Energy-Efficient Algorithms on Mesh-Connected Systems with Additional Communication Links

by

Patrick J. Poon

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2013

Doctoral Committee:

Professor Quentin F. Stout, Chair
Associate Professor Kevin Compton
Professor Jeffrey Lagarias
Associate Professor Seth Pettie

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Energy-Efficient Algorithms on Mesh-Connected Systems with Additional Communication Links

by

Patrick J. Poon

Chair: Quentin F. Stout

Energy consumption has become a critical factor constraining the design of massively parallel computers, necessitating the development of new models and energy-efficient algorithms. In this work we take a fundamental abstract model of massive parallelism, the mesh-connected computer, and extend it with additional communication links motivated by recent advances in on-chip photonic interconnects. This new means of communication with optical signals rather than electrical signals can reduce the energy and/or time of calculations by providing faster communication between distant processing elements. Processors are arranged in a two-dimensional grid with wire connections between adjacent neighbors and an additional one or two layers of non-crossing optical connections. Varying constraints on the layout of optics affect how powerful the model can be. In this dissertation, three optical interconnection layouts are defined: the optical mesh, the optical mesh of trees, and the optical pyramid. For each layout, algorithms for solving important problems are presented. Since energy usage is an important factor, running times are given in terms of a peak-power constraint, where peak power is the maximum number of processors active at any one time. These results demonstrate advantages of optics in terms of improved time and energy usage over the standard mesh computer without optics. One of the most signif-

icant results shows an optimal nonlinear time/peak-power tradeoff for sorting on the optical pyramid. This work shows asymptotic theoretical limits of computation and energy usage on an abstract model which takes physical constraints and developing interconnection technology into account.

CHAPTER 1

Introduction

In recent years, power consumption has become an important factor to consider when designing computers from mobile devices to supercomputers, necessitating the development of new models and energy-efficient algorithms. As the number of processing units in machines has increased, so has energy usage, which is a problem when there is a limit on the available total energy or peak power. In addition, increasing transistor densities has brought about physical constraints of heat dissipation, limiting the fraction of chips operating at full speed. Research has shown that the maximum utilization of transistors on chips will only continue to decrease in future generations of processors [22, 73]. Another limitation is the fact that processors occupy physical volume: parallel computers with many processors will always contain some processors that are physically far away from each other and therefore require more time and energy to communicate between. When data is distributed among many processors, algorithms must take advantage of locality to save time and reduce energy usage or else running massive parallel computers will be infeasible. This important fact is typically ignored in shared-memory models such as the parallel random access machine (PRAM).

Many different models of distributed-memory parallel architectures have been studied, such as hypercubes, meshes, and pyramids. Several mesh models have been analyzed and built ever since von Neumann introduced the cellular automata model [75]. The mesh is a scalable parallel computer architecture and has also been used as a model of many physical processes where locality strongly affects behavior [69, 77]. Here, both roles are intertwined in the consideration of energy consumption in massively parallel computation.

The goal of this doctoral research is to develop energy-efficient algorithms on a model that captures physical constraints and incorporates new interconnection technology. Specifically, the model is a mesh-connected computer with the addition of connections, called "optical" connections or *optics*. The use of optical connections in this model is motivated by current research on optical interconnects between processing units in parallel computers. These optical interconnects provide fast communication between distant processing

units using little energy [58]. Restrictions imposed on these optical connections in this model are that they cannot cross each other and that they must be straight in a single layer of optical connections. Research has shown that optical waveguide crossings have a significant impact on optical signals, and therefore, we prohibit crossings [15]. It is also difficult to build multiple layers in hardware [11], so we allow only one or two layers of optics and do not allow the optics to cross within any one layer. While "optics" will be the term used to refer to these additional communication links, the actual implementation may be other technologies as well, such as carbon nanotubes or whatever may emerge in future interconnects and fabrication. The analyses only requires that the interconnect provides the capability of being able to transmit information quickly over long distances with low power relative to the capabilities of standard wires.

