{-1}(Y_2) \diamond \cdots \diamond f_{\mathcal{V}_d}^{-1}(Y_d),$$

$$\mathcal{G}_\mathcal{V}^{-1}(Y) = g_{\mathcal{V}_1}^{-1}(Y_1) \diamond g_{\mathcal{V}_2}^{-1}(Y_2) \diamond \cdots \diamond g_{\mathcal{V}_d}^{-1}(Y_d),$$

$$\mathcal{H}_\mathcal{V}^{-1}(Y) = h_{\mathcal{V}_1}^{-1}(Y_1) \diamond h_{\mathcal{V}_2}^{-1}(Y_2) \diamond \cdots \diamond h_{\mathcal{V}_d}^{-1}(Y_d).$$

Inverse functions for $\mathcal{F}'_\mathcal{S}$, $\mathcal{H}'_\mathcal{S}$, $\mathcal{G}'_\mathcal{S}$, $\mathcal{H}''_\mathcal{S}$, and $\mathcal{G}''_\mathcal{S}$

Let d and c be positive integers such that $c < d \leq 2c$. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases. Assume that $\mathcal{M}$ is a general reduction of $\mathcal{L}$ with a reduction factor $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$, multiplicant sublist $\mathcal{L}'$, and multiplier sublist $\mathcal{L}''$. Let $\alpha : [d]^+ \to [d]^+$ be a permutation such that $\alpha(\mathcal{L}) = \mathcal{L}' \diamond \mathcal{L}''$. Let $\tilde{\mathcal{S}} = (s_1, s_2, \ldots, s_b) = \mathcal{S}_1 \diamond \mathcal{S}_2 \diamond \cdots \diamond \mathcal{S}_{d-c}$, and let $\mathcal{I} = (\underbrace{1, 1, \ldots, 1}_{c-b})$. Let $\mathcal{F}_\mathcal{S} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$, $\mathcal{H}_\mathcal{S} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$, and $\mathcal{G}_\mathcal{S} : \Omega_{\mathcal{L}''} \to \Omega_{\tilde{\mathcal{S}}}$. The functions $\mathcal{F}'_\mathcal{S}\colon \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$, $\mathcal{H}'_\mathcal{S}\colon \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$,

$\mathcal{G}'_\mathcal{S}\colon \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'}$, $\mathcal{H}''_\mathcal{S}\colon \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'}$, and $\mathcal{G}''_\mathcal{S}\colon \Omega_{\mathcal{L}' \diamond \mathcal{L}''} \to \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'}$ are defined as follows: for all $(i_1, i_2, \ldots, i_d) \in \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$,

$$\mathcal{F}'_\mathcal{S}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{F}_\mathcal{S}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2} \ldots, i_c),$$

$$\mathcal{H}'_\mathcal{S}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{H}_\mathcal{S}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2} \ldots, i_c),$$

$$\mathcal{G}'_\mathcal{S}((i_1, i_2, \ldots, i_d)) = [(s_1 i_1, s_2 i_2, \ldots, s_b i_b) + \mathcal{G}_\mathcal{S}((i_{c+1}, i_{c+2}, \ldots, i_d))] \diamond (i_{b+1}, i_{b+2} \ldots, i_c),$$

$$\begin{aligned}\mathcal{H}''_\mathcal{S}((i_1, i_2, \ldots, i_d)) \quad &= \quad [(s_1 t_{l_{\alpha(1)}}(i_1), s_2 t_{l_{\alpha(2)}}(i_2), \ldots, s_b t_{l_{\alpha(b)}}(i_b)) + \mathcal{H}_\mathcal{S}((i_{c+1}, i_{c+2}, \ldots, i_d))] \\ &\quad\ \diamond (t_{l_{\alpha(b+1)}}(i_{b+1}), t_{l_{\alpha(b+2)}}(i_{b+2}) \ldots, t_{l_{\alpha(c)}}(i_c)),\end{aligned}$$

$$\begin{aligned}\mathcal{G}''_\mathcal{S}((i_1, i_2, \ldots, i_d)) \quad &= \quad [(s_1 t_{l_{\alpha(1)}}(i_1), s_2 t_{l_{\alpha(2)}}(i_2), \ldots, s_b t_{l_{\alpha(b)}}(i_b)) + \mathcal{G}_\mathcal{S}((i_{c+1}, i_{c+2}, \ldots, i_d))] \\ &\quad\ \diamond (t_{l_{\alpha(b+1)}}(i_{b+1}), t_{l_{\alpha(b+2)}}(i_{b+2}) \ldots, t_{l_{\alpha(c)}}(i_c)).\end{aligned}$$

The inverse function of $\mathcal{F}'_\mathcal{S}$, $\mathcal{F}'^{-1}_\mathcal{S} : \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'} \to \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$, can be specified as follows: for all $(j_1, j_2, \ldots, j_c) \in \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'}$,

$$\mathcal{F}'^{-1}_\mathcal{S}((j_1, j_2, \ldots, j_c)) = (i_1, i_2, \ldots, i_d),$$

where for all $k \in [b]^+$,

$$i_k = \lfloor j_k / s_k \rfloor,$$

for all $k \in \{b+1, b+2, \ldots, c\}$,

$$(i_{b+1}, i_{b+2}, \ldots, i_c) = (j_{b+1}, j_{b+2}, \ldots, j_c),$$

and for all $k \in \{c+1, c+2, \ldots, d\}$,

$$(i_{c+1}, i_{c+2}, \ldots, i_d) = \mathcal{F}^{-1}_\mathcal{S}((j_1 \bmod s_1, j_2 \bmod s_2, \ldots, j_b \bmod s_b)).$$

The inverse function of $\mathcal{H}'_\mathcal{S}$, $\mathcal{H}'^{-1}_\mathcal{S} : \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'} \to \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$, can be specified as follows: for all $(j_1, j_2, \ldots, j_c) \in \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}]\times\mathcal{L}'}$,

$$\mathcal{H}'^{-1}_\mathcal{S}((j_1, j_2, \ldots, j_c)) = (i_1, i_2, \ldots, i_d),$$

where for all $k \in [b]^+$,

$$i_k = \lfloor j_k / s_k \rfloor,$$

for all $k \in \{b+1, b+2, \ldots, c\}$,

$$(i_{b+1}, i_{b+2}, \ldots, i_c) = (j_{b+1}, j_{b+2}, \ldots, j_c),$$

and for all $k \in \{c+1, c+2, \ldots, d\}$,

$$(i_{c+1}, i_{c+2}, \ldots, i_d) = \mathcal{H}^{-1}_\mathcal{S}((j_1 \bmod s_1, j_2 \bmod s_2, \ldots, j_b \bmod s_b)).$$

The inverse function of $\mathcal{G}'_{\mathcal{S}}$, $\mathcal{G}'^{-1}_{\mathcal{S}} : \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'} \rightarrow \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$, can be specified as follows: for all $(j_1, j_2, \ldots, j_c) \in \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$,

$$\mathcal{G}'^{-1}_{\mathcal{S}}((j_1, j_2, \ldots, j_c)) = (i_1, i_2, \ldots, i_d),$$

where for all $k \in [b]^+$,

$$i_k = \lfloor j_k / s_k \rfloor,$$

for all $k \in \{b+1, b+2, \ldots, c\}$,

$$(i_{b+1}, i_{b+2}, \ldots, i_c) = (j_{b+1}, j_{b+2}, \ldots, j_c),$$

and for all $k \in \{c+1, c+2, \ldots, d\}$,

$$(i_{c+1}, i_{c+2}, \ldots, i_d) = \mathcal{G}^{-1}_{\mathcal{S}}((j_1 \bmod s_1, j_2 \bmod s_2, \ldots, j_b \bmod s_b)).$$

The inverse function of $\mathcal{H}''_{\mathcal{S}}$, $\mathcal{H}''^{-1}_{\mathcal{S}} : \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'} \rightarrow \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$, can be specified as follows: for all $(j_1, j_2, \ldots, j_c) \in \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$,

$$\mathcal{H}''^{-1}_{\mathcal{S}}((j_1, j_2, \ldots, j_c)) = (i_1, i_2, \ldots, i_d),$$

where for all $k \in [b]^+$,

$$i_k = t^{-1}_{l_{\alpha(k)}}(\lfloor j_k / s_k \rfloor),$$

for all $k \in \{b+1, b+2, \ldots, c\}$,

$$(i_{b+1}, i_{b+2}, \ldots, i_c) = (j_{b+1}, j_{b+2}, \ldots, j_c),$$

and for all $k \in \{c+1, c+2, \ldots, d\}$,

$$(i_{c+1}, i_{c+2}, \ldots, i_d) = \mathcal{H}^{-1}_{\mathcal{S}}((j_1 \bmod s_1, j_2 \bmod s_2, \ldots, j_b \bmod s_b)).$$

The inverse function of $\mathcal{G}''_{\mathcal{S}}$, $\mathcal{G}''^{-1}_{\mathcal{S}} : \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'} \rightarrow \Omega_{\mathcal{L}' \diamond \mathcal{L}''}$, can be specified as follows: for all $(j_1, j_2, \ldots, j_c) \in \Omega_{[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'}$,

$$\mathcal{G}''^{-1}_{\mathcal{S}}((j_1, j_2, \ldots, j_c)) = (i_1, i_2, \ldots, i_d),$$

where for all $k \in [b]^+$,

$$i_k = t^{-1}_{l_{\alpha(k)}}(\lfloor j_k / s_k \rfloor),$$

for all $k \in \{b+1, b+2, \ldots, c\}$,

$$(i_{b+1}, i_{b+2}, \ldots, i_c) = (j_{b+1}, j_{b+2}, \ldots, j_c),$$

and for all $k \in \{c+1, c+2, \ldots, d\}$,

$$(i_{c+1}, i_{c+2}, \ldots, i_d) = \mathcal{G}^{-1}_{\mathcal{S}}((j_1 \bmod s_1, j_2 \bmod s_2, \ldots, j_b \bmod s_b)).$$

Inverse function for $\mathcal{U}_{\mathcal{V}}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases such that $\mathcal{M}$ is a simple reduction of $\mathcal{L}$ with a reduction factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_c)$. Let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_c$. The function $\mathcal{U}_{\mathcal{V}} : \Omega_{\tilde{\mathcal{V}}} \rightarrow \Omega_{\mathcal{M}}$ is defined as follows: for all $(i_1,\ i_2,\ \ldots,\ i_d) \in \Omega_{\tilde{\mathcal{V}}}$,

$$\mathcal{U}_{\mathcal{V}}((i_1,\ i_2,\ \ldots,\ i_d)) = {u_{\mathcal{V}_1}}^{-1}(I_1) \diamond {u_{\mathcal{V}_2}}^{-1}(I_2) \diamond \cdots \diamond\ {u_{\mathcal{V}_c}}^{-1}(I_c),$$

where I_1, I_2, …, I_c are partitions of $(i_1, i_2, \ldots, i_d)$ such that for all $k \in [c]^+$, $|I_k| = |\mathcal{V}_k|$, and $I_1 \diamond I_2 \diamond \cdots \diamond I_c = (i_1, i_2, \ldots, i_d)$.

The inverse function of $\mathcal{U}_{\mathcal{V}}$, $\mathcal{U}_{\mathcal{V}}^{-1} : \Omega_{\mathcal{M}} \rightarrow \Omega_{\tilde{\mathcal{V}}}$, can be specified as follows: for all $(j_1, j_2, \ldots, j_c) \in \Omega_{\mathcal{M}}$,

$$\mathcal{U}_{\mathcal{V}}^{-1}((j_1, j_2, \ldots, j_c)) = u_{\mathcal{V}_1}(j_1) \diamond u_{\mathcal{V}_2}(j_2) \diamond \cdots \diamond u_{\mathcal{V}_c}(j_c).$$

Inverse functions for multi-step embeddings

Given a graph G and a graph H, suppose that G is embedded into H through a sequence of k ($k > 1$) intermediate embedding steps and the embedding function $\mathcal{E} : V_G \rightarrow V_H$ can be expressed as

$$\mathcal{E} = \alpha_k \circ f_k \circ \alpha_{k-1} \circ f_{k-1} \circ \alpha_{k-2} \circ \cdots \circ \alpha_1 \circ f_1 \circ \alpha_0,$$

where the function composition operator "$\circ$" is right associative; for all $i \in [k]^+$, f_i is the embedding function applied in the i-th step of embedding; for all $j \in [k]^+$, α_j is a permutation of the dimensions in the range of f_i; and α_0 is a permutation of the dimensions in the graph G.

If for all $i \in [k]^+$, f_i is one of the embedding functions discussed earlier in this subsection, then its inverse function f_i^{-1} has already been well specified. Since all of these functions are bijections, the inverse function for $\mathcal{E}$ can be expressed as

$$\mathcal{E}^{-1} = \alpha_0^{-1} \circ f_1^{-1} \circ \alpha_1^{-1} \circ f_2^{-1} \circ \alpha_2^{-1} \circ \cdots \circ \alpha_{k-1}^{-1} \circ f_k^{-1} \circ \alpha_k^{-1},$$

where for all $j \in [k+1]$, α_j^{-1} is the inverse permutation of α_j.

All of the embedding functions implicitly defined in Section 3.5 of the preceding chapter satisfy the condition above. Thus, the inverse functions of these multi-step embedding functions can all be easily constructed from the inverse functions we described in this subsection.

4.3.2 Logical Address Identification

Logical address identification enables each physical node in the system graph to determine which logical node in the task graph will be embedded into it. Logical address identification provides the basis for the loading of the sequential codes into the physical nodes as well as for the support of logical level communication.

We now show how the logical address identification can be performed quickly in a partitionable parallel processing system. Let us assume that the global system graph is a

c-dimensional mesh (torus). Each task graph will be allocated a sub-mesh of the global system graph, which we call a partition, as the system graph to accommodate the task graph. Let the *base* of a partition be the physical address of the node in the partition that has the smallest index in each dimension among all physical nodes in that partition. A partition can be completely defined by its shape and base. If a partition has base $(b_1, b_2, \ldots, b_c)$ and shape $(m_1, m_2, \ldots, m_c)$, then the nodes in the partition have addresses of the form $(b_1 + i_1, b_2 + i_2, \ldots, b_c + i_c)$, where for all $k \in [c]^+$, $i_k \in [m_k]$.

Given a parallel program in the form of a task graph and an allocated system partition in the form of a system graph of the same size, let us assume that the task graph is of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and the partition has base $B = (b_1, b_2, \ldots, b_c)$ and shape $\mathcal{M} = (m_1, m_2, \ldots, m_c)$. We also assume that an embedding function $\mathcal{E} : \Omega_{\mathcal{L}} \rightarrow \Omega_{\mathcal{M}}$ is used for the embedding, and the inverse function of $\mathcal{E}$, $\mathcal{E}^{-1} : \Omega_{\mathcal{M}} \rightarrow \Omega_{\mathcal{L}}$, is known. The host first broadcasts a message containing $\mathcal{E}$, $\mathcal{E}^{-1}$, $\mathcal{L}$, $\mathcal{M}$, and B to all of the physical nodes in the system. Each physical node then checks the values of B and $\mathcal{M}$ and decides whether it belongs to the partition. If it does not belong to the partition, it will ignore all of the other steps for the setup of this parallel program. Therefore from now on, we can simply talk about "sending a message to all of the physical nodes in the partition", or say that "all of the physical nodes in the partition perform the following operation".

Using the inverse function $\mathcal{E}^{-1}$, all of the physical nodes in the partition calculate in parallel the logical addresses of the nodes in the task graph to be embedded into them. For any node in the partition with physical address $(i_1, i_2, \ldots, i_c)$, the address of the logical node to be embedded into it is

$$\mathcal{E}^{-1}((i_1 - b_1, i_2 - b_2, \ldots, i_c - b_c)).$$

4.3.3 Program Loading

So far, we have finished the first step in embedding the parallel program in the form of a task graph into the system graph: each physical node in the partition knows what is the address of the logical node to be embedded into it. The next step is actually to load the codes for the logical nodes into the corresponding physical nodes in the system graph. We call the problem in this step the *program loading problem.*

The solutions to the program loading problem depend on the communication networks between the host and the processors as well as the properties of the task graph itself. We now propose an approach based on a broadcast network.