In the following chapters, results will be presented for three different layouts of optical connections, each constructed given certain restrictions on whether different lengths of optics are allowed and if there are one or two layers of optics. First, in Chapter 4, a simple model with one layer of fixed-length optical connections is considered, which will be referred to as the optical mesh. Results in this chapter give faster algorithms for some problems over the standard mesh model and therefore also give algorithms that consume less energy. In addition, the tradeoffs between time and power usage of these algorithms are analyzed. Note that all previous mesh algorithms assumed all processors are powered all the time so these time-power tradeoffs are new. These results with the most basic optical connection layout affirm the benefits of optical connections. Second, in Chapter 5, we consider the model with a more complex interconnection network with two layers of optical connections called an optical mesh of trees. Most of the results for this model show a significant improvement of running time over the standard mesh. In addition, a sorting algorithm using minimal energy is given. Third, in Chapter 6, we consider the model with one layer of optical connections called the optical pyramid. One of the most interesting results for the optical pyramid is a unique algorithm for sorting that incurs a sublinear increase in running time if the available power is decreased.

Before describing layouts, definitions of frequently used operations and some general lower bound results involving the mesh with optical connections will be stated in Chapter 3. Afterwards, the following chapters will describe implementation of these operations in their respective models. Once these implementations are shown, more difficult algorithms for harder problems can be presented. A variety of types of problems will be analyzed ranging from those that involve digital images to graphs and ordered data. Permutation, sorting, and finding the minimum spanning forest of a graph are examples of the more complex problems presented.

In Chapter 2, the model is formally defined and an overview of related work is given.

CHAPTER 2

Preliminaries

2.1 Description of Model

A *mesh-connected computer*, or *mesh*, of size N is a parallel computer consisting of N processors arranged in an $n \times n$ square lattice, where $n^2 = N$. Running times are expressed in terms of N to emphasize the relationship with the size of the mesh, which is also usually the size of the input. To simplify exposition, we assume that n is a power of 2, with modifications to the more general case being straightforward. $P(i, j)$ denotes the processor at coordinates (i, j), for $i, j \in \{0, 1, \ldots, n-1\}$, and for notational convenience, all indexing will start at 0. Each processor $P(i, j)$ can communicate with only its four nearest adjacent neighbors $P(i \pm 1, j)$ and $P(i, j \pm 1)$, if they exist. Each processor has a fixed number of words of memory, each of size $\Theta(\log N)$, enough to store its location in the mesh and a constant number of other values. Processors have constant area and communication links between processors have constant length, so all operations on values stored in a processor's memory, including transmitting a value to a neighbor, take constant time and energy.

We say that a processor is *active* if it is calculating or communicating and is otherwise inactive and not using energy. More precisely, an inactive processor is in a very low power sleep mode and the algorithms in this work consider the power needed above this level. We define *power* as the number of processors active at a given time and total *energy* usage as the integral of power over time. Energy can often be viewed as equivalent to the concept of *work* as it is used in parallel algorithms. In this model, we give some algorithms that minimize time given *peak power*, where peak power is defined as the maximum number of processors using energy at any one time over the duration of the algorithm. That is, the total number of processors calculating or communicating at any one time is limited by the peak-power constraint. Most of the algorithms that will be presented run at peak power most of the time, so their total energy is $\Theta(\text{time} \times \text{peak power})$ unless otherwise stated. Peak power is denoted by S. It is often the case that there is a tradeoff between peak power

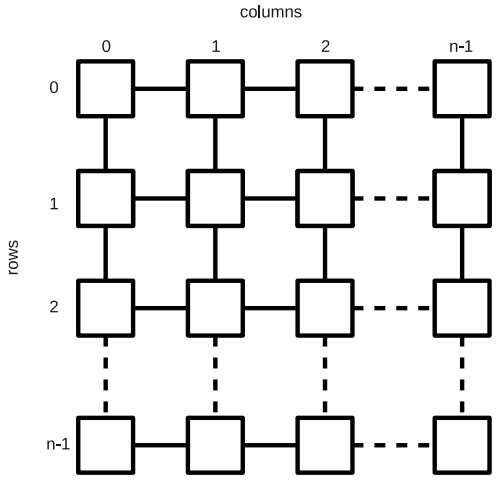


Figure 2.1: An $n \times n$ mesh computer.

usage and time since problems are usually solved faster when more processors are working together. In addition to algorithms with time-power tradeoffs, algorithms that minimize total energy usage are important.