Assume that there are n logical nodes in the task graph and all these nodes are based on k different types of sequential codes. We use a table of length n to specify for each logical address which code type will be used for the logical node. We call this table the *code type specification table.* This table can be either specified by the programmer or derived automatically from the parallel program by the compiler. We incorporate this table into the following simple code called "Loader":

```
Program Loader (In: message);
    var T: code_type_specification_table;
    begin
        If In.type = T(logical_address)
            then save In.code
            end;
```

The host broadcasts the Loader to all of the physical nodes in the partition, and then sequentially broadcasts all of the different codes prefixed with their unique code types to all of the physical nodes in the partition. Each physical node then uses the Loader to decide whether the incoming code has the same code type as specified in the code type specification table for the logical address assigned to it. If the two types agree, the physical node keeps the code; otherwise it discards it.

The time needed for this program loading process is proportional to the number of different code types used in the parallel program, but not to the size of the task graph. This approach works especially well for large parallel programs in which only a limited number of code types are used.

4.4 Parallel Generation of Translation Tables

If under an embedding scheme, a logical node X is mapped into a physical node Y, we call X the corresponding logical node of Y, and Y the corresponding physical node of X. Since we want to support efficiently at execution time parallel neighboring communications on the logical task graph level, each physical node must have a translation table to translate the logical destination addresses of the messages to their corresponding physical addresses under the embedding. Each entry in the table is an ordered pair "(logical address, physical address)". For each physical node, the length of this table equals the degree of the corresponding logical node. When a logical node needs to send a message to one of its logical neighbors, the corresponding physical node automatically looks up the translation table to determine the corresponding physical address of this logical neighbor, prefix the message with the ordered pair "(logical address, physical address)", then send the message to its physical destination by our data routing schemes discussed in the next section.

Here we propose two methods for the parallel generation of the translation tables. Method 1 is based on the parallel computation of the embedding function and can be applied to all of our embedding functions. Method 2 is based on parallel data movements of the ordered pairs "(logical address, physical address)" computed in the logical address identification stage and is used only if the embedding has unit dilation cost. We assume that each physical node in the partition already knows the embedding function $\mathcal{E}$, the shape of the task graph, its physical address $X = (x_1, x_2, \ldots, x_c)$, and the logical address $Y = (y_1, y_2, \ldots, y_d)$ of the corresponding node in the task graph.

Method 1: First, each physical node in the partition generates a list of all of the addresses of the logical neighbors of the corresponding logical node. This list consists of all of the addresses in the set

$$\{Y' = (y'_1, y'_2, \ldots, y'_d) | \forall k \in [d]^+,\ y'_k \in [l_k],\ \delta_m(Y, Y') = 1\}$$

if the task graph is a mesh, or all of the addresses in the set

$$\{Y' = (y'_1, y'_2, \ldots, y'_d) | \forall k \in [d]^+,\ y'_k \in [l_k],\ \delta_t(Y, Y') = 1\}$$

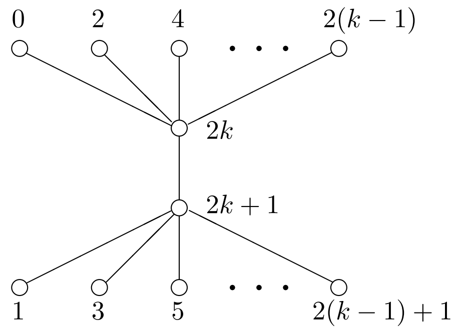


Figure 4.1: Example for the link contention problem

if the task graph is a torus. Then using the embedding function $\mathcal{E}$, each physical node computes the physical address for the address of each logical neighbor in the list above, and make each pair "(logical address of a logical neighbor, corresponding physical address)" an entry in the translation table. Both of the two steps above can be performed by all of the physical nodes in parallel. The time complexity of Method 1 is proportional to the product of d and the complexity of the embedding function.

The following Method 2 is used only if the embedding has unit dilation cost.

Method 2: Each physical node in the partition generates a message (Y, X), which is the pair of the logical and the physical addresses assigned to the node, and sends it to all of its physical neighbors. Each physical node then makes each of its incoming messages an entry in its translation table.

The time complexity of Method 2 depends on the communication mode of the links. By mode 1, only one unit of time is needed for broadcasting the messages. By mode 2, two units of time are needed for broadcasting the messages. By mode 3, $2c$ units of time are needed for broadcasting the messages. All of these will be explained in the following section when we discuss conflict-free data routing.

4.5 Conflict-free Data Routing

Given an embedding $\mathcal{E}$ of a task graph G into a c-dimensional system graph H with dilation cost ρ, in this section we consider the problem of how to simulate in the system graph H the parallel neighboring communications in the task graph G and provide the data routing complexities for these simulations. The main objective here is to avoid link conflicts to minimize the simulation steps.

In general, because of the link contention problem, a small dilation cost of an embedding does not imply that any set of parallel neighboring communications in the task graph can be simulated in the system graph with a small data routing complexity. For example, assume that G is a ring of size $2(k+1)$ for some integer $k > 1$, H is the graph given in Figure 4.1, and G is embedded into H by the scheme described in the figure. This embedding has a dilation cost of 3. We consider the following set of parallel neighboring communications in G: for all $i \in [k]$, node $2i$ sends a message to node $2i + 1$. It takes a minimum of $k+2$ parallel data routing steps to simulate in H this set of parallel neighboring communications in G. In this case, the dilation cost is a constant three, whereas the data routing complexity is an increasing function of the size of the task graph.

In this section, we first propose a simple data routing scheme that can automatically carry out our data routing strategies to simulate any permutation type or scatter type set of parallel neighboring communications in the task graph. We then design for each of our embedding functions a data routing strategy to achieve conflict-free simulation of parallel neighboring communication in the task graph, and analyze the correponding data routing complexities. We show that because of the regularity of our embedding schemes, for the embedding functions defined in Chapter 3, they can support the simulation of parallel neighboring communications in the task graph with the following data routing complexities: most with complexities equal to their dilation costs, and the others with complexities either equal to their dilation costs plus 1 or 2, or equal to twice or four times their dilation costs.

Given a function $\mathcal{E}$ for embedding a d-dimensional graph G of shape $\mathcal{L}$ into another c-dimensional graph H of shape $\mathcal{M}$, for any permutation α on $[d]^+$ and any permutation β on $[c]^+$, the function $\beta \circ \mathcal{E} \circ \alpha$ embeds a graph G' of shape $\alpha^{-1}(\mathcal{L})$ into another graph H' of shape $\beta(\mathcal{M})$. Since G and G' are isomorphic, and H and H' are isomorphic, the function $\mathcal{E}$ and the function $\beta \circ \mathcal{E} \circ \alpha$ have the same dilation cost and the same data routing complexities. To simplify our presentation, in the following subsections we consider only embedding functions without permutations surrounding them.

4.5.1 Shortest-path Data Routing Scheme

We assume that the communication requirements of parallel programs are always in the form of permutation type or scatter type sets of parallel neighboring communications in the task graph. Given a permutation type or scatter type set of parallel neighboring communications in the task graph, a *macro data routing cycle* is the time that the system takes to finish the simulation of all of the parallel neighboring communications in the set. The length of a macro data routing cycle depends on the nature of the embedding function as well as the set. A macro data routing cycle consists of one or more *parallel data movement steps.* Within each data movement step, a message moves from one processor to a neighboring processor through a physical link. We use the term *data routing complexity* to mean the number of parallel data movement steps that is sufficient to simulate in the system graph any permutation type or scatter type set of parallel neighboring communications in the task graph.

Our objective in this section is to achieve efficient data routing to support the parallel inter-process communication in parallel programs. Given a set of parallel neighboring communications in the task graph, we not only need a good data routing strategy to minimize the communication delay, but also need a good data routing mechanism so that the system can automatically carry out the routing strategies at execution time with minimum control overhead.

Because of the regularity of our embedding functions, we can use a very simple data routing scheme to automatically carry out each of the data routing strategies for our embedding functions (to be constructed in the following subsections). The general scheme is described as follows:

1. For each embedding function, we first design a *routing vector* $(i_1, i_2, \ldots, i_d)$, where d is the dimension of the task graph and i_k is a positive integer for all $k \in [d]^+$. The routing vector stores all of the information about our routing strategy for the embedding function.

2. Given a set S of parallel neighboring communications in the task graph, we partition S into as many subsets as the number of distinct integers in the routing vector: each subset consists of all of the messages in S with source and destination addresses differing in one of the dimensions in which the routing vector has the same integer.

3. Communications within any single subset of S described above are simulated at the same time. Communications belonging to different subsets of S described above are simulated at different time.

4. Let the *current node* of a message be the physical node in the system graph where the message currently stays. When the communications in a subset are simulated, each message involved repeats the following steps until it reaches its physical destination:

   (a) randomly choose a dimension of the system graph among those where the physical address of the current node and the physical address of the destination node of the message differ;

   (b) move along the chosen dimension to approach its physical destination until it reaches a physical node whose address has the same value in the chosen dimension as the address of the physical destination of the message;

   (c) wait in the current node until all of the messages in the subset finish step (b).

For example, suppose that the routing vector for an embedding function is $(1, 1, 2, 2)$, and

$$\begin{aligned} S = \quad & \{[(1,1,1,1),(1,2,1,1)],[(1,2,2,2),(0,2,2,2)],\\ & [(2,2,0,3),(2,2,1,3)],[(2,2,3,2),(2,3,3,2)],\\ & [(3,2,2,1),(3,2,2,0)],[(2,2,3,2),(3,2,3,2)],\\ & [(3,3,2,2),(3,3,1,2)],[(1,2,1,2),(1,2,1,3)]\} \end{aligned}$$

is a set of parallel neighboring communications in a task graph, where $[X, Y]$ means that logical node X needs to send a message to logical node Y. S will first be decomposed into sets

$$\begin{aligned} S_1 = \quad & \{[(1,2,2,2),(0,2,2,2)],[(2,2,3,2),(3,2,3,2)],\\ & [(1,1,1,1),(1,2,1,1)],[(2,2,3,2),(2,3,3,2)]\} \end{aligned}$$

and

$$\begin{aligned} S_2 = \quad & \{[(2,2,0,3),(2,2,1,3)],[(3,3,2,2),(3,3,1,2)],\\ & [(3,2,2,1),(3,2,2,0)],[(1,2,1,2),(1,2,1,3)]\}. \end{aligned}$$

All of the communications in S_1 will be simulated at the same time. All of the communications in S_2 will be simulated at the same time, either before or after the simulation of S_1.

As will be shown later, the regularity of our embedding functions guarantees that link conflicts can be avoided in the data routing scheme above. Since all of the messages will move to their destinations along the shortest paths without worrying about the contention of links with other messages, we call this data routing scheme the *shortest-path data routing scheme.*

In the following subsections, we design a routing vector for each of our embedding functions, show that there are no link conflicts in the routing process, and provide the data routing complexities for simulating in the system graph the parallel neighboring communications in the task graph. We use the coordinated parallel data movements to describe our routing strategies. When we perform a coordinated parallel data movement along a dimension for k steps, we imply that all messages will go along that dimension in their corresponding shortest paths for at most k steps. As will be proved in the following subsections, for all of our embedding functions, the addresses of the images of any pair of neighboring nodes in the task graph differ in exactly either one or two positions. Therefore, for any pair of neighboring nodes in the task graph, there is at least one shortest path corresponding to their images in the system graph that follows only one or two dimensions of the system graph, and we need only to perform one or two coordinated parallel data movements to send all of the messages to their destinations.

In the following discussion, we assume that all of the links work in communication mode 1 unless stated otherwise. The data routing complexities can be easily generalized to those for communication modes 2 or 3 by stepwise simulation. Let us assume that for a particular embedding scheme, the data routing complexity under communication mode 1 is ρ, and the system graph is a c-dimensional graph. Since each bidirectional coordinated parallel data movement can be simulated by two unidirectional coordinated parallel data movements, the data routing complexity for this embedding scheme under communication mode 2 will be bounded from above by 2ρ. Since each unidirectional coordinated parallel data movement along all of the dimensions can be simulated by c unidirectional coordinated parallel data movements, one for each dimension, the data routing complexity for this embedding scheme under communication mode 3 will be bounded from above by $2c\rho$.

4.5.2 Data Routing for Single-step Embeddings

We first consider the case in which the task graph is embedded into the system graph using one of our single-step embedding functions. We further divide the embedding functions in this category into three classes: those with unit dilation cost, those with dilation costs of 2, and those with dilation costs greater than 2.

Theorem 4.5.1 *Assume that a d-dimensional task graph G is embedded into a c-dimensional system graph H using some embedding function with unit dilation cost. Then any scatter type set of parallel neighboring communications in G can be simulated in H by one parallel data movement step. The routing vector for such an embedding function is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

Proof. Given any scatter type set of parallel neighboring communications in G, the system can perform a bidirectional coordinated parallel data movement along each dimension for one step at the same time. For every message involved in the communications, since its source and destination are at distance 1, and since the embedding is a one-to-one mapping, there must be a unique link on which this message can travel (one bidirectional link is considered to be two unidirectional links). Since different messages use different links, all of the messages can reach their destinations in this process without link conflicts. □

Since the embedding functions $f_{\mathcal{L}}$, $h_{\mathcal{L}}$, $\mathcal{F}_{\mathcal{V}}$, and $\mathcal{H}_{\mathcal{V}}$ all have unit dilation cost with respect to their appropriate domains and ranges as stated in Theorems 3.3.1, 3.3.3, and 3.4.1, we have the following corollary.

Corollary 4.5.1 *Assume that a d-dimensional task graph G is embedded into a c-dimensional system graph H using the function $f_{\mathcal{L}}$, $h_{\mathcal{L}}$, $\mathcal{F}_{\mathcal{V}}$, or $\mathcal{H}_{\mathcal{V}}$ with unit dilation cost. Then any set of scatter type parallel neighboring communications in G can be simulated in H by one parallel data movement step. The routing vector for these embedding functions is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

Corollary 4.5.2 *Let G be a d-dimensional square torus or a d-dimensional square mesh with each dimension being of length ℓ, and let H be a c-dimensional square torus or a c-dimensional square mesh of the same size, where $d < c$, and c is divisible by d. We define the following three cases:*

(1) *G is a mesh and H is a torus or a mesh.*

(2) *G and H are both toruses.*

(3) *G is a torus, H is a mesh, and ℓ is even.*

Assume that G is embedded into H using one of the procedures described in the proof of Theorem 3.5.4 with unit dilation cost. Then in each of the three cases above, any scatter type set of parallel neighboring communications in G can be simulated in H by one parallel data movement step. The routing vector for these embedding procedures is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$. □

Theorem 4.5.2 *Assume that a d-dimensional task graph G is embedded into a c-dimensional system graph H using an embedding function with a dilation cost of 2. Then any permutation type set of parallel neighboring communications in G can be simulated in H by two parallel data movement steps. The routing vector for such an embedding function is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

Proof. The system can first perform a bidirectional coordinated parallel data movement along each dimension for one step at the same time, and then perform another bidirectional coordinated parallel data movement along each dimension for one step at the same time. Since for every message involved in the communications, its source and destination satisfy the condition in Lemma 4.2.2 with $\rho = 1$, all of the messages can reach their destinations in the two coordinated parallel data movements. □

Since the embedding functions $g_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{V}}$ have dilation costs of 2 with respect to their appropriate domains and ranges as stated in Theorems 3.3.2 and 3.4.1, we have the following corollary.

Corollary 4.5.3 *Assume that a d-dimensional task graph G is embedded into a c-dimensional system graph H using the embedding function $g_{\mathcal{L}}$ or $\mathcal{G}_{\mathcal{V}}$ with a dilation cost* *of 2. Then any permutation type set of parallel neighboring communications in G can be simulated in H by two parallel data movement steps. The routing vector for these two embedding functions is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* $\Box$

Corollary 4.5.4 *Let G be a d-dimensional square torus with each dimension being of odd length ℓ, and let H be a c-dimensional square mesh of the same size, where $d < c$, and c is divisible by d. Assume that G is embedded into H using the procedure described in the proof of Theorem 3.5.4 with a dilation cost of 2. Then any scatter type set of parallel neighboring communications in G can be simulated in H by two parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* $\Box$

Theorem 4.5.3 *Let d and c be positive integers such that $c < d \leq 2c$. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases. Assume that $\mathcal{M}$ is a general reduction of $\mathcal{L}$ with a reduction factor $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{d-c})$, multiplicant sublist $\mathcal{L}'$, and multiplier sublist $\mathcal{L}''$. Let $\alpha : [d]^+ \to [d]^+$ be a permutation such that $\alpha(\mathcal{L}) = \mathcal{L}' \diamond \mathcal{L}''$. Let $\tilde{\mathcal{S}} = (s_1, s_2, \ldots, s_b) = \mathcal{S}_1 \diamond \mathcal{S}_2 \diamond \cdots \diamond \mathcal{S}_{d-c}$, and let $\mathcal{I} = (\underbrace{1, 1, \ldots, 1}_{c-b})$. Assume that a task graph G of shape $\mathcal{L}' \diamond \mathcal{L}''$ is embedded into a system graph H of shape $[\tilde{\mathcal{S}} \diamond \mathcal{I}] \times \mathcal{L}'$ using an embedding function as specified in the following cases:*

(1) *$\mathcal{F}'_{\mathcal{S}}$, if G is a mesh;*

(2) *$\mathcal{H}'_{\mathcal{S}}$, if G and H are both toruses, and for all $i \in [d-c]^+$, $\mathcal{S}_i$ consists of at least two components such that the first component is an even number;*

(3) *$\mathcal{G}'_{\mathcal{S}}$, if G and H are both toruses, and the condition on the lists in $\mathcal{S}$ stated in (2) is not satisfied;*

(4) *$\mathcal{H}''_{\mathcal{S}}$, if G is a torus, H is a mesh, and for all $i \in [d-c]^+$, $\mathcal{S}_i$ consists of at least two components such that the first component is an even number;*

(5) *$\mathcal{G}''_{\mathcal{S}}$, if G is a torus, H is a mesh, and the condition on the lists in $\mathcal{S}$ stated in (4) is not satisfied.*

Then any permutation type set of parallel neighboring communications in the last $d-c$ dimensions of G can be simulated in H by two parallel data movement steps in cases (3) and (5), and by one parallel data movement step in all of the other cases. Any permutation type set of parallel neighboring communications in the first c dimensions of G can be simulated in H by $2\max\{s_1, s_2, \ldots, s_c\}$ parallel data movement steps in cases (4) and (5), and by $\max\{s_1, s_2, \ldots, s_c\}$ parallel data movement steps in all of the other cases. Any permutation type set of parallel neighboring communications in which at least one occurs in the first c dimensions of G and one occurs in the last $d-c$ dimensions of G can be simulated in H by $1 + \max\{s_1, s_2, \ldots, s_c\}$ parallel data movement steps in cases (1) and (2), by $2 + \max\{s_1, s_2, \ldots, s_c\}$ parallel data movement steps in case (3), by $1 + 2\max\{s_1, s_2, \ldots, s_c\}$ parallel data movement steps in case (4), and by $2(1 + \max\{s_1, s_2, \ldots, s_c\})$ parallel data movement steps in case (5). The routing vector for these embedding functions is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [c]^+$ and $i_k = 2$ for all $k \in \{c+1, c+2, \ldots, d\}$. $\Box$

Proof. We consider only case (1). The proofs for the other cases are similar. By the definition of the function $\mathcal{F}'_{\mathcal{S}}$, we know that for all $i \in [c]^+$, any pair of neighboring logical nodes along the i-th dimension of G is at exactly s_i links apart along the i-th dimension of H after the embedding; for all $i \in \{c+1, c+2, \ldots, d\}$, any pair of neighboring logical nodes along the i-th dimension of G is still neighbors along one of the first c dimensions of H after the embedding.

Hence, by Lemma 4.2.1, any permutation type set of parallel neighboring communications in the last $d-c$ dimensions of G can be simulated by performing a bidirectional coordinated parallel data movement for one step in all of the dimensions of H, and any permutation type set of parallel neighboring communications in the first c dimensions of G can be simulated by performing a bidirectional coordinated parallel data movement for $\max\{s_1, s_2, \ldots, s_c\}$ steps in all of the dimensions of H.

For any permutation type set of parallel neighboring communications in which at least one occurs in the first c dimensions of G and one occurs in the last $d-c$ dimensions of G, we first split the set into two subsets such that all of the neighboring communications in the first subset occur in the first c dimensions of G, and all of the neighboring communications in the second subset occur in the last $d-c$ dimensions of G. Then we simulate the parallel neighboring communications in the two subsets with the two procedures described above sequentially. □

We note that if the data routing in the last theorem is performed under communication mode 3, then in casees (1), (2), and (3), any permutation type set of parallel neighboring communications in the first c dimensions of G can be simulated in H by $2\sum_{k=1}^{c} s_k$ parallel data movement steps, and in cases (4) and (5), any permutation type set of parallel neighboring communications in the first c dimensions of G can be simulated in H by $4\sum_{k=1}^{c} s_k$ parallel data movement steps. The data routing complexities here under communication mode 3 are lower than $2c$ times the corresponding complexities under communication mode 1.

Theorem 4.5.4 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases such that $\mathcal{M}$ is a simple reduction of $\mathcal{L}$. Let $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_c)$ be a reduction factor of $\mathcal{L}$ into $\mathcal{M}$ such that for all $i \in [c]^+$, the elements in the list $\mathcal{V}_i$ are in non-increasing order; let $\tilde{\mathcal{V}} = \mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_c$; let v_i denote the index in $[d]^+$ such that l_{v_i} is the first element in $\mathcal{V}_i$. Let $\pi : [d]^+ \to [d]^+$ be a permutation such that $\pi(\mathcal{L}) = \tilde{\mathcal{V}}$. Assume that a torus or a mesh of shape $\tilde{\mathcal{V}}$ is embedded into a torus or a mesh of shape $\mathcal{M}$ using the function $\mathcal{U}_{\mathcal{V}}$. Then any permutation type set of parallel neighboring communications in G can be simulated in H by $2\max_{1\le i\le c}\{m_i/l_{v_i}\}$ parallel data movement steps if G is a torus and H is a mesh, or by $\max_{1\le i\le c}\{m_i/l_{v_i}\}$ parallel data movement steps otherwise. The routing vector for the embedding function $\mathcal{U}_{\mathcal{V}}$ is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

Proof. By the definition of $\mathcal{U}_{\mathcal{V}}$, every pair of neighboring nodes in G has images in H along a single dimension and within a distance of $2\max_{1\le i\le c}\{m_i/l_{v_i}\}$ if G is a torus and H is a mesh, or along a single dimension and within a distance of $\max_{1\le i\le c}\{m_i/l_{v_i}\}$ otherwise. The theorem follows from Lemma 4.2.1. □

We note that if the data routing in the last theorem is performed under communication mode 3, then any permutation type set of parallel neighboring communications in G can be simulated in H by $2\sum_{i=1}^{c}\{m_i/l_{v_i}\}$ parallel data movement steps if G is a torus and H is a mesh, or by $\sum_{i=1}^{c}\{m_i/l_{v_i}\}$ parallel data movement steps otherwise.

Corollary 4.5.5 *Assume that a d-dimensional hypercube G is embedded into a c-dimensional torus or a c-dimensional mesh H of shape $(m_1, m_2, \ldots, m_c)$ and of the same size using the function $\mathcal{U}_{\mathcal{V}}$. Then any permutation type set of parallel neighboring communications in G can be simulated in H by $\max\{m_1, m_2, \ldots, m_c\}/2$ parallel data movement steps. The routing vector for this embedding is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.*
□

We note that if the data routing in the last corollary is performed under communication mode 3, then any permutation type set of parallel neighboring communications in G can be simulated in H by $\sum_{i=1}^{c} m_i$ parallel data movement steps.

4.5.3 Data Routing for Multi-step Embeddings

We first define an embedding pattern, which is a generalization of our scheme to embed square meshes into either square toruses or square meshes.

Embedding a mesh into a torus or a mesh through sequential elimination

Let d and c be positive integers such that $d > c$, and let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ be a radix-base. We assume that there are two sets of radix-bases $\{\mathcal{L}'_i \,|\, i \in [k+1], |\mathcal{L}'_i| = c\}$ and $\{\mathcal{L}''_i \,|\, i \in [k]\}$ such that

(a) $\mathcal{L} = \mathcal{L}'_0 \diamond \mathcal{L}''_{k-1} \diamond \mathcal{L}''_{k-2} \diamond \cdots \diamond \mathcal{L}''_0$, and

(b) for all $i \in [k]^+$, $\mathcal{L}'_i$ is a general reduction of $\mathcal{L}'_{i-1} \diamond \mathcal{L}''_{i-1}$ with a reduction factor $\mathcal{S}_i = (S_i^1, S_i^2, \ldots, S_i^{|\mathcal{L}''_{i-1}|})$, multiplicant sublist $\mathcal{L}'_{i-1}$, and multiplier sublist $\mathcal{L}''_{i-1}$.

Given a mesh G of shape $\mathcal{L}$ and a torus or a mesh H of shape $\mathcal{M} = \mathcal{L}'_k$, we can sequentially embed G into H in k steps. Let $\{I_i \,|\, i \in [k+1]\}$ be a set of intermediate meshes such that for all $i \in [k+1]$, I_i is of shape

$$\mathcal{L}'_i \diamond \mathcal{L}''_{k-1} \diamond \mathcal{L}''_{k-2} \diamond \cdots \diamond \mathcal{L}''_i.$$

We have $I_0 = G$, $I_k = H$, and for all $i \in [k]^+$,

$$\mathcal{L}'_i \diamond \mathcal{L}''_{k-1} \diamond \mathcal{L}''_{k-2} \diamond \cdots \diamond \mathcal{L}''_i$$

is a general reduction of

$$\mathcal{L}'_{i-1} \diamond \mathcal{L}''_{k-1} \diamond \mathcal{L}''_{k-2} \diamond \cdots \diamond \mathcal{L}''_{i-1}$$

with a reduction factor $\mathcal{S}_i$, a multiplicant sublist $\mathcal{L}'_{i-1} \diamond \mathcal{L}''_{k-1} \diamond \mathcal{L}''_{k-2} \diamond \cdots \diamond \mathcal{L}''_i$, and a multiplier sublist $\mathcal{L}''_{i-1}$. Hence, I_{i-1} can be embedded into I_i using the embedding function $\mathcal{F}'_{\mathcal{S}_i}$. Thus, G can be sequentially embedded into H through k intermediate steps: $G = I_0 \to I_1 \to I_2 \to \cdots \to I_k = H$.

If a graph G is embedded into another graph H using this embedding procedure, we say that G is embedded into H through k steps of *sequential elimination.* The global objective of this embedding procedure is to eliminate the last $d - c$ dimensions of G. We achieve this goal in k intermediate steps. In the i-th step, for all $i \in [k]^+$, I_{i-1} is embedded into I_i by eliminating the last $|\mathcal{L}''_{i-1}|$ dimensions in I_{i-1} simultaneously and distributing these dimensions over the first c dimensions of I_i. In the next two lemmas, we study the relative positions in H of the images of any pair of neighboring nodes in G after G is embedded into H through k steps of sequential elimination. All of the symbols defined for the procedure above will have the same meaning throughout the remainder of this chapter unless stated otherwise.

Lemma 4.5.1 *If a mesh G is embedded into another mesh H through k steps of sequential elimination, then for all $i \in [k]^+$, the mesh I_{i-1} with shape*

$$\mathcal{L}'_{i-1} \diamond \mathcal{L}''_{k-1} \diamond \cdots \diamond \mathcal{L}''_{i-1}$$

is embedded into the mesh I_i with shape

$$\mathcal{L}'_{i} \diamond \mathcal{L}''_{k-1} \diamond \cdots \diamond \mathcal{L}''_{i}$$

with the following properties:

1. *For every two nodes in I_{i-1} differing in only one of the first c positions, their images in I_i still differ in only the same position.*
2. *For every two nodes in I_{i-1} differing in only one of the last $|\mathcal{L}''_{i-1}|$ positions and differing by 1 in that position, their images in I_i differ in only one of the first c positions and differ by 1 in that position.*
3. *For every two nodes in I_{i-1} differing in only one of the remaining positions, their images in I_i still differ in only the same position by the same amount.*

□

Proof. By the definition on page 85, for all $i \in [k]^+$, the reduction factor $\mathcal{S}_i = (S_i^1, S_i^2, \ldots, S_i^{|\mathcal{L}''_{i-1}|})$. Let $p = c + \sum_{j=i-1}^{k-1} |\mathcal{L}''_j|$ be the length of the shape of I_{i-1}, and $q = p - |\mathcal{L}''_{i-1}|$ be the length of the shape of I_i. Let the list $\tilde{\mathcal{S}}_i = (s_1, s_2, \ldots, s_b) = S_i^1 \diamond S_i^2 \diamond \cdots \diamond S_i^{|\mathcal{L}''_{i-1}|}$, for some positive integer $b \le c$. By the definition of function $\mathcal{F}'_{\mathcal{S}_i}$, for every node $X = (x_1, x_2, \ldots, x_b, x_{b+1}, \ldots, x_p)$ in I_{i-1}, we have

$$\mathcal{F}'_{\mathcal{S}_i}(X) = (s_1 x_1 + e_1, s_2 x_2 + e_2, \ldots, s_b x_b + e_b, x_{b+1}, \ldots, x_q),$$

where $(e_1, e_2, \ldots, e_b) = \mathcal{F}_{\mathcal{S}_i}(x_{q+1}, x_{q+2}, \ldots, x_p)$. We can rewrite the image of X as

$$\mathcal{F}'_{\mathcal{S}_i}(X) = (s_1 x_1 + e_1, s_2 x_2 + e_2, \ldots, s_b x_b + e_b, x_{b+1}, \ldots, x_q).$$

Suppose that two nodes in I_{i-1} differ only in the j-th position and differ by ρ in that position, where $j \in [c]^+$. If $j \in [b]^+$, then by the definition of $\mathcal{F}'_{\mathcal{S}_i}$, the images are of the forms $(y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_q)$ and $(y_1, y_2, \ldots, y_{j-1}, y_j \pm \rho s_j, y_{j+1}, \ldots, y_q)$. If $j \in \{b+1, b+2, \ldots, c\}$, then the images are of the forms $(y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_q)$ and $(y_1, y_2, \ldots, y_{j-1}, y_j \pm \rho, y_{j+1}, \ldots, y_q)$. Thus, the lemma is true.

Suppose that two nodes in I_{i-1} differ only in one of the last $|\mathcal{L}''_{i-1}|$ positions and differ by 1 in that position. Then by the definition of $\mathcal{F}'_{\mathcal{S}_i}$ and the fact that $\mathcal{F}_{\mathcal{S}_i}$ has unit dilation cost because I_{i-1} is a mesh, the images of the two nodes must differ in only one of the first b positions and differ by 1 in that position. Since $b \leq c$, the lemma is true.

Suppose that two nodes in I_{i-1} differ only in the j-th position and differ by ρ in that position, and that position is any of the remaining positions. Then by the definition of $\mathcal{F}'_{\mathcal{S}_i}$, the images of the two nodes will be of the forms $(y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_q)$ and $(y_1, y_2, \ldots, y_{j-1}, y_j \pm \rho, y_{j+1}, \ldots, y_q)$ and thus the lemma is again true. □

Lemma 4.5.2 *Assume that a mesh G is embedded into another mesh H through k steps of sequential elimination. Then the images of any two adjacent nodes in G differ in exactly one position.* □

Proof. If $k = 1$, the lemma follows directly from Lemma 4.5.1. Assume that $k > 1$. Let u and v be an arbitrary pair of adjacent nodes in G. We use induction on the embedding step i to prove the following stronger assertion: for all $i \in [k]^+$, after the i-th embedding step,

(a) if u and v are neighbors along one of the first c dimensions of G or along one of the last $|\mathcal{L}''_{i-1} \diamond \mathcal{L}''_{i-2} \diamond \cdots \diamond \mathcal{L}''_0|$ dimensions of G, then their images in I_i differ in only one of the first c dimensions of I_i, and

(b) if u and v are neighbors along one of the remaining dimensions of G, then their images in I_i remain neighbors along the same dimension of I_i.