When defining algorithms for this model, we must ensure that the peak power constraint is not violated. One way to make this evident is to use algorithms with known running times and power usage as subroutines. In addition, algorithms are often expressed in terms of data movement operations specified by available peak power, which represent sequences of communication steps, rather than explicitly indicating which processors are on at a given time.

The model being studied will be called the *mesh with optical connections*. The added communication links between processors will be called *optical connections* or *optics*, and the communication links in the standard mesh will be called *wire connections* or *wires*. This differentiates between the connections in the standard mesh model and the added connections. Communication time over an optical connection is counted the same as communication over a single wire on the mesh, which is a constant. As mentioned previously, optical connections cannot cross each other and they must be straight in a single layer of optical connections. For convenience, it is also required that the optics are oriented such that each is parallel to an axis. In addition, we allow only one or two layers of noncrossing optics. Optics have a fixed width so the total area taken up by optics must be $O(N)$. Different layouts of optical connections will be defined in the following chapters.

A *submesh* is the mesh induced by a rectangular subset of the processors in a mesh. It is often useful to refer to submeshes because we can partition a mesh into submeshes and

treat each submesh independently as regular meshes.

An *optical mesh* is a subset of the processors in the mesh with optical connections that form a mesh using only optical connections or can simulate a mesh with a constant factor overhead. The optical interconnection network in the fixed-length optical connections model defined in Chapter 4 consists of an optical mesh, and the mesh with optical pyramid defined in Chapter 6 contains many optical meshes of different sizes.

A d-dimensional mesh of size n^d is composed of n $(d-1)$-dimensional meshes of size n^{d-1}. Each processor has coordinates $(x_0,\ldots,x_{d-1})$, for $x_0,\ldots,x_{d-1} \in \{0,1,\ldots,n-1\}$. A processor at coordinates $(x_0,\ldots,x_{d-1})$ can communicate directly with only its $2d$ adjacent processors, which are the processors at coordinates $(x_0 \pm 1, x_1, \ldots, x_{d-1}), (x_0, x_1 \pm 1, \ldots, x_{d-1}), \ldots, (x_0, x_1, \ldots, x_{d-1} \pm 1)$, if they exist.

2.2 Overview

The goal is to develop algorithms for this new parallel machine model that take power usage and energy into account. The idea is that this model will allow problems to be solved faster or with less energy (or both) than the standard mesh model. Often, these algorithms will minimize running time given a peak power restriction, thus also reducing total energy usage. There is an inherent tradeoff between power usage and running time, and this will be analyzed. Many algorithms have standard linear time-power tradeoffs, while others have a non-linear tradeoff that can be shown to be more efficient than a standard linear tradeoff.

The field of energy-efficient computing will continue to grow as an active area of research, and this work seeks to contribute to it. The results in this work provide motivation for more energy-aware algorithms and future hardware.

2.3 Motivation for the Model

Parallel computing is important and has become prevalent in today's society as parallel computers can be found in a variety of places from small handheld devices to large supercomputers. In recent years, computers have become increasingly more parallel with the addition of more processors and cores to machines. There are predictions that this trend will continue and the number of cores on a single chip will continue to increase [4, 12, 30]. Because of such developments, parallel computing is currently an important field of study and much research is needed to determine how to best utilize many cores. This justifies the study of parallel algorithms.

Adding connections between processors in the mesh-connected computer model is representative of what is possible in reality because it takes into account several factors that impose real physical constraints on on-chip hardware. The first issue is the space constraint, which limits several aspects, including the amount of bandwidth across the chip, the number of possible connections, the area the interconnects use, the number of connections incident to a processing node, the number of layers of connections, and memory per processor. These issues are due to the fact that interconnects and processors take up physical space on the chip. The second issue is the energy usage factor. As seen even in processors today, not all cores on a chip can run at full frequency at once due to the limits of heat dissipation [31, 32]. One way to mitigate these factors is to use many simple cores as opposed to few complex cores [54]. Further, exascale predictions assume that simpler cores and optical interconnects will be necessary to continue increasing performance [61]. All of these aspects are captured in the mesh with optical connections; therefore, it is an appropriate model.

Power limitations motivate the development of algorithms that demonstrate the awareness of the power limits of the chip. This has not been studied much before, at least not at the on-chip level, and these problems must be addressed. In general, energy is a limited resource and we strive to be as efficient as possible with energy usage. Algorithms that are adaptable to power restrictions will allow parallel machines to run more efficiently, especially in low-power situations.

Since this models current and future technology, there are not many known results, especially in algorithms that take power usage into account. It is important to study algorithms dependent on peak power, as this is a real physical constraint that has become more important in recent years. Peak power usage is a critical limitation as the number of cores continues to increase in computers.

2.4 Related Work

2.4.1 On-Chip Optics and the Importance of Energy Efficiency

Presently, much research is focused on achieving exascale computing [37], and a dominant obstacle is energy usage. One approach to overcoming power efficiency challenges has been adding optical connections or lasers onto processors for faster and more energy-efficient communication. In Liu et al. [43], researchers have demonstrated laser connections on silicon operating at room temperature. With the promise of computers with optical connections in the future, developing efficient algorithms for such parallel machines will

help carry this technology further.

The use of lasers on chips for communication is an active and popular area of research [15, 29, 35, 39, 52, 74]. In general, using optical connections in hardware has been motivated by several advantages. Optical connections can link locations far apart and transfer data quickly between them since light is fast relative to electrical connections and suffers less attenuation. In addition, transferring a signal over optics consumes less power than transferring a signal over standard electrical wires. Optics scale to long distances while taking nearly the same energy as for shorter distances [58] and are being studied as a means to reduce power usage [51].

Though the mesh with optical connections model is directly motivated by this forthcoming interconnection technology, the analysis here is purely theoretical. Actual implementation is a separate area of research. For example, it may be the case that shorter length optics in the model are more efficiently implemented with wire connections.

In terms of related work in the energy efficiency of algorithms, there is an area of research that studies low-energy algorithms on distributed networks like wireless sensor networks [2, 60]. There is not any known previous work on energy-efficient algorithms on such a structured distributed network as the mesh other than [65].

2.4.2 Interconnection Layouts

Since the 1970's, there has been a large body of work studying different interconnection networks on chips. Leighton's book [40] contains a good overview of results including analysis of different layouts such as the shuffle-exchange graph and the mesh of trees. While interconnection networks similar to the layouts analyzed in this dissertation have been studied in the past, no previous work considered the constraints of energy consumption since energy efficiency has only recently become a significant issue.

2.4.3 The Mesh-Connected Computer

Since the model being studied in this dissertation is an extension of the mesh-connected computer model, many known results about the mesh are used. The mesh has been extensively studied in the past, and Miller and Stout [49] give a good summary and references for mesh algorithms. It will often be the case that a mesh algorithm is run as a subroutine in algorithms in the mesh with optical connections.

2.4.3.1 Lower Bounds

In the mesh model, there are two fundamental properties that provide lower bounds. The first is communication diameter, which is relevant in situations when data needs to be moved from one side of the mesh to the other. More generally, the diameter of a graph is defined to be the maximum distance between any pair of vertices, where the distance between vertices is the shortest path between them. On a standard mesh of size N, the diameter is $\Theta(\sqrt{N})$. If a problem requires global communication, that is, at least one processor may receive information originated by any processor, then any algorithm that solves the problem takes $\Omega(\sqrt{N})$ time. The other property that can be used to determine lower bounds is the bisection bandwidth. There are often problems that require movement of data where the running time is bounded below by the number of wires there are to transfer the data, for example, the problem of permuting N items on a mesh of size N, where all the data in columns 0 through $\frac{n}{2} - 1$ may need to be moved to columns $\frac{n}{2}$ through $n - 1$. The bisection bandwidth between these two halves of the mesh, which is the number of wires connecting the two halves, is $\sqrt{N}$. Since $\Theta(N)$ data must be moved over these wires, any algorithm that does this must have a running time of $\Omega(\sqrt{N})$.