Induction basis: For $i = 1$, by Lemma 4.5.1, the assertion is true.
Induction hypothesis: Assume that for any $1 \leq i \leq p < k$, the assertion is true.
Induction step: Now we consider the case after the $(p+1)$-th embedding step. Let u and v be two arbitrary neighbors in G, and let u' and v' be their respective images in I_p. There are three cases.

Case 1. u and v are neighbors along either one of the first c dimensions of G or along one of the last $|\mathcal{L}''_{p-1} \diamond \mathcal{L}''_{p-2} \diamond \cdots \diamond \mathcal{L}''_0|$ dimensions of G. By our induction hypothesis, u' and v' differ in only one of the first c dimensions of I_p. By Lemma 4.5.1, the images of u' and v' in I_{p+1} differ in only one of the first c dimensions of I_{p+1}.

Case 2. u and v are neighbors along one of the dimensions of G that has index in $\{t+1, t+2, \ldots, t+|\mathcal{L}''_p|\}$, where $t = c + \sum_{j=p+1}^{k-1} |\mathcal{L}''_j|$. By our induction hypothesis, u' and v' remain neighbors along the same dimension of I_p. By Lemma 4.5.1, the images of u' and v' in I_{p+1} differ in only one of the first c dimensions of I_{p+1}.

Case 3. u and v are neighbors along one of the remaining dimensions of G. By our induction hypothesis, u' and v' remain neighbors along the same dimension of I_p. Again by Lemma 4.5.1, the images of u' and v' in I_{p+1} remain neighbors along the same dimension of I_{p+1}.

This completes the proof of the lemma. □

Theorem 4.5.5 *Assume that a d-dimensional mesh G is embedded into a c-dimensional torus or a c-dimensional mesh H through sequential elimination with dilation cost ρ. Then any permutation type set of parallel neighboring communications in G can be simulated in H by ρ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

Proof. By Lemma 4.5.2, the images of any pair of neighboring nodes in G will lie along a single dimension in H. By the definition of dilation cost, we know that the two images must be within ρ steps along this dimension. By Lemma 4.2.1, we can simultaneously perform along each dimension of H a bidirectional coordinated parallel data movement for ρ steps to send all of the messages to their destinations without link conflicts. $\square$

Corollary 4.5.6 *Let G be a d-dimensional square mesh for which each dimension has length ℓ, and let H be a c-dimensional square torus or a c-dimensional square mesh of the same size, where $d > c$. If G is embedded into H with the procedure described in the proof of Theorem 3.5.3, then any permutation type set of parallel neighboring communications in G can be simulated in H by $\ell^{(d-c)/c}$ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* $\square$

Proof. The corollary follows from Theorem 4.5.5 by showing that the procedure we use to embed G into H is a special case of the sequential elimination for embedding a mesh into a torus or a mesh and has dilation cost $\ell^{(d-c)/c}$. $\square$

Corollary 4.5.7 *Let G be a d-dimensional square torus for which each dimension has even length ℓ, and let H be another c-dimensional square torus of the same size, where $d > c$. Assume that G is embedded into H with the procedure described in the proof of Theorem 3.5.3. Then any permutation type set of parallel neighboring communications in G can be simulated in H by $\ell^{(d-c)/c}$ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* $\square$

Proof. The only difference between this embedding process and the embedding of a mesh into a torus or a mesh through sequential elimination is that in each step, the former uses the function $\mathcal{H}'_{\mathcal{S}_i}$ whereas the latter uses the function $\mathcal{F}'_{\mathcal{S}_i}$. All of the properties we proved for the embedding of a mesh into a torus or a mesh through sequential elimination hold, as long as we use a function with unit dilation cost in each embedding step. We also know that this embedding procedure has dilation cost $\ell^{(d-c)/c}$. $\square$

Corollary 4.5.8 *Let G be a d-dimensional square torus for which each dimension has even length ℓ, and let H be a c-dimensional square mesh of the same size, where $d > c$. Assume that G is embedded into H with the procedure described in the proof of Theorem 3.5.3. Then any permutation type set of parallel neighboring communications in G can be simulated in H by $2\ell^{(d-c)/c}$ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* $\square$

Proof. By the procedure described in the proof of Theorem 3.5.3, the torus G is first embedded into the torus I_{u-v} through $u - v$ steps of sequential elimination, where I_{u-v} has the same shape as H, say $\mathcal{M}$, and then the torus I_{u-v} is embedded into the mesh H by $\mathcal{T}_{\mathcal{M}}$. By Corollary 4.5.7, any message involved in the communications has source and destination in I_{u-v} along a single dimension and at a distance less than or equal to $\ell^{(d-c)/c}$. Since for all $i \in [c]^+$, $\mathcal{T}_{\mathcal{M}}$ maps every pair of neighboring nodes in I_{u-v} along the i-th dimension to nodes in H along the same dimension and at a distance less than or equal to 2, any message involved in the communications has source and destination in H along a single dimension and at a distance less than or equal to $2\ell^{(d-c)/c}$. By Lemma 4.2.1, we can simultaneously perform along each dimension of H a bidirectional coordinated parallel data movement for $2\ell^{(d-c)/c}$ steps to send all the messages to their destinations without link conflicts. $\Box$

Corollary 4.5.9 *Let G be either a d-dimensional square mesh for which each dimension has length ℓ, or a d-dimensional square torus for which each dimension has even length ℓ, and let H be a c-dimensional square torus or a c-dimensional square mesh of the same size, where $d < c$, c is not divisible by d, and a is the greatest common divisor of c and d. Assume that G is embedded into H with the procedure described in the proof of Theorem 3.5.5. Then any permutation type set of parallel neighboring communications in G can be simulated in H by $\ell^{(d-a)/c}$ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* $\Box$

Proof. The embedding is achieved through two steps. In the first step, G is embedded into a square mesh G' of higher dimension with unit dilation cost. In the second step, G' is embedded into H through sequential elimination. Since the neighborship in G is maintained in G', any permutation type set of parallel neighboring communications in G corresponds to a permutation type set of parallel neighboring communications in G'. The corollary now follows from Corollary 4.5.6. $\Box$

Theorem 4.5.6 *Let G be a d-dimensional square torus for which each dimension has odd length ℓ, and let H be a c-dimensional square torus or a c-dimensional square mesh of the same size, where $d > c$. Let a be the greatest common denominator of d and c. If G is embedded into H with the procedure described in the proof of Theorem 3.5.3 for the case in which H is a mesh, then any permutation type set of parallel neighboring communications in G can be simulated in H by $4\ell^{(d-c)/c}$ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [c]^+$, and $i_k = \lceil (k-c)/a \rceil + 1$ for all $c < k \leq d$.* $\Box$

Proof. Let $u = d/a$, and $v = c/a$. By the procedure described in the proof of Theorem 3.5.3, the torus G, which has shape $\mathcal{L} = (\ell, \ell, \ldots, \ell)$, is first embedded into the mesh I_0 of the same shape using the embedding function $\mathcal{T}_{\mathcal{L}}$ with a dilation cost of 2. I_0 is then embedded into H through $u - v$ steps of sequential elimination. For each $i \in [u-v]^+$, the i-th step of this elimination procedure uses the embedding function $\mathcal{F}_{\mathcal{S}_i}$ to embed the last a dimensions of I_{i-1} into the first c dimensions of I_i.

For all $i \in [d]^+$, every two neighboring nodes in G along the i-th dimension are mapped by $\mathcal{T}_{\mathcal{L}}$ to two nodes in G' along the same dimension and at a distance less than or equal to 2. Furthermore, every two neighboring nodes in G' along any of the first c dimensions are mapped by the sequential elimination to nodes in H along a single dimension and at a distance of $\ell^{(u-v)/v}$. Hence, every two neighboring nodes in G along any of the first c dimensions are mapped by the procedure described in the proof of Theorem 3.5.3 to nodes in H that satisfy the conditions in Lemma 4.2.1 if we interpret them as the source and destination of a message, with $\rho = 2\ell^{(u-v)/v}$. Therefore, by Lemma 4.2.1, we conclude that any permutation type set of parallel neighboring communications in G along the first c dimensions can be simulated in H by simultaneously performing along each dimension of H a bidirectional coordinated parallel data movement for $2\ell^{(u-v)/v}$ steps.

For all $i \in [d]^+$, every two neighboring nodes in G along the i-th dimension are mapped by $\mathcal{T}_{\mathcal{L}}$ to nodes in G' along the same dimension and at a distance less than or equal to 2. Furthermore, for all $i \in [u-v]^+$, every two neighboring nodes in G' along the k-th dimension, for $k \in \{d-ia+1, d-ia+2, \ldots, d-(i-1)a\}$, are mapped by the sequential elimination to nodes in H along a single dimension and at a distance of $\ell^{(u-v-i)/v}$. Therefore, every two neighboring nodes in G along the k-th dimension are mapped by the procedure described in the proof of Theorem 3.5.3 to nodes in H that satisfy the conditions in Lemma 4.2.2 if we interpret them as the source and destination of a message, with $\rho = \ell^{(u-v-i)/v}$. Hence, by Lemma 4.2.2, we conclude that for all $i \in [u-v]^+$, any permutation type set of parallel neighboring communications in G along the k-th dimension, for $k \in \{d-ia+1, d-ia+2, \ldots, d-(i-1)a\}$, can be simulated in H by first simultaneously performing along each dimension of H a bidirectional coordinated parallel data movement for $\ell^{(u-v-i)/v}$ steps, and then simultaneously performing along each dimension of H a bidirectional coordinated parallel data movement for $\ell^{(u-v-i)/v}$ steps. Similarly, any permutation type set of parallel neighboring communications in the first c dimensions of G can be simulated in H by first simultaneously performing along each dimension of H a bidirectional coordinated parallel data movement for $\ell^{(u-v)/v}$ steps, and then simultaneously performing along each dimension of H a bidirectional coordinated parallel data movement for $\ell^{(u-v)/v}$ steps.

Given any permutation type set of parallel neighboring communications in G, we can partition it into $u-v+1$ subsets each of which contains neighboring communications either along only the first c dimensions of G, or along only those dimensions of G that have indices in $\{d-ia+1, d-ia+2, \ldots, d-(i-1)a\}$ for every fixed $i \in [u-v]^+$. Then we sequentially simulate each of these $u-v+1$ subsets of parallel neighboring communications. The global data routing complexity is thus

$$2\sum_{i=0}^{u-v} \ell^{(u-v-i)/v} = 2\ell^{(d-c)/c} \sum_{i=0}^{u-v} \ell^{-i/v}.$$

Since $\ell > 2$ and $v \geq 1$, we have $\ell^{-1/v} < 1/2$, and

$$\begin{aligned}\sum_{i=0}^{u-v} \ell^{-i/v} &< \sum_{i=0}^{\infty} (\ell^{-1/v})^i \\ &= \frac{1}{1-\ell^{-1/v}} \\ &< \frac{1}{1-1/2} \\ &= 2.\end{aligned}$$

Therefore, we conclude that $2\ell^{(d-c)/c} \sum_{i=0}^{u-v} \ell^{-i/v} < 4\ell^{(d-c)/c}$. □

For each of the embedding functions or procedures discussed in this section up to now, we provided a data routing vector for the task graph G. The following is the only exception in this thesis. In this case we find it most natural to define the data routing vector for an intermediate graph G'. To carry out this data routing strategy, each processor has to translate communications in the task graph G into communications in G' before invoking the general shortest-path data routing scheme. In this special case, the translation is achieved by a simple function $s(x)$.

Theorem 4.5.7 *Let G be a d-dimensional square torus for which each dimension has odd length ℓ, and let H be a c-dimensional square torus or a c-dimensional square mesh of the same size, where $d < c$, c is not divisible by d, and a is the greatest common divisor of c and d. Assume that G is embedded into H with the procedure described in the proof of Theorem 3.5.5 for the case in which H is a mesh. Then any permutation type set of parallel neighboring communications in G can be simulated in H by $4\ell^{(d-a)/c}$ parallel data movement steps. The routing vector for this embedding procedure is $(i_1, i_2, \ldots, i_{dv})$, where $i_k = 1$ for all $k \in [c]^+$, and $i_k = \lceil (k-c)/a \rceil + 1$ for all $c < k \leq dv$.* $\square$

Proof. Let $u = d/a$, and $v = c/a$. The d-dimensional square torus G is first embedded into another vd-dimensional torus G' of shape $(\ell^{1/v}, \ell^{1/v}, \ldots, \ell^{1/v})$ by the embedding function $\mathcal{H}_{\mathcal{V}}$ with unit dilation cost, where $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$ and for all $i \in [d]^+$, $\mathcal{V}_i = (\ell^{1/v}, \ell^{1/v}, \ldots, \ell^{1/v})$ and $|\mathcal{V}_i| = v$. The square torus G' is then embedded into the square torus or the square mesh H of the same size and lower dimension using the procedure described in the proof of Theorem 3.5.3. By Theorem 4.5.6, we know that our routing vector can be used to simulate in H any permutation type set of parallel neighboring communications in G' with a data routing complexity $2\ell^{(d-a)/c} \sum_{i=0}^{u-1} \ell^{-i/v} < 4\ell^{(d-a)/c}$.

Since the routing vector in this theorem works only for graph G', we need an algorithm to decide for each neighboring communication in G along which dimension in G' the communication will be after the first embedding step.

Let $\mathcal{L} = (\ell^{1/v}, \ell^{1/v}, \ldots, \ell^{1/v})$ and $|\mathcal{L}| = v$. We define the function $s : [\ell - 1] \to [v]^+$ as follows:

$$s(x) = k \text{ where } h_{\mathcal{L}}(x) \text{ and } h_{\mathcal{L}}(x+1) \text{ differ in the } k\text{-th position},$$

for all $x \in [\ell - 1]$.

If a neighboring communication in G occurs between nodes with addresses $(x_1, x_2, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_d)$ and $(x_1, x_2, \ldots, x_{j-1}, x_j + 1, x_{j+1}, \ldots, x_d)$, then we can conclude that this communication will occur in the p-th dimension of G' after the first embedding step, where

$$p = (j-1)v + s(x_j).$$

$\square$

4.6 Data Routing Complexity versus Dilation Cost

This section summarizes our data routing complexities for simulating in a system graph H the parallel neighboring communications in a task graph G after G is embedded into H by one of the embedding functions or procedures defined in Chapter 3. The complexity results fall into five categories: basic embeddings, embeddings for increasing dimension, embeddings for lowering dimension, embeddings among square graphs for lowering dimension, and embeddings among square graphs for increasing dimension. Within each category, we classify the complexity results according to the domains and ranges of the embeddings. In each case, we list the graph type of G, the graph type of H, the embedding function we use, the type of neighboring communications in G, and the corresponding data routing complexity. In each case, we assume that the domain and the range satisfy the conditions specified in Chapter 3 when the embedding function was defined, and that ρ is the dilation cost we derived in Chapter 3 for the embedding function under the corresponding domain and range assumptions. We express the corresponding data routing complexity as a function of ρ.

Basic embeddings.

1. If G is a line, H is a torus or a mesh, and we use the function $f_{\mathcal{L}}$, then any scatter type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
2. If G is a ring of odd size, H is a mesh, and we use the function $g_{\mathcal{L}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
3. If G is a ring of even size, H is a mesh, and we use the function $h_{\mathcal{L}}$, then any scatter type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
4. If G is a ring, H is a torus, and we use the function $h_{\mathcal{L}}$, then any scatter type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

Embeddings for increasing dimension

1. If G is a mesh, H is a torus or a mesh, and we use the function $\mathcal{F}_{\mathcal{V}}$, then any scatter type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
2. If G is a torus, H is a torus, and we use the function $\mathcal{H}_{\mathcal{V}}$, then any scatter type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
3. If G is a torus, H is a mesh, and we use the function $\mathcal{H}_{\mathcal{V}}$, where each list in $\mathcal{V}$ consists of at least two components such that the first component is an even number, then any scatter type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
4. If G is a torus, H is a mesh, and we use the function $\mathcal{G}_{\mathcal{V}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

Embeddings for lowering dimension

1. If G is a torus or a mesh, H is a torus or a mesh, and we use the function $\mathcal{U}_{\mathcal{V}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.
2. If G is a hypercube, H is a torus or a mesh, and we use the function $\mathcal{U}_{\mathcal{V}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

3. If G is a mesh, H is a torus or a mesh, and we use the function $\mathcal{F}'_{\mathcal{S}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity $\rho + 1$.

4. If G is a torus, H is a torus, and we use the function $\mathcal{H}'_{\mathcal{S}}$, where each list in $\mathcal{S}$ consists of at least two components such that the first component is an even number, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity $\rho + 1$.

5. If G is a torus, H is a torus, and we use the function $\mathcal{G}'_{\mathcal{S}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity $\rho + 2$.

6. If G is a torus, H is a mesh, and we use the function $\mathcal{H}''_{\mathcal{S}}$, where each list in $\mathcal{S}$ consists of at least two components such that the first component is an even number, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity $\rho + 1$.

7. If G is a torus, H is a mesh, and we use the function $\mathcal{G}''_{\mathcal{S}}$, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity $\rho + 2$.

Embeddings among square graphs for lowering dimension

1. If G is a mesh, H is a torus or a mesh, and we use the embedding procedure from the proof of Theorem 3.5.3, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

2. If G is a torus of even size, H is a torus, and we use the embedding procedure from the proof of Theorem 3.5.3, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

3. If G is a torus of even size, H is a mesh, and we use the embedding procedure from the proof of Theorem 3.5.3, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

4. If G is a torus of odd size, H is a mesh, and we use the embedding procedure from the proof of Theorem 3.5.3, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity 2ρ.

5. If G is a torus of odd size, H is a torus, and we use the embedding procedure from the proof of Theorem 3.5.3 for the case in which H is a mesh, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity 4ρ.

Embeddings among square graphs for increasing dimension

1. If G is a d-dimensional torus or a c-dimensional mesh, H is a c-dimensional torus or a c-dimensional mesh, where c is divisible by d, and we use the embedding procedure from the proof of Theorem 3.5.4, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

2. If G is a d-dimensional mesh, H is a c-dimensional torus or a c-dimensional mesh, where c is not divisible by d, and we use the embedding procedure from the proof of Theorem 3.5.5, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

3. If G is a d-dimensional torus of even size, H is a c-dimensional torus or a c-dimensional mesh, where c is not divisible by d, and we use the embedding procedure from the proof of Theorem 3.5.5, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity ρ.

4. If G is a d-dimensional torus of odd size, H is a c-dimensional mesh, where c is not divisible by d, and we use the embedding procedure from the proof of Theorem 3.5.5, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity 2ρ.

5. If G is a d-dimensional torus of odd size, H is a c-dimensional torus, where c is not divisible by d, and we use the embedding procedure from the proof of Theorem 3.5.5 for the case in which H is a mesh, then any permutation type set of parallel neighboring communications in G can be simulated in H with data routing complexity 4ρ.

4.7 Conclusion

In this chapter, we use graph embedding technique to map system topology independent parallel programs onto parallel processing systems. We identify three tasks in implementing a program mapping: (1) logical address identification, by which each processor identifies the process in the parallel program to be mapped into it; (2) code loading, by which codes for different processes get loaded into the corresponding processors; and (3) translation table generation, by which each processor can transform inter-process communication into inter-processor communication automatically at execution time.

For logical address identification, we propose a parallel solution based on parallel evaluation of the inverse of the mapping function by all of the processors. Since all of our mapping functions have time complexities either proportional to a constant or proportional to $(d-c)$, where d is the dimension of the task graph and c is the dimension of the system graph, our logical address identification has low time complexity and can be performed at execution time.

For code loading, we propose a parallel approach based on logical address identification. The time required for the program loading process is proportional to the number of different code types used in the parallel program, but not to the size of the task graph. This approach works especially well for large parallel programs in which only a limited number of code types are used. In this approach, we assume that broadcast is the only means for the host to send messages to the physical nodes. Since a broadcast network is available or simulated in all SIMD, MSIMD, and MIMD systems, this approach is applicable to a wide range of parallel processing systems.

For translation table generation, we propose two methods. The first method is based on the parallel computation of the embedding function and can be applied to all of our embedding functions. The second method is based on parallel data movements of the ordered pairs "(logical address, physical address)" computed in the logical address identification stage and is used only if the embedding has unit dilation cost. The time complexity of the first method is proportional to the product of d and the complexity of the embedding function, where d is the dimension of the task graph. The time complexity of the second method under the bidirectional link assumption is a constant.

In this chapter, we also design for each of our graph embedding functions a data routing strategy to achieve conflict-free simulation in the system graph of either any scatter type set of parallel neighboring communications in the task graph if the embedding has unit dilation cost, or any permutation type set of parallel neighboring communications in the task graph otherwise. In most cases, these data routing strategies can simply take the form of data routing vectors. We propose a simple data routing scheme, the *shortest-path data routing scheme,* that can automatically carry out our data routing strategies at execution time. This scheme has low overhead, and can be easily implemented either by software or by hardware. This scheme uses our data routing strategies and local information to ensure that all of the messages can move along the shortest paths to their destinations without link conflicts.

We analyze the correponding data routing complexities for each of our graph embeddings. Let ρ denote the dilation cost of a graph embedding. The data routing complexity for each of our graph embeddings is ρ (16 cases), $\rho + 1$ (3 cases), $\rho + 2$ (2 cases), 2ρ (2 cases), or 4ρ (2 cases).

Chapter 5
Task Graph Contraction

5.1 Introduction

In the preceding two chapters, we discussed graph embeddings for which the guest graph and the host graph have the same number of nodes, and applied these embedding results to mapping parallel programs onto parallel processing systems in which each processor has only one process mapped into it. If we want to execute a parallel program with more processes than the processors available in a parallel processing system, we have to study the corresponding many-to-one graph mapping problem and many-to-one program mapping problem.

In this chapter, we first generalize the optimization measure for graph embedding to build our many-to-one graph mapping model of the optimal program mapping problem. In addition to minimizing dilation cost, we also need to balance the number of guest nodes mapped into each host node. In the corresponding program mapping problem, since all of the processes mapped into the same processor have to be executed sequentially, an even distribution of the processes over the processors minimizes the computation time.

We show that for toruses and meshes, we can obtain many-to-one graph mapping by first contracting the guest graph into some intermediate graph of the same size as the host graph, and then using our embedding schemes to embed the intermediate graph into the host graph. Although this decomposition of the many-to-one graph mapping into two steps will generally reduce our chances of global optimization, we show that in our special problem domain, we can generalize each of our embedding schemes into the many-to-one version by performing an appropriate contraction step before the embedding and still achieve optimal or good many-to-one mapping results.

At the end of this chapter, we use the many-to-one graph mapping functions to generalize the program mapping approach described in Chapter 4 and thereby achieve many-to-one program mapping.

To simplify our graph mapping model, we assume that in the task graph all of the links carry the same communication load and all of the processes require the same computation time.

5.2 Generalized Optimization Measures

In the graph embedding problem, we use dilation cost as the optimization measure. In the corresponding program mapping problem, dilation cost is the number of system cycles required by a single process in the task graph to send a message to one of its neighboring processes in the worst case.

In the many-to-one graph mapping problem, while dilation cost is still important, we have to take into consideration the evenness of the distribution of the nodes in the guest graph over the nodes in the host graph. The evenness of the node distribution is an abstraction of the following issue in the corresponding program mapping problem. If more than one process is mapped into a single processor, these processes have to proceed sequentially. The computation time of each processor is proportional to the number of processes mapped into the processor. Since the computation time of the entire system is determined by the maximum computation time needed by each processor, we should try to balance the number of processes mapped into each processor to minimize system computation time and improve processor utilization.

In this chapter, we define the following two optimization measures for our many-to-one graph mappings. Let $\mathcal{E}$ be any many-to-one graph mapping.

1. Dilation cost $D(\mathcal{E})$: the maximum distance in the host graph between images of any pair of neighboring nodes in the guest graph.

2. Node evenness $E(\mathcal{E})$: using $\eta(v)$ to denote the number of nodes in the guest graph mapped into node v in the host graph, node evenness is defined to be $\max\{\eta(v)/\eta(v') | v \text{ and } v' \text{ are nodes in the host graph}\}$.

By the definition of node evenness, we conclude that for a mapping of finite node evenness, any node in the host graph has at least one node in the guest graph mapped into it. The best value for node evenness is 1 if the ratio of the size of the guest graph to the size of the host graph is an integer, and 2 otherwise. Since any node in the host graph is the image of at least one node in the guest graph, unless the guest graph is not connected or the host graph has only one node, the best value for the dilation cost is 1.

Therefore, for any many-to-one mapping $\mathcal{E}$ such that $D(\mathcal{E}) = 1$, and $E(\mathcal{E}) = 1$ if the ratio of the size of the guest graph to the size of the host graph is an integer and $E(\mathcal{E}) = 2$ otherwise, we can claim that the many-to-one mapping $\mathcal{E}$ is optimal with respect to our optimization measures.

5.3 Contraction before Embedding

In this section, we first present a general scheme to generalize our embedding functions into their many-to-one versions. A many-to-one mapping constructed by this scheme has the same dilation cost as its corresponding embedding function. We also provide a set of special cases in which we can achieve optimal many-to-one mappings.

A general many-to-one mapping scheme

Let G be a torus or a mesh of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and H be a torus or a mesh of shape $\mathcal{M} = (m_1, m_2, \ldots, m_c)$. Assume that we can find an intermediate graph G' such that G' is of the same type as G; G' has shape $\mathcal{L}' = (l'_1, l'_2, \ldots, l'_d)$; for some $\kappa > 0$, $l_i = \kappa l'_i$ for all $i \in [d]^+$; $\prod_{i=1}^{d} l'_i = \prod_{i=1}^{c} m_i$; and there exists an embedding function $\mathcal{E} : \Omega_{\mathcal{L}'} \to \Omega_{\mathcal{M}}$ to embed G' into H with dilation cost ρ. We first define the contraction function $\mu_{\kappa,d} : \Omega_{(l_1,l_2,\ldots,l_d)} \to \Omega_{(l'_1,l'_2,\ldots,l'_d)}$ as follows:

$$\mu_{\kappa,d}((x_1, x_2, \ldots, x_d)) = \left(\left\lfloor \frac{x_1}{\kappa} \right\rfloor, \left\lfloor \frac{x_2}{\kappa} \right\rfloor, \ldots, \left\lfloor \frac{x_d}{\kappa} \right\rfloor \right)$$

for all $(x_1, x_2, \ldots, x_d) \in \Omega_{(l_1,l_2,\ldots,l_d)}$. If $d = 1$, as with the case in Chapter 3, we also write $\mu_{\kappa,1} : \Omega_{(l_1)} \to \Omega_{(l'_1)}$ as $\mu_\kappa : [l_1] \to [l'_1]$.

We can use the contraction function $\mu_{\kappa,d}$ to map G into G' with unit dilation cost. We can map G into H with the composite many-to-one mapping function $\mathcal{E} \circ \mu_{\kappa,d} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$. We map each node $(x_1, x_2, \ldots, x_d) \in \Omega_{\mathcal{L}}$ in G into the node

$$\mathcal{E}(\mu_{\kappa,d}((x_1, x_2, \ldots, x_d)))$$

in H.

This many-to-one mapping has a dilation cost of ρ. Each node in H has exactly κ^d nodes in G mapped into it.

If the embedding function $\mathcal{E}$ has unit dilation cost, then this general mapping scheme is optimal. Therefore, if $\mathcal{E}$ is any one of the functions in $\{f_{\mathcal{L}}, r_{\mathcal{L}}, h_{\mathcal{L}}, \mathcal{F}_{\mathcal{V}}, \mathcal{H}_{\mathcal{V}}\}$, and the types and shapes of G' and H satisfy the conditions we specified in Chapter 3 for such a function to have unit dilation cost, then this general scheme provides an optimal many-to-one mapping.

The many-to-one mapping derived from this scheme is not optimal in general. In the remainder of this section, we provide some special conditions on the domains and ranges of graph mappings. If these conditions are satisfied, we show how to design optimal many-to-one mappings.

Mapping a ring of even size into a mesh

Given a ring G of size $\mathtt{g}$ and a mesh H of shape $\mathcal{L}$ and of size $\mathtt{h}$, where $\mathtt{g} = 2\kappa\mathtt{h}$ and κ is any positive integer. We can reduce the dilation cost from 2 to 1 by avoiding the use of the function $g_{\mathcal{L}}$. For any positive even integer n, we define the fold function $\nu_n : [n] \to [n/2]$ as follows:

$$\nu_n(x) = \begin{cases} x, & \text{if } 0 \le x < n/2, \\ n - 1 - x, & \text{otherwise} \end{cases}$$

for all $x \in [n]$.

We can use the fold function $\nu_{\mathtt{g}}$ to map the ring G of size $\mathtt{g}$ into a line G' of size $\mathtt{g}/2$, and then use the contraction function μ_κ to map the line G' into another line G'' of size $\mathtt{h}$. Such a contraction of G into G'' has unit dilation cost. We can map G into H by the composite mapping function $f_{\mathcal{L}} \circ \mu_\kappa \circ \nu_{\mathtt{g}} : [\mathtt{g}] \to \Omega_{\mathcal{L}}$. We map each node $x \in [\mathtt{g}]$ in G into node

$$f_{\mathcal{L}}(\mu_\kappa(\nu_{\mathtt{g}}(x)))$$

in H.

The many-to-one mapping function $f_{\mathcal{L}} \circ \mu_\kappa \circ \nu_{\mathtt{g}}$ has unit dilation cost. Each node in H has exactly 2κ nodes in G mapped into it. Thus, this mapping scheme is optimal.

Mapping a torus into a mesh of higher dimension

Let G be a torus of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and let H be a mesh of shape $(m_1, m_2, \ldots, m_c)$ such that $(m_1, m_2, \ldots, m_c)$ is an expansion of $(l'_1, l'_2, \ldots, l'_d)$ with an expansion factor $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d)$; $l_i = 2\kappa l'_i$ for all $i \in [d]^+$; and κ is any positive integer. Let $\pi : [c]^+ \to [c]^+$ be a permutation such that $\pi(\mathcal{V}_1 \diamond \mathcal{V}_2 \cdots \diamond \mathcal{V}_d) = (m_1, m_2, \ldots, m_c)$. If there is at least one integer $i \in [d]^+$ such that l'_i is an odd number, we can reduce the dilation cost from 2 to 1 by avoiding the use of the function $\mathcal{G}_\mathcal{V}$.