In terms of energy usage, a simple bound of $\Omega(N)$ energy is required for anything that requires examining the whole input or accessing something from each processor at least once. For problems such as sorting and permutation, $\Omega(N^{3/2})$ total energy is required since simple matrix transposition results in items moving a total distance of $\Theta(N^{3/2})$. Mesh algorithms achieving these lower bounds have been widely known and refined since the 1970's [27, 38, 44, 49, 50, 55–57, 68]. In addition, any lower bound for a serial algorithm gives a lower bound for energy usage on a parallel computer as a serial computer can simulate a parallel computer.

CHAPTER 3

Definitions and Fundamental Properties

3.1 Definitions of Operations

While we denote the available power by S, it is often more intuitive to express running times in terms of the fraction of processors active in order to better see the tradeoff relationship between time and power. We denote the fraction of processors that are active by r, with $\frac{1}{N} \leq r \leq 1$, therefore $S = rN$.

3.1.1 Basic Operations

On a standard mesh, the following operations take $\Theta(\frac{1}{r} + \sqrt{N})$ time using rN peak power. The lower bound comes from the communication diameter and evenly dividing the peak power of the optimal algorithm, and it is easy to achieve this bound. With optics, we will see that a faster running time can be achieved.

Broadcast. To *broadcast* a value stored in a processor is to send it to all other processors in the mesh.

Semigroup Operations. Let value $a_{i,j}$ be initially stored in processor $P(i,j)$, for $0 \leq i,j \leq n-1$. A *semigroup operation* $\otimes$ is an associative operation. A *row reduction* with respect to $\otimes$ is an operation that determines the result of applying $\otimes$ over all values in a row, for every row. Formally, every $P(i,j)$ determines $a_{i,0} \otimes a_{i,1} \otimes \cdots \otimes a_{i,n-1}$. Similarly, a *column reduction* determines the result of applying a semigroup operation over all values in a column, that is, $P(i,j)$ determines $a_{0,j} \otimes a_{1,j} \otimes \cdots \otimes a_{n-1,j}$. A *global reduction* with respect to a semigroup operation $\otimes$ is the result of applying $\otimes$ over all N values. Since $\otimes$ is an associative operation, we must define an order of values in which $\otimes$ is applied. One way is to order values a_i, for $0 \leq i \leq N-1$, so that processor $P(i,j)$ has value a_{ni+j}, which is the row-major indexing scheme. A global reduction determines $a_0 \otimes a_1 \otimes \cdots \otimes a_{N-1}$. We assume that $\otimes$ can be computed in constant time.

Scan. A *row scan* is an operation where each processor $P(i,j)$, initially storing value $a_{i,j}$, determines $a_{i,0} \otimes a_{i,1} \otimes \cdots \otimes a_{i,j}$,where $\otimes$ is an associative binary operator. Similarly, a *column scan* is when processor $P(i,j)$ determines $a_{0,j} \otimes a_{1,j} \otimes \cdots \otimes a_{i,j}$. If we let the processors be labeled in row-major order, P_i, for $0 \leq i \leq N-1$, each initially containing a_i, a *global scan* results in each processor P_i, knowing the i^{th} prefix $a_0 \otimes a_1 \otimes \cdots \otimes a_i$. Again, we assume that $\otimes$ can be computed in constant time.

3.1.2 Permutation and Sorting

Permutation and sorting are fundamental operations which require extensive communication, and are often used as a key step in many parallel algorithms. Some key time and energy lower bounds are given in Section 3.2.

Permutation. Given N items on a mesh, each with its own unique destination processor, a permutation operation moves each item to its destination processor.

Sorting. Given N comparable items on a mesh, stored one per processor, a sorting operation moves the items into sorted order, one per processor, in a specified ordering such as row-major ordering or snake-like ordering.

3.2 General Lower Bounds for the Mesh with Optics

A benefit of optical connections on meshes is the ability to decrease the diameter of the mesh, effectively lowering the communication lower bound and reducing the total energy required to move data across the mesh. This capability gives the potential for achieving significant reductions in time and energy for problems on the mesh. Unfortunately, optics, by the fact that each has a minimal fixed width, do not change the bisection bandwidth lower bound. For problems where bandwidth is the dominant limiting factor, optics will not be able to improve the absolute minimum running time, but they may be able to reduce running times in limited peak power situations. This uses the fact that optics can move a limited amount of data large distances across the mesh at a time.