We define the fold function $\bar{\nu}_\mathcal{L} : \Omega_{(l_1,l_2,\ldots,l_d)} \to \Omega_{(l_1/2,l_2/2,\ldots,l_d/2)}$ as follows:

$$\bar{\nu}_\mathcal{L}((x_1, x_2, \ldots, x_d)) = (\nu_{l_1}(x_1), \nu_{l_2}(x_2), \ldots, \nu_{l_d}(x_d))$$

for all $(x_1, x_2, \ldots, x_d) \in \Omega_{(l_1,l_2,\ldots,l_d)}$. We can first use the fold function $\bar{\nu}_\mathcal{L}$ to map the torus G of shape $(l_1, l_2, \ldots, l_d)$ into a mesh G' of shape $(l_1/2, l_2/2, \ldots, l_d/2)$ with unit dilation cost. Then we can use the contraction function $\mu_{\kappa,d}$ to map the mesh G' of shape $(l_1/2, l_2/2, \ldots, l_d/2)$ into another mesh G'' of shape $(l'_1, l'_2, \ldots, l'_d)$ with unit dilation cost. We can map G into H by the composite mapping function $\pi \circ \mathcal{F}_\mathcal{V} \circ \mu_{\kappa,d} \circ \bar{\nu}_\mathcal{L} : \Omega_{(l_1,l_2,\ldots,l_d)} \to \Omega_{(m_1,m_2,\ldots,m_c)}$. We map each node $(x_1, x_2, \ldots, x_d) \in \Omega_{(l_1,l_2,\ldots,l_d)}$ in G into node

$$\pi(\mathcal{F}_\mathcal{V}(\mu_{\kappa,d}(\bar{\nu}_\mathcal{L}((x_1, x_2, \ldots, x_d)))))$$

in H.

Since all of the four component functions have unit dilation cost, the many-to-one mapping function $\pi \circ \mathcal{F}_\mathcal{V} \circ \mu_{\kappa,d} \circ \bar{\nu}_\mathcal{L}$ has unit dilation cost. Each node in H has exactly $(2\kappa)^d$ nodes in G mapped into it. Thus, this mapping scheme is optimal.

Mapping a torus or a mesh into another one with lower dimension

Let d and c be positive integers such that $d > c$, and let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{L}' = (l_{\pi(1)}, l_{\pi(2)}, \ldots, l_{\pi(c)})$ be radix-bases where $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. We first define the contraction function $\xi_{\pi,d,c} : \Omega_\mathcal{L} \to \Omega_{\mathcal{L}'}$ as follows:

$$\xi_{\pi,d,c}((x_1, x_2, \ldots, x_d)) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(c)})$$

for all $(x_1, x_2, \ldots, x_d) \in \Omega_\mathcal{L}$.

Assume that G is a mesh of shape $\mathcal{L}$, and H is another mesh of shape $\mathcal{L}'$. If G is mapped into H using the contraction function $\xi_{\pi,d,c}$, then each supernode of shape $(l_{\pi(c+1)}, l_{\pi(c+2)}, \ldots, l_{\pi(d)})$ in G is mapped into a single node in H. This mapping has unit dilation cost. Since each node in H is the image of the same number of nodes in G, this many-to-one mapping is optimal.

Assume that G is a mesh of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and H is a torus or a mesh of shape $\mathcal{M} = (m_1, m_2, \ldots, m_c)$, where $l_{\pi(i)} = m_i\kappa$ for all $i \in [c]^+$; κ is any positive integer; and $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. We can first use the contraction function $\xi_{\pi,d,c}$ to map the mesh G into another mesh G' of shape $(l_{\pi(1)}, l_{\pi(2)}, \ldots, l_{\pi(c)})$, and then use the contraction function $\mu_{\kappa,c}$ to map the mesh G' into the torus or the mesh H. The composite mapping function is $\mu_{\kappa,c} \circ \xi_{\pi,d,c} : \Omega_\mathcal{L} \to \Omega_\mathcal{M}$. Since $\mu_{\kappa,c}$ and $\xi_{\pi,d,c}$ both have unit dilation cost, the many-to-one mapping function $\mu_{\kappa,c} \circ \xi_{\pi,d,c}$ has unit dilation cost. Each node in H has exactly $\kappa^c \prod_{i=c+1}^{d} l_{\pi(i)}$ nodes in G mapped into it. Thus, this many-to-one mapping is optimal.

Assume that G is a torus of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and H is another torus of shape $\mathcal{M} = (m_1, m_2, \ldots, m_c)$, where $l_{\pi(i)} = m_i\kappa$ for all $i \in [c]^+$; κ is any positive integer; and $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. As with the previous case, we can use the many-to-one mapping function $\mu_{\kappa,c} \circ \xi_{\pi,d,c}$ to map G into H. The resulting many-to-one mapping is also optimal.

Assume that G is a torus of shape $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, and H is a mesh of shape $\mathcal{M} = (m_1, m_2, \ldots, m_c)$, where $l_{\pi(i)} = 2m_i\kappa$ for all $i \in [c]^+$; κ is any positive integer; and $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. We can first use the contraction function $\xi_{\pi,d,c}$ to map the torus G into another torus G' of shape $\mathcal{L}' = (l_{\pi(1)}, l_{\pi(2)}, \ldots, l_{\pi(c)})$, then use the fold function $\bar{\nu}_{\mathcal{L}'}$ to map the torus G' into a mesh G'' of shape $\mathcal{L}'' = (m_1\kappa, m_2\kappa, \ldots, m_c\kappa)$, and finally use contraction function $\mu_{\kappa,c}$ to map the mesh G'' into the mesh H. Therfore, the composite mapping function is $\mu_{\kappa,c} \circ \bar{\nu}_{\mathcal{L}'} \circ \xi_{\pi,d,c} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{M}}$. Since $\mu_{\kappa,c}$, $\bar{\nu}_{\mathcal{L}'}$, and $\xi_{\pi,d,c}$ all have unit dilation cost, the many-to-one mapping function $\mu_{\kappa,c} \circ \bar{\nu}_{\mathcal{L}'} \circ \xi_{\pi,d,c}$ has unit dilation cost. Each node in H has exactly $(2\kappa)^c \prod_{i=c+1}^{d} l_{\pi(i)}$ nodes in G mapped into it. Thus, this many-to-one mapping is optimal.

5.4 Many-to-one Program Mapping

Using many-to-one graph mapping functions, we can perform many-to-one program mappings. With slight modifications, the scheme for one-to-one program mapping, which we described in Chapter 4, will still work in the many-to-one program mapping context.

5.4.1 Inverses for Many-to-one Mapping Functions

Since each element in the range of a many-to-one function corresponds to a set of elements in the domain, we can define the inverse of a many-to-one function $\mathcal{E} : D \to R$ to be the one-to-one mapping $\mathcal{E}^{-1} : R \to 2^D$ (2^D is the power set of D) such that for all $y \in R$, $\mathcal{E}^{-1}(y) = \{x \mid x \in D,\ \mathcal{E}(x) = y\}$. All of the many-to-one mapping functions defined in this chapter are of the form

$$\alpha \circ \beta,$$

where α is an embedding function (identity function as a special case) that has known inverse form, and β is a many-to-one contraction function that is either $\mu_{\kappa,d}$, $\mu_\kappa \circ \nu_{\mathtt{g}}$, $\mu_{\kappa,d} \circ \bar{\nu}_{\mathcal{L}}$, $\mu_{\kappa,c} \circ \xi_{\pi,d,c}$, or $\mu_{\kappa,c} \circ \bar{\nu}_{\mathcal{L}'} \circ \xi_{\pi,d,c}$. Therefore, we need only to determine inverses for these many-to-one contraction functions.

Inverse for $\mu_{\kappa,d}$

Let $(l_1, l_2, \ldots, l_d)$ and $(l'_1, l'_2, \ldots, l'_d)$ be radix-bases, where $l_i = \kappa l'_i$ for all $i \in [d]^+$. The contraction function $\mu_{\kappa,d} : \Omega_{(l_1,l_2,\ldots,l_d)} \to \Omega_{(l'_1,l'_2,\ldots,l'_d)}$ has been defined as follows:

$$\mu_{\kappa,d}((x_1, x_2, \ldots, x_d)) = \left(\left\lfloor \frac{x_1}{\kappa} \right\rfloor, \left\lfloor \frac{x_2}{\kappa} \right\rfloor, \ldots, \left\lfloor \frac{x_d}{\kappa} \right\rfloor \right)$$

for all $(x_1, x_2, \ldots, x_d) \in \Omega_{(l_1,l_2,\ldots,l_d)}$.

The inverse of $\mu_{\kappa,d}$, $\mu_{\kappa,d}^{-1} : \Omega_{(l'_1,l'_2,\ldots,l'_d)} \to 2^{\Omega_{(l_1,l_2,\ldots,l_d)}}$, can be specified as follows:

$$\mu_{\kappa,d}^{-1}((y_1, y_2, \ldots, y_d)) = \{(y_1\kappa + c_1, y_2\kappa + c_2, \ldots, y_d\kappa + c_d) \mid \forall i \in [d]^+,\ c_i \in [\kappa]\}$$

for all $(y_1, y_2, \ldots, y_d) \in \Omega_{(l'_1,l'_2,\ldots,l'_d)}$.

Inverse for $\mu_\kappa \circ \nu_{\mathtt{g}}$

The fold function $\nu_n : [n] \to [n/2]$ has been defined as follows:

$$\nu_n(x) = \begin{cases} x, & \text{if } 0 \le x < n/2, \\ n-1-x, & \text{otherwise} \end{cases}$$

for all $x \in [n]$, where n is an even positive integer.

The inverse of ν_n, $\nu_n^{-1} : [n/2] \to 2^{[n]}$, can be specified as follows:

$$\nu_n^{-1}(y) = \{y, n-1-y\}$$

for all $y \in [n/2]$.

Let $\mathtt{g} = 2\kappa\mathtt{h}$. The inverse of $\mu_\kappa \circ \nu_{\mathtt{g}}$, $\nu_{\mathtt{g}}^{-1} \circ \mu_\kappa^{-1} : [\mathtt{h}] \to 2^{[\mathtt{g}]}$, can thus be specified as follows:

$$\nu_{\mathtt{g}}^{-1} \circ \mu_\kappa^{-1}(y) = \{y\kappa + c, \mathtt{g} - 1 - y\kappa - c \mid c \in [\kappa]\}$$

for all $y \in [\mathtt{h}]$.

Inverse for $\mu_{\kappa,d} \circ \bar{\nu}_{\mathcal{L}}$

Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, where l_i is an even number for all $i \in [d]^+$. The fold function $\bar{\nu}_{\mathcal{L}} : \Omega_{(l_1,l_2,\ldots,l_d)} \to \Omega_{(l_1/2,l_2/2,\ldots,l_d/2)}$ has been defined as follows:

$$\bar{\nu}_{\mathcal{L}}((x_1, x_2, \ldots, x_d)) = (\nu_{l_1}(x_1), \nu_{l_2}(x_2), \ldots, \nu_{l_d}(x_d))$$

for all $(x_1, x_2, \ldots, x_d) \in \Omega_{(l_1,l_2,\ldots,l_d)}$.

The inverse of $\bar{\nu}_{\mathcal{L}}$, $\bar{\nu}_{\mathcal{L}}^{-1} : \Omega_{(l_1/2,l_2/2,\ldots,l_d/2)} \to 2^{\Omega_{(l_1,l_2,\ldots,l_d)}}$, can be specified as follows:

$$\bar{\nu}_{\mathcal{L}}^{-1}((y_1, y_2, \ldots, y_d)) = \{(e_1, e_2, \ldots, e_d) \mid \forall i \in [d]^+,\ e_i \in \{y_i, l_i - 1 - y_i\}\}$$

for all $(y_1, y_2, \ldots, y_d) \in \Omega_{(l_1/2,l_2/2,\ldots,l_d/2)}$.

Therefore, the inverse of $\mu_{\kappa,d} \circ \bar{\nu}_{\mathcal{L}}$, which is $\bar{\nu}_{\mathcal{L}}^{-1} \circ \mu_{\kappa,d}^{-1} : \Omega_{(l'_1,l'_2,\ldots,l'_d)} \to 2^{\Omega_{(l_1,l_2,\ldots,l_d)}}$ where $l_i = 2\kappa l'_i$ for all $i \in [d]^+$, can be specified as follows:

$$\begin{aligned}\bar{\nu}_{\mathcal{L}}^{-1} \circ \mu_{\kappa,d}^{-1}((y_1, y_2, \ldots, y_d)) = \{&(e_1, e_2, \ldots, e_d) \mid \\ &\forall i \in [d]^+,\ e_i \in \{y_i\kappa + c_i, l_i - 1 - y_i\kappa - c_i\} \text{ where } c_i \in [\kappa]\}\end{aligned}$$

for all $(y_1, y_2, \ldots, y_d) \in \Omega_{(l'_1,l'_2,\ldots,l'_d)}$.

Inverse for $\mu_{\kappa,c} \circ \xi_{\pi,d,c}$

Let d and c be positive integers such that $c < d$. Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$ and $\mathcal{L}' = (l_{\pi(1)}, l_{\pi(2)}, \ldots, l_{\pi(c)})$ be radix-bases, where $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. The contraction function $\xi_{\pi,d,c} : \Omega_{\mathcal{L}} \to \Omega_{\mathcal{L}'}$ has been defined as follows:

$$\xi_{\pi,d,c}((x_1, x_2, \ldots, x_d)) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(c)})$$

for all $(x_1, x_2, \ldots, x_d) \in \Omega_{\mathcal{L}}$.

The inverse of $\xi_{\pi,d,c}$, $\xi_{\pi,d,c}^{-1} : \Omega_{\mathcal{L}'} \to 2^{\Omega_{\mathcal{L}}}$, can be specified as follows:

$$\xi^{-1}_{\pi,d,c}((y_1,y_2,\ldots,y_c)) = \{(e_1,e_2,\ldots,e_d) \mid \text{if } i \in \{\pi(1),\pi(2),\ldots,\pi(c)\},\ e_i = y_{\pi^{-1}(i)}; \text{otherwise } e_i \in [l_i]\}$$

for all $(y_1,y_2,\ldots,y_c) \in \Omega_{\mathcal{L}'}$.

Let $\mathcal{L} = (l_1,l_2,\ldots,l_d)$ and $\mathcal{M} = (m_1,m_2,\ldots,m_c)$, where $l_{\pi(i)} = m_i\kappa$ for all $i \in [c]^+$; κ is any positive integer; and $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. The inverse function of $\mu_{\kappa,c} \circ \xi_{\pi,d,c}$, which is $\xi^{-1}_{\pi,d,c} \circ \mu^{-1}_{\kappa,c} : \Omega_{\mathcal{M}} \to 2^{\mathcal{L}}$, can be specified as follows:

$$\begin{aligned}\xi^{-1}_{\pi,d,c} \circ \mu^{-1}_{\kappa,c}((y_1,y_2,\ldots,y_c)) = \{(e_1\kappa + c_1, e_2\kappa + c_2, \ldots, e_d\kappa + c_d) \mid\ & \text{if } i \in \{\pi(1),\pi(2),\ldots,\pi(c)\},\ e_i = y_{\pi^{-1}(i)},\ c_i \in [\kappa]; \\ & \text{otherwise } e_i = 0,\ c_i \in [l_i]\}\end{aligned}$$

for all $(y_1,y_2,\ldots,y_c) \in \Omega_{\mathcal{M}}$.

Inverse for $\mu_{\kappa,c} \circ \bar{\nu}_{\mathcal{L}'} \circ \xi_{\pi,d,c}$

Let $\mathcal{L} = (l_1,l_2,\ldots,l_d)$, $\mathcal{M} = (m_1,m_2,\ldots,m_c)$, and $\mathcal{L}' = (l_{\pi(1)},l_{\pi(2)},\ldots,l_{\pi(c)})$ be radix-bases, where $l_{\pi(i)} = 2m_i\kappa$ for all $i \in [c]^+$; κ is any positive integer; and $\pi : [d]^+ \to [d]^+$ is a permutation on $[d]^+$. The inverse function of $\mu_{\kappa,c} \circ \bar{\nu}_{\mathcal{L}'} \circ \xi_{\pi,d,c}$, which is $\xi^{-1}_{\pi,d,c} \circ \bar{\nu}^{-1}_{\mathcal{L}'} \circ \mu^{-1}_{\kappa,c} : \Omega_{\mathcal{M}} \to 2^{\mathcal{L}}$, can be specified as follows:

$$\begin{aligned}\xi^{-1}_{\pi,d,c} \circ \bar{\nu}^{-1}_{\mathcal{L}'} \circ \mu^{-1}_{\kappa,c}((y_1,y_2,\ldots,y_c)) = \{(e_1\kappa + c_1, e_2\kappa + c_2, \ldots, e_d\kappa + c_d) \mid\ & \text{if } i \in \{\pi(1),\pi(2),\ldots,\pi(c)\},\ c_i \in [\kappa], \\ & e_i \in \{y_{\pi^{-1}(i)}, l_{\pi^{-1}(i)} - 1 - y_{\pi^{-1}(i)}\}; \\ & \text{otherwise } c_i \in [l_i],\ e_i = 0\}\end{aligned}$$

for all $(y_1,y_2,\ldots,y_c) \in \Omega_{\mathcal{M}}$.