As will be shown in subsequent chapters, adding optical connections allows sorting and permutation to be solved in $o(N^{3/2})$ total energy, which is below the lower bound for permutation in a standard mesh.

In addition to the bisection bandwidth, there are some important lower bounds on meshes with at most a constant ℓ layers of optical connections. These include the following.

Diameter: no matter how optics are added, the diameter of the mesh is $\Omega(\log N)$. Since

each processor is adjacent to only a constant number of processors, the number of reachable processors is at most exponential in the number of time steps.

Permutation Energy: no matter how optics are added, the total energy to permute is $\Omega(N \log N)$. To see this, note that the comments above about the number of reachable processors shows that if processors are connected to at most t others, then in $q = (\log_t N)/2 - 1$ steps, no processor can communicate with more than $\sum_{i=0}^{q} t^i < t^{q+1} = \sqrt{N}$ processors. Given an optical layout, construct the following permutation: go through the processors in row-major order. For each processor, set its destination to be the first processor which is not reachable in q or fewer steps and which is not already the destination of some other processor. It is possible that the final $\sqrt{N}$ processors have no such destination, in which case choose the destination arbitrarily from the processors that are not yet destinations. Thus $\Omega(N)$ processors are sending to a destination requiring $\Omega(\log N)$ steps. Note this also shows that most permutations require $\Omega(N \log N)$ total energy.

Sorting Energy: there is the obvious $\Omega(N \log N)$ lower bound on energy due to the number of comparisons required. This is also a lower bound on the energy needed for data movement. This follows from the permutation bound since a permutation can always be achieved by sorting, using the destination as the key.

Note that while all parallel computers have the $\Omega(N \log N)$ lower bound for sorting comparisons, they do not all share this lower bound for permutation. In standard models of shared-memory machines, permutation takes only $\Theta(N)$ operations, while the above proof shows that for distributed-memory machines with bounded degree, the lower bound on data movement is $\Omega(N \log N)$.

Since we have an $\Omega(N \log N)$ energy lower bound for permutation and sorting on the mesh with optical connections, it follows that there is an $\Omega(\frac{1}{r} \log N)$ time lower bound for permutation and sorting when rN peak power is available. Note that, for a problem such as comparison-based sorting, the lower bound comes from the serial time lower bound from the number of required comparisons. If it were possible to sort faster than $\Omega(\frac{1}{r} \log N)$ time, a serial algorithm could simulate this faster algorithm and sort faster than $\Omega(N \log N)$ time.

Observation 3.1. *Given a mesh of size N with optical connections and rN peak power, permuting N items and sorting N items requires* $\Omega(\frac{1}{r} \log N)$ *time.*

In addition, any problem that requires data at each processor be examined takes $\Omega(\frac{1}{r})$ time.

3.2.1 Stepwise Simulation of Mesh Algorithms

Given any algorithm using a certain amount of power and time, an algorithm using less power and a linearly proportional increase in time can be created by stepwise simulation, provided that it can be determined at what times processors are active during the execution of the original algorithm. We should point out that there are a few technicalities about the implementation of stepwise simulation of algorithms in practice. The first is that processors must know at what point in time they must wake up from their sleep state and start computation. If it is known before the start of the execution of the algorithm when processors need to be active, then an internal timer in each processor can be used to trigger it between sleep and active states when it is needed during the execution. If it is not known beforehand when processors must be active, we assume that processors can be woken up by being sent a message from another connected processor. Technically, when there is communication between two processors, both must be active simultaneously, but in our model we assume energy moves with the data being communicated and so communicating between two processors requires only one unit of power.

As an example of stepwise simulation, we saw in Section 2.4 that on a standard mesh of size N without optics, sorting takes $\Theta(\sqrt{N})$ time and $\Theta(N^{3/2})$ total energy with N peak power. Therefore, when $S = rN$ of the processors are active, this gives a $\Theta(N^{3/2}/S) = \Theta(\frac{1}{r}\sqrt{N})$ running time because the mesh algorithm can be simulated by using one active processor per $\sqrt{N/S} \times \sqrt{N/S}$ submesh with $\Theta(N/S)$ time overhead per step of the mesh algorithm. Note that whenever the original algorithm requires the mesh to be fully powered, it is known when processors are active since all of them are active all the time.

Stepwise simulation is a useful technique that will be used for sorting small subsets of items in the different layouts of optical connections in the following chapters.

Theorem 3.2. *On a standard mesh of size N, N items can be sorted in $\Theta(\frac{1}{r}\sqrt{N})$ time using rN peak power.* □

CHAPTER 4

One Layer of Fixed-Length Connections (Optical Mesh)

In this chapter, we consider a network where the optical connections are all of the same length. This is a good introductory model, and it is probably easiest to build optical connections all of the same length in hardware. The optimal layout arranges the optical connections in a mesh network, so this model will be called a mesh with an *optical mesh*.

Let $f(n) \leq n$ and assume that $n/f(n)$ is an integer. The asymptotic time bounds will still hold when $n/f(n)$ is not an integer, but a few minor details need to be taken care of. An $n/f(n) \times n/f(n)$ mesh consisting of only processors and their optical connections of length $f(n)$ is created on top of the wire mesh. Processors at locations (i, j), for $i, j \in \{0, f(n), 2f(n), \ldots, (n/f(n)-1)f(n)\}$, will be part of this $n/f(n) \times n/f(n)$ optical mesh (see Figure 4.1). This optical mesh allows faster data movement across the mesh, which decreases the original $\Omega(\sqrt{N})$ lower bound for data movement due to the diameter of the communication network. In this model, the communication diameter gives a lower bound of $\Omega(f(n) + \sqrt{N}/f(n))$ for any operation or algorithm that requires communication between any two arbitrary processors.

Another advantage of the optical mesh is that it can be used to execute mesh algorithms on $O((n/f(n))^2)$ data since there are $(n/f(n))^2$ processors that are part of the optical mesh. The optical mesh is most useful when it has to work with only a reduced amount of data.

With this mesh of optical connections, it is now possible to do basic mesh operations that do not require much movement, like broadcast and the semigroup operations of maximum or minimum, in $\Theta(f(n) + \sqrt{N}/f(n))$ time.

Also, note that $f(n) = n^{1/2}$ is optimal for these basic operations, taking $\Theta(N^{1/4})$ time. Within the class of optical meshes it achieves the optimal communication diameter of $\Theta(N^{1/4})$. However, as stated before, the bisection bandwidth is still $\Theta(N^{1/2})$.

The basic operations of broadcast, reduction, and scan will be described with optical connections of length $f(n)$, but for the rest of the algorithms, $f(n) = n^{1/2}$ will be assumed.

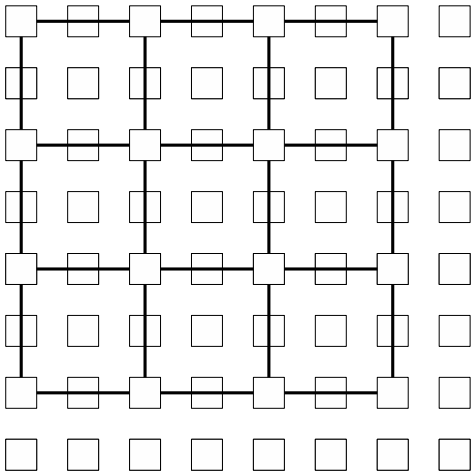


Figure 4.1: 8×8 optical mesh with optical connections of length 2.

4.1 Basic Operations

Given optical connections of length $f(n)$, where $f(n) \leq n$, the communication diameter gives a lower bound of $\Omega(f(n) + \sqrt{N}/f(n))$ time for any algorithm or operation that may require moving data from a processor to any other processor in the mesh. It will be shown that $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time using rN peak power can be achieved for broadcast, semigroup, and scan operations.

4.1.1 Broadcast

Sometimes it is useful to broadcast a collection of values located in some region of the mesh. A single processor does not have enough memory to store more than a constant number of values, so when a broadcast of more than a constant number of values occurs, every processor views every value that is broadcast but does not necessarily store all the values.