5.4.2 Conflict-free Data Routing

Assume that a task graph is mapped into a system graph of smaller size using a many-to-one mapping function. Since two different parallel neighboring communication reqirements in the task graph may be mapped into the system graph so that the messages involved have the same physical source and physical destination, link conflicts are generally unavoidable. In this section we are interested only in the simulation of sets of parallel neighboring communications in the task graph in which no two messages involved have the same physical source and physical destination. Given any set of parallel neighboring communications in the task graph, we can first partition the set into maximum size subsets that have the property above, and then sequentially simulate these subsets in the system graph.

Definition 5.4.1 Assume that a graph G is mapped into another graph H using a mapping function $\mathcal{E}$. A set of parallel neighboring communications in G is *parallelizable* in H if under this mapping, no two messages involved have the same physical source and physical destination. □

By Definition 5.4.1, we have the following proposition.

Proposition 5.4.1 *Let $\mathcal{L} = (l_1, l_2, \ldots, l_d)$, $\mathcal{L}' = (l'_1, l'_2, \ldots, l'_d)$, and $\mathcal{M} = (m_1, m_2, \ldots, m_c)$ be radix-bases, where $l_i = \kappa l'_i$ for all $i \in [d]^+$, and κ is any positive integer. Assume that a task graph G of shape $\mathcal{L}$ is mapped into a system graph H of shape $\mathcal{M}$ with our general many-to-one mapping scheme, say the many-to-one mapping function $\mathcal{E} \circ \mu_{\kappa,d}$ where $\mathcal{E}$ is an embedding function, $\mu_{\kappa,d}$ embeds G into another graph G' of shape $\mathcal{L}'$, and $\mathcal{E}$ embeds G' into H. If we know that any permutation (scatter) type set of parallel neighboring communications in G' can be simulated in H by ρ parallel data movement steps, then any permutation (scatter) type set of parallel neighboring communications in G that is parallelizable in H can be simulated in H by ρ parallel data movement steps. The routing vector for this mapping function is the same as that for embedding function $\mathcal{E}$.* □

By Definition 5.4.1 and Theorem 4.5.1, we have the following corollary.

Corollary 5.4.1 *Assume that a d-dimensional task graph G is mapped into a system graph H using a many-to-one mapping function with unit dilation cost. Then any scatter type set of parallel neighboring communications in G that is parallelizable in H can be simulated in H by one parallel data movement step. The routing vector for any such mapping function with unit dilation cost is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

The following result follows from Corollary 5.4.1.

Corollary 5.4.2 *Assume that a d-dimensional task graph G is mapped into a c-dimensional system graph H using any one of the many-to-one mapping functions in $\{f_{\mathcal{L}} \circ \mu_\kappa \circ \nu_{\mathbf{g}},\ \pi \circ \mathcal{F}_{\mathcal{V}} \circ \mu_{\kappa,d} \circ \bar{\nu}_{\mathcal{L}},\ \mu_{\kappa,c} \circ \xi_{\pi,d,c},\ \mu_{\kappa,c} \circ \bar{\nu}_{\mathcal{L}'} \circ \xi_{\pi,d,c}\}$ with unit dilation cost. Then any scatter type set of parallel neighboring communications in G which is parallelizable in H can be simulated in H by one parallel data movement step. The routing vector for these mapping functions is $(i_1, i_2, \ldots, i_d)$, where $i_k = 1$ for all $k \in [d]^+$.* □

5.5 Conclusion

In this chapter, we introduce node evenness as another optimization objective in addition to dilation cost. For such a mapping, not only must we minimize the maximum distance between images of any pair of neighboring nodes, but we must also balance the number of nodes from the guest graphs mapped into each node in the host graph. A mapping with good node evenness ensures an even distribution of the nodes from the guest graph over the nodes in the host graph, and thus ensures an even computation load for each processor and good processor utilization.

We show that for toruses and meshes, we can obtain many-to-one graph mapping by first contracting the guest graph into some intermediate graph of the same size as the host graph, and then using our embedding schemes to embed the intermediate graph into the host graph. Although this decomposition of the many-to-one graph mapping into two steps will generally reduce our chances of global optimization, we show that in our special problem domain, we can generalize each of our embedding schemes into a many-to-one version by performing an appropriate contraction step before the embedding and still achieve optimal or good many-to-one mapping results.

The task graph contraction scheme based on edge grammars [BS84, BGK*85, BS87] is the only non-heuristic scheme in the literature that works for more than one graph family. However, since it allows for only one parameter, graph size, in the definition of graph families, the definitional power of the edge grammar is limited. For example, in the mesh family, each edge grammar can define only the square meshes of a fixed dimension, which is a small subset of the entire mesh family. For the same reason, within a truncatable graph family, for any integers x and y such that $x > y$, there is only one way to contract $G(x)$ into $G(y)$. As pointed out in [NS86], this is not optimal for many common parallel algorithms. By edge grammar, a mesh cannot be contracted into another mesh of different dimension. For every case in which the edge grammar can be used, our contractions can achieve at least the same contraction quality (with the contraction function $\bar{\nu}_{\mathcal{L}}$ defined in this chapter). By our approach, a torus or a mesh can be contracted into another torus or another mesh of either higher or lower dimension.

In this chapter, we also generalize our one-to-one program mapping approach in chapter 4 to the many-to-one program mapping approach.

Chapter 6
Conclusion

This thesis covers the communication optimization and abstraction in parallel processing systems. The major objectives of this research are (1) to design efficient schemes for mapping parallel programs onto parallel processing systems to minimize the communication overhead incurred by the mismatch of the communication characteristics of the parallel programs and those of the parallel processing systems, and (2) to support logical inter-process communication at execution time to improve program readability, verifiability, productivity, and portability. We use graph mapping as a tool to achieve these two objectives. In this chapter, we summarize our major contributions in this research. Our results fall into the following five categories: (1) embeddings among toruses and meshes, (2) many-to-one mappings among toruses and meshes, (3) mapping parallel programs onto parallel processing systems, (4) conflict-free data routing after program mapping, and (5) programming aspects of our program mapping approach.

Embeddings among toruses and meshes

An embedding is a bijective graph mapping. We study embeddings among toruses and meshes of various dimensions and various shapes. We use dilation cost as the optimization measure.

We generalize the concept of Gray code for the radix-2 (binary) numbering system to similar sequences for mixed-radix numbering systems. We use mixed-radix numbering systems as a basic tool to derive efficient embedding functions and perform dilation cost analyses.

Let G be a guest graph, and H be a host graph. We study the following kinds of embeddings among toruses and meshes: (i) basic embeddings, in which G is either a line or a mesh, and (ii) generalized embeddings, in which G and H can both be of higher dimension. Generalized embeddings are divided into two classes: embeddings for increasing dimension (in which the dimension of G is lower than that of H) and embeddings for lowering dimension (in which the dimension of G is higher than that of H). For increasing dimension, we study only those cases in which the shapes of G and H satisfy the condition of expansion. For lowering dimension, we study only those cases in which the shapes of G and H satisfy the condition of reduction.

All of our basic embeddings are optimal. For all cases except (i) G is a ring of odd size and H is a mesh, and (ii) G is a ring and H is a line, our embeddings have unit dilation cost; for the two exceptional cases above, our embeddings have an optimal dilation cost of 2.

For increasing dimension where the shapes of G and H satisfy the condition of expansion, our embeddings have dilaton costs of either 1 or 2, depending on the types of graphs of G and H. Except for the case where G is a torus of even size and H is a mesh, these embeddings are all optimal. For the special cases in which either (i) H is a hypercube, or (ii) G and H are both square and the dimension of H is divisible by that of G, the shapes of G and H always satisfy the condition of expansion, and our embeddings are always optimal.

For lowering dimension where the shapes of G and H satisfy the condition of reduction, the dilation costs of our embeddings depend on the shapes of G and H. These embeddings are not optimal in general. For the special case in which G is a hypercube, the shapes of G and H always satisfy the condition of reduction. For the special case in which both G and H are square, then either the shapes of G and H satisfy the condition of reduction, or G can always be embedded into H through a sequence of intermediate graphs in which every pair of successive graphs have shapes satisfying the condition of reduction. In either case, the dilation costs of our embeddings are $2\ell^{(d-c)/c}$ if G is a torus and H is a mesh, and $\ell^{(d-c)/c}$ otherwise, where ℓ is the length of the dimensions of G, d is the dimension of G, and c is the dimension of H. We also derive lower bounds on the dilation costs of embeddings among square toruses and square meshes. Using these lower bounds, we show that for fixed values of d and c, our embeddings are all optimal to within a constant.

Only a few special cases of the problem of embedding among toruses and meshes have been studied in the literature. Our embeddings cover many cases for which there was no previous result in the literature. A summary and comparison of our results with those in the literature is given in the conclusion of Chapter 3, together with a discussion of the differences between our results and the simulation results in the literature.

Let n be the size of G and H, and d and c be the dimensions of G and H respectively. For the sequential computation model, our basic embeddings, generalized embeddings for increasing dimension, and generalized embeddings for lowering dimension all have complexity proportional to cn; our embeddings based on simple reduction have complexity proportional to dn, where d and c are always less than or equal to $\log n$. A parallel implementation of our embeddings is also given in Chapter 4. For the parallel computation model, our basic embeddings, generalized embeddings for increasing dimension, and generalized embeddings for lowering dimension all have complexity proportional to c, and our embeddings based on simple reduction have complexity proportional to d.

Many-to-one mappings among toruses and meshes

An embedding is a one-to-one mapping between a host graph and a guest graph of the same size. In a many-to-one mapping, the size of the guest graph can be greater than that of the host graph, and more than one node in the guest graph can be mapped into a single node in the host graph. Many-to-one mappings are important in applications because the size of a task graph is usually greater than that of a system graph.

For many-to-one mappings, we introduce node evenness as another optimization objective in addition to dilation cost. For such a mapping, not only must we minimize the maximum distance between images of any pair of neighboring nodes, but we must also balance the number of nodes from the guest graphs mapped into each node in the host graph. A mapping with good node evenness ensures an even distribution of the nodes from the guest graph over the nodes in the host graph, and thus ensures an even computation load for each processor and good processor utilization.

We show that for toruses and meshes, we can obtain many-to-one graph mapping by first contracting the guest graph into some intermediate graph of the same size as the host graph, and then using our embedding schemes to embed the intermediate graph into the host graph. Although this decomposition of the many-to-one graph mapping into two steps will generally reduce our chances of global optimization, we show that in our special problem domain, we can generalize each of our embedding schemes into a many-to-one version by performing an appropriate contraction step before the embedding and still achieve optimal or good many-to-one mapping results.

Mapping parallel programs onto parallel processing systems

We use graph mapping technique to map system topology independent parallel programs onto parallel processing systems. We identify three tasks in implementing a program mapping: (1) logical address identification, by which each processor identifies the process in the parallel program to be mapped into it; (2) code loading, by which codes for different processes get loaded into the corresponding processors; and (3) translation table generation, by which each processor can transform inter-process communication into inter-processor communication automatically at execution time.

For logical address identification, we propose a parallel solution based on parallel evaluation of the inverse of the mapping function by all of the processors. Since all of our mapping functions have time complexities either proportional to a constant or proportional to $(d-c)$, where d is the dimension of the task graph and c is the dimension of the system graph, our logical address identification has low time complexity and can be performed at execution time.

For code loading, we propose a parallel approach based on logical address identification. The time required for the program loading process is proportional to the number of different code types used in the parallel program, but not to the size of the task graph. This approach works especially well for large parallel programs in which only a limited number of code types are used. In this approach, we assume that broadcast is the only means for the host to send messages to the physical nodes. Since a broadcast network is available or simulated in all SIMD, MSIMD, and MIMD systems, this approach is applicable to a wide range of parallel processing systems.

For translation table generation, we propose two methods. The first method is based on the parallel computation of the embedding function and can be applied to all of our embedding functions. The second method is based on parallel data movements of the ordered pairs "(logical address, physical address)" computed in the logical address identification stage and is used only if the embedding has unit dilation cost. The time complexity of the first method is proportional to the product of d and the complexity of the embedding function, where d is the dimension of the task graph. The time complexity of the second method under the bidirectional link assumption is a constant.

Conflict-free data routing after program mapping

In graph mappings, dilation cost measures the maximum distance in the host graph between the images of any pair of neighboring nodes in the guest graph. In the mapping of a task graph (which represents a parallel program) to a system graph (which represents a parallel processing system), dilation cost gives a measure of the maximum number of links a message must traverse in the worst-case if a single process sends a message to some other process. For the case in which more than one process wants to communicate with some other processes, multiple messages may need to traverse a given link at the same time, causing additional delay in the delivery of the messages. Because of the link contention problem, the fact that a mapping has a small dilation cost does not generally imply that any set of parallel neighboring communications in the task graph can be simulated in the system graph with a small data routing complexity.

We define two types of sets of parallel neighboring communications in a task graph: (i) permutation type (at any instant, each node in the task graph can send only one message to one of its neighbors and receive only one message from one of its neighbors), and (ii) scatter type (at any instant, each node in the task graph can send one message to each of its neighbors and receive one message from each of its neighbors). We design for each of our graph mapping functions a data routing strategy to achieve conflict-free simulation in the system graph of either any scatter type set of parallel neighboring communications in the task graph if the mapping has unit dilation cost, or any permutation type set of parallel neighboring communications in the task graph otherwise. In most cases, these data routing strategies can simply take the form of data routing vectors. We propose a simple data routing scheme, the *shortest-path data routing scheme,* that can automatically carry out our data routing strategies at execution time. This scheme has low overhead, and can be easily implemented either by software or by hardware. This scheme uses our data routing strategies and local information to ensure that all of the messages can move along the shortest paths to their destinations without link conflicts.

We analyze the correponding data routing complexities for each of our graph mappings. Let ρ denote the dilation cost of a graph mapping. The data routing complexity for each of our graph mappings is ρ (17 cases), $\rho + 1$ (3 cases), $\rho + 2$ (2 cases), 2ρ (2 cases), or 4ρ (2 cases). A comparison of data routing complexity and dilation cost for each of our embeddings is given in Section 4.7, Chapter 4.

Programming aspects of our program mapping approach

Our program mapping approach supports communication abstraction and portability of parallel programs. All communication in parallel programs can be specified on the logical task graph level. System topologies are completely transparent to these programs. Programmers do not need to concern themselves with low-level data routing. The logical inter-process communication is not transformed into inter-processor communication until execution time. As a result, even the object code of such parallel programs are transportable. The communication abstraction supported by our program mapping approach can also improve readability, verifiability, and productivity of parallel programs.

As future work, we expect to expand the research reported in this dissertation in the following directions: (i) generalize our graph mapping to more graph families; (ii) generalize the mapping model for many-to-one mapping to allow the nodes and edges to have various weights; (iii) implement the shortest-path data routing scheme on an existing system.

Appendix

In this appendix, we prove that $\sum_{k=0}^{d-1}\binom{k}{\lfloor k/2\rfloor}$ can be rewritten as $\varepsilon_{d-1}2^{d-1}$, where $\varepsilon_0 = \varepsilon_1 = \varepsilon_2 = 1$, and for all $d \geq 3$, $\varepsilon_{d-1} > \varepsilon_d$. It is easy to check that for $d \in [3]$, the assertion is true. Therefore, we only need to prove the case in which $d \geq 3$.

Proposition A.1 For all positive integers k, $\binom{k}{\lfloor k/2\rfloor} = 2^{k-1}C_{k-1}$, where

$$C_{k-1} = \begin{cases} \prod_{j=1}^{(k-1)/2}(1-1/(2j+2)), & \text{for } k-1 \text{ even and } k-1 \geq 0; \\ \prod_{j=2}^{k/2}(1-1/(2j)), & \text{for } k-1 \text{ odd and } k-1 \geq 1. \end{cases}$$

Proof. We use induction on odd k's and even k's.