Proposition 4.1. *$O(\min\{f(n), n/f(n)\})$ data stored in any $f(n) \times f(n)$ submesh of an $n \times n$ mesh with fixed-length optical connections of length $f(n)$ can be broadcast to all processors in the mesh in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time.*

Proof. Note that in any $f(n) \times f(n)$ submesh, there exists a processor P with optical connections. Therefore, using standard data movement operations in the $f(n) \times f(n)$ wire submesh, $O(\min\{f(n), n/f(n)\})$ data can be moved to P in $\Theta(f(n))$ time. Each time a single value arrives at P, a standard broadcast operation can be used on the $n/f(n) \times n/f(n)$ optical mesh to broadcast the value to all processors that have optical connections. The

mesh can be divided into $(n/f(n))^2$ disjoint submeshes of size $f(n)^2$ that each contain a processor with optical connections by just dividing the mesh into equal-sized submeshes. When each processor with optical connections sees data from the broadcast, it can forward the data to the processors in its $f(n) \times f(n)$ submesh by using a standard broadcast. Using pipelining, all processors can see all $O(\min\{f(n), n/f(n)\})$ data. Since the operations used on the wire mesh take $\Theta(f(n))$ time and the operations on the optical mesh take $\Theta(\sqrt{N}/f(n))$ time, the total time for this broadcast operation over the whole mesh is $\Theta(f(n) + \sqrt{N}/f(n))$. This can be accomplished with each processor being active for at most a constant number of time steps so this has $\Theta(N)$ total energy usage. The minimum time can be achieved with $\min\{N/f(n), \sqrt{N}f(n)\}$ peak power and linear slowdown gives $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time using rN peak power.

A specific case of this data movement operation is the broadcast of any single value from one processor to all processors and the report of a single value from any processor to a specific processor. On a regular mesh, this takes $\Theta(\frac{1}{r} + \sqrt{N})$ time, but on a mesh with optical connections, this takes $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time. This is easily generalized so that $\min\{f(n), n/f(n)\}$ data located anywhere on the mesh can be moved to any arbitrary location in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time.

Looking at how the data is moved, we can make another generalization and say that if we divide the mesh up into $n/f(n)$ strips of size $f(n) \times n/f(n)$ (or $n/f(n) \times f(n)$), we can broadcast $O(\min\{f(n), n/f(n)\})$ data stored in each strip to every processor in the same strip in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time. This can be achieved due to the fact that each strip uses its own optical connections, disjoint from all the other optical connections that the other strips are using, so each strip can be seen as acting independently of the other strips. □

4.1.2 Semigroup Operations

Proposition 4.2. *The row and column reductions of an $n \times n$ mesh with fixed-length optical connections of length $f(n)$ can be computed in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time using rN peak power.*

Proof. To do a row reduction, the mesh is divided into $(n/f(n))^2$ submeshes, each of size $f(n)^2$. Within each submesh the semigroup operation applied to each row is found using the standard operations on just wire connections in $\Theta(f(n))$ time. Now using pipelining, these results can be moved to the optical mesh at the right time to produce the correct result. Once the result for each row is determined, a proper broadcast of the information is left to inform all processors.

In detail, at the end of the semigroup operation on the wire submeshes step, define $b_{i,k} = a_{i,kf(n)} \otimes a_{i,kf(n)+1} \otimes \cdots \otimes a_{i,(k+1)f(n)-1}$, for $0 \leq i \leq n-1$ and $0 \leq k \leq n/f(n)-1$, which are the values stored at processor $P(i,kf(n))$, and denote the time $t = 0$. Imagine each $n/f(n) \times n$ strip of processors doing the same thing in parallel. At time $t = 1$, each value stored at processor $P(s,0)$, for $s \in \{0, f(n), 2f(n), \ldots, n-f(n)\}$, is moved to processor $P(s,f(n))$ using the optical connection between them, which is combined using the associative operator with the value stored in that processor. Also, values stored in processors $P(s+j,0)$, for $1 \leq j \leq f(n)-1$, are moved up to processor $P(s+j-1,0)$. At time step t, for $0 \leq t \leq f(n)+n/f(n)-2$, the associative operation is applied to values located in the processors with optical connections in a pipelined manner. For $i+k \geq t$, $b_{i,k}$ is located at processor $P(f(n)-t+k,kf(n))$. For $i \leq t$, $b_{i,0} \otimes b_{i,1} \otimes \cdots