Case 1. k is even.
Basis. $k = 2$.
We have $\binom{2}{1} = 2 = 2C_1$.
Induction hypothesis. Assume that the proposition is true for all positive, even integers $k \leq a$, where a is an even number.
Induction step. Prove for $k = a + 2$.

$$\begin{aligned}\binom{a+2}{\lfloor (a+2)/2\rfloor} &= \binom{a+2}{(a+2)/2} \\ &= \frac{(a+2)!}{((a+2)/2)!((a+2)/2)!} \\ &= 2^2(1-\frac{1}{a+2})\binom{a}{a/2} \\ &= 2^2(1-\frac{1}{a+2})2^{(a-1)}C_{a-1} \\ &= 2^{(a+2)-1}(1-\frac{1}{a+2})\prod_{j=2}^{a/2}(1-\frac{1}{2j}) \\ &= 2^{(a+2)-1}C_{(a+2)-1}.\end{aligned}$$

Case 2. k is odd.

Basis. $k = 1$.
We have $\binom{1}{0} = 1 = 2^0C_0$.
Induction hypothesis. Assume that the proposition is true for all positive, odd integers $k \leq a$, where a is an odd number.

Induction step. Prove for $k = a + 2$.

$$\begin{aligned}
\binom{a+2}{\lfloor (a+2)/2 \rfloor} &= \binom{a+2}{(a+1)/2} \\
&= \frac{(a+2)!}{((a+1)/2)!((a+3)/2)!} \\
&= 2^2(1 - \frac{1}{a+3})\binom{a}{\lfloor a/2 \rfloor} \\
&= 2^2(1 - \frac{1}{a+3})2^{a-1}C_{a-1} \\
&= 2^{(a+2)-1}(1 - \frac{1}{a+3})\prod_{j=1}^{(a-1)/2}(1 - \frac{1}{2j+2}) \\
&= 2^{(a+2)-1}C_{(a+2)-1}.
\end{aligned}$$

□

Proposition A.2 For all positive integers k, $C_k \leq C_{k-1}$.
Proof. We consider two cases:

Case 1. k is odd.
Since $C_{k-1} = \prod_{j=1}^{(k-1)/2}(1 - 1/(2j+2)) = \prod_{j'=2}^{(k+1)/2}(1 - 1/(2j'))$, we have $C_{k-1} = C_k$.

Case 2. k is even.
Since $C_k = \prod_{j=1}^{k/2}(1-1/(2j+2)) = \prod_{j'=2}^{k/2+1}(1-1/(2j')) = (1-1/(k+2))C_{k-1}$, we have $C_k < C_{k-1}$. □

Proposition A.3 Let $t_m = \sum_{k=0}^{m}\binom{k}{\lfloor k/2 \rfloor}$. Then $t_m = \varepsilon_m 2^m$, where $\varepsilon_m = (\varepsilon_{m-1} + C_{m-1})/2$ and $C_{m-1} < \varepsilon_m < \varepsilon_{m-1}$ for all $m \geq 3$.
Proof. We use induction on m.
Basis. $m = 3$.
Since $t_3 = 7$, $\varepsilon_2 = 1$, and $C_2 = 3/4$, we have $t_3 = \varepsilon_3 2^3$, where $\varepsilon_3 = 7/8 = (\varepsilon_2 + C_2)/2$, and $C_2 < \varepsilon_3 < \varepsilon_2$.
Induction hypothesis. Assume that the proposition is true for all positive integers $m \leq a$.
Induction step. Prove for $m = a + 1$.
Since $t_{a+1} = \sum_{k=0}^{a+1}\binom{k}{\lfloor k/2 \rfloor} = t_a + 2^a C_a = 2^{a+1}(\varepsilon_a + C_a)/2$, we have $t_{a+1} = \varepsilon_{a+1}2^{a+1}$, where $\varepsilon_{a+1} = (\varepsilon_a + C_a)/2$.
Since $C_{a-1} < \varepsilon_a < \varepsilon_{a-1}$ and $C_a \leq C_{a-1}$, we have $C_a < \varepsilon_a$. Hence, $\varepsilon_{a+1} < \varepsilon_a$ and $C_a < \varepsilon_{a+1}$. Therefore, $C_a < \varepsilon_{a+1} < \varepsilon_a$. □

From the recurrence relation $\varepsilon_m = (\varepsilon_{m-1} + C_{m-1})/2$ for all $m \geq 3$, we also have $\varepsilon_m = (1/2)^{m-2} + \sum_{k=2}^{m-1}(1/2)^{m-k}C_k$, for all $m \geq 3$.

From the last three propositions, we can conclude that $\sum_{k=0}^{d-1}\binom{k}{\lfloor k/2 \rfloor}$ can be rewritten as $\varepsilon_{d-1}2^{d-1}$, where $\varepsilon_0 = \varepsilon_1 = \varepsilon_2 = 1$, and for all $d \geq 3$, $\varepsilon_{d-1} > \varepsilon_d$.

Bibliography

[AR82] Romas Aleliunas and Arnold L. Rosenberg. On embedding rectangular grids in square grids. *IEEE Trans. Computer*, C-31(9):907–913, September 1982.

[Ata85] Mikhail J. Atallah. On multidimensional arrays of processors. *Proc. 1985 Allerton Conf., also as Purdue Technical Report CSD-TR-528*, 1–11, 1985.

[BB82] Dan H. Ballard and Christopher M. Brown. *Computer Vision.* Prentice-Hall, 1982.

[BCLR86] Sandeep Bhatt, Fan Chung, Tom Leighton, and Arnold Rosenberg. Optimal simulations of tree machines. *27th Annual Symposium on Foundations of Computer Science*, 274–282, October 1986.

[Ber83] Francine Berman. Edge grammars and parallel computation. *Proceedings of the 1983 Allerton Conference, Urbana, Illinois*, 214–223, 1983.

[BGK*85] Francine Berman, Michael Goodrich, Charles Koelbel, W.J. Robison III, and Karen Showell. Prep-p: a mapping processor for chip computers. *Proceedings of International Conference on Parallel Processing*, 731–733, 1985.

[BMS87] Said Bettayeb, Zevi Miller, and I. Hal Sudborough. Embedding grids into hypercubes. *Paper draft*, 1–30, August 1987.

[BS84] Francine Berman and Lawrence Snyder. On mapping parallel algorithms into parallel architectures. *Proceedings of International Conference on Parallel Processing*, 307–309, 1984.

[BS87] Francine Berman and Lawrence Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4:439–458, 1987.

[Bun72] David M. Bunton. *Abstract and Linear Algebra.* Addison-Wesley, 1972.

[CS86] Tony F. Chan and Youcef Saad. Multigrid algorithms on the hypercube multiprocessor. *IEEE Transactions on Computers*, C-35(11):969–977, November 1986.

[DEL78a] R. A. DeMillo, S. C. Eisenstat, and R. J. Lipton. On small universal data structures and related combinatorial problems. *Proc. Johns Hopkins Conf. on Information Sciences and Systems, Baltimore, Md.*, 408–411, 1978.

[DEL78b] Richard A. DeMillo, Stanley C Eisenstat, and Richard J. Lipton. Preserving average proximity in arrays. *Communications of the ACM*, 21(3):228–231, March 1978.

[DJ86] Sanjay R. Deshpande and Roy M. Jenevein. Scalability of a binary tree on a hypercube. *Proceedings of International Conference on Parallel Processing*, 661–668, 1986.

[Ell88] John A. Ellis. Embedding rectangular grids into square grids. *Lecture Notes in Computer Science (319), Springer-Verlag. (Also: Proc. of the 3rd Aegean Workshop on Computing, AWOC88, Corfu, Greece.)*, 181–190, 1988.

[Fit74] Carl H. FitzGerald. Optimal indexing of the vertices of graphs. *Mathematics of Computation*, 28(127):825–831, July 1974.

[Fox83] Geoffrey C. Fox. Decomposition of scientific problems for concurrent processors. *Cal. Tech. Technical Report, CALT-68-986*, 1983.

[GK84] Allan Gottlieb and Clyde P. Kruskal. Complexity results for permuting data and other computations on parallel processors. *Journal of the Association for Computing Machinery*, 31(2):193–209, April 1984.

[Har66] L. H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1:385–393, 1966.

[Hil85] W. Daniel Hillis. *The Connection Machine.* The MIT Press, 1985.

[HJ87] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in boolean cubes with expansion two dilation two. *Proceedings of International Conference on Parallel Processing*, 188–191, 1987.

[HKS*83] Tsutomu Hoshino, Toshio Kawai, Tomonori Shirakawa, Junichi Higashino, Akira Yamaoka, Takashi Ito, Hachidai Sato, and Kazuo Sawada. Pacs: a parallel microprocessor array for scientific calculations. *ACM Tran. on Computer Systems*, 1(3):709–728, August 1983.

[HMR73] Alan J. Hoffman, Michael S. Martin, and Donald J. Rose. Complexity bounds for regular finite difference and finite element grids. *SIAM J. Numer. Anal.*, 10(2):364–369, April 1973.

[HMR83] Jia-Wei Hong, Kurt Mehlhorn, and Arnold L Rosenberg. Cost trade-offs in graph embeddings, with applications. *Journal of the Association for Computing Machinery*, 30(4):709–728, October 1983.

[JGD87] L. Jamieson, D. Gannon, and R. Douglass, editors. *The Characteristics of Parallel Algorithms.* MIT Press, 1987.

[KA85] S. Rao Kosaraju and Mikhail J. Atallah. Optimal simulations between arrays of processors. *Preliminary Version*, 1–21, 1985.

[KA88] S. Rao Kosaraju and Mikhail J. Atallah. Optimal simulations between mesh-connected arrays of processors. *Journal of the Association for Computing Machinery*, July 1988.

[KWA82] T. Kushner, A. Y. Wu, and Rosenfeld A. Image processing on zmob. *IEEE Transactions on Computers*, C-31(10), October 1982.

[LED76] R. J. Lipton, S. C. Eisenstat, and R. A. DeMillo. Space and time hierarchies for classes of control structures and data structures. *Journal of the Association for Computing Machinery*, 23(4):720–732, October 1976.

[Lei83] C.E. Leiserson. *Area-Efficient VLSI Computation.* MIT Press, Cambridge, Massachusetts, 1983.

[LM87a] Hungwen Li and Massimo Maresca. Polymorphic-torus architecture for computer vision. *IBM Technical Report*, 1–30, February 1987.

[LM87b] Hungwen Li and Massimo Maresca. Polymorphic-torus network for supercomputing. *IBM Technical Report RC 12568 (#56551)*, 1–36, March 1987.

[LR82] F. T. Leighton and A. L. Rosenberg. Three-dimensional circuit layouts. *Unpublished manuscript*, 1982.

[LW87] Ten-Hwang Lai and William White. Embedding pyramids in hypercubes. *The Ohio State University Technical Report O SU-CISRC-11/87-TR41*, 1–25, November 1987.

[MN86] Yuen-wah Ma and Bhagirath Narahari. Optimal mappings among interconnection networks for performance evaluation. *Proceedings of the 6th International Conference on Distributed Computing Systems*, 16–25, May 1986.

[MS88] Burkhard Monien and I. Hal Sudborough. Simulating binary trees on hypercubes. *Lecture Notes in Computer Science (319), Springer-Verlag. (Also: Proc. of the 3rd Aegean Workshop on Computing, AWOC88, Corfu, Greece.*, 170–180, 1988.

[MT87] Yuen-wah Eva Ma and Lixin Tao. Embeddings among toruses and meshes. *Proceedings of International Conference on Parallel Processing*, 178–187, August 1987.

[NS80a] D. Nassimi and S. Sahni. Finding connected components and connected-ones on a mesh-connected parallel computer. *SIAM J. Comput.*, 1980.

[NS80b] David Nassimi and Sartaj Sahni. An optimal routing algorithm for mesh-connected parallel computers. *Journal of the Association for Computing Machinery*, 27(1):6–29, January 1980.

[NS81] David Nassimi and Sartaj Sahni. Data broadcasting in simd computers. *IEEE Transactions on Computers*, C-30(2):101–107, February 1981.

[NS82] David Nassimi and Sartaj Sahni. Parallel algorithms to set up the benes permutation network. *IEEE Transactions on Computers*, C-31(2):148–154, February 1982.

[NS86] Philip A. Nelson and Lawrence Snyder. Programming solutions to the algorithm contraction problem. *Proceedings of International Conference on Parallel Processing*, 258–261, August 1986. n6.

[Oru84] A. Yavuz Oruc. A classification of cube-connected networks with a simple control scheme. *IEEE Transactions on Computers*, C-33(8):769–772, August 1984.

[PBe85] J. Potter, K Batcher, and etc. *The Massively Parallel Processor.* The MIT Press, 1985.

[Pot83] J. Potter. Image processing on the massively parallel processor. *Computer*, 62–67, January 1983.

[PV79] Franco P. Preparata and Jean Vuillemin. The cube-connected-cycles: a versatile network for parallel computation. *IEEE*, 140–147, 1979.

[RJD77] E. M. Reingold, Nievergelt J., and N. Deo. *Combinatorial Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1977.

[RK82] Azriel Rosenfeld and Avinash C. Kak. *Digital Picture Processing.* Volume 1, Academic Press, 2 edition, 82.

[Ros75] Arnold L. Rosenberg. Preserving proximity in arrays. *SIAM J. Comput*, 4(4):443–460, December 1975.

[Ros78] Arnold L. Rosenberg. Data encodings and their costs. *Acta Informatica*, 9:273–292, 1978.

[Ros79] Arnold L. Rosenberg. Encoding data structures in trees. *Journal of the Association for Computing Machinery*, 26(4):668–689, October 1979.

[Ros83] A. L. Rosenberg. Three-dimensional vlsi: a case study. *Journal of the Association for Computing Machinery*, 30(3), July 1983.

[RS78] Arnold L. Rosenberg and Lawrence Snyder. Bounds on the costs of data encodings. *Math. Systems Theory*, 12:9–39, 1978.

[S87] Greenberg D. S. Optimum expansion embeddings of meshes in hypercubes. *Technical Report YALEU/CSD/RR-535*, 1987.

[SC87] Joel H. Saltz and Marina C. Chen. Automated problem mapping: the crystal runtime system. *Research Report YALEU/DCS/RR-510*, 11 pages, January 1987.

[SEM87] P. Sadayappan, Fikret Ercal, and Steven Martin. Mapping finite element graphs onto processor meshes. *Proceedings of International Conference on Parallel Processing*, 192–195, August 1987.

[Sny82] Lawrence Snyder. Introduction to the configurable, highly parallel computer. *Computer*, 47–56, January 1982.

[Sny83] Lawrence Snyder. Introduction to the poker parallel programming environment. *Proceedings of International Conference on Parallel Processing*, 289–292, 1983.

[Sny84] Lawrence Snyder. Parallel programming and the poker programming environment. *Computer*, 27–36, July 1984.

[SS85] Youcef Saad and Martin H. Schultz. Topological properties of hypercubes. *Research Report YALEU/DCS/RR-389*, 1–17, June 1985.

[Tho79] C. D. Thompson. Area-time complexity for vlsi. *Proc. 11th ACM Symp. on Theory of Computing (Atlanta, Ga., May 1979)*, 81–88, May 1979.

[TK77] C.D. Thompson and H.T. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20(4):263–271, April 1977.

[TM75] J. P. Tremblay and R. Manohar. *Discrete Mathematical Structures with Applications to Computer Science*. McGraw-Hill Inc., 1975.

[Val81] L. G. Valiant. Universality considerations in vlsi circuits. *IEEE Transactions on Computers*, C-30:135–140, 1981.

[Val82] L. G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, May 1982.

[Wu85] Angela Y. Wu. Embedding of tree networks into hypercubes. *Journal of Parallel and Distributed Computing*, 2:238–249, 1985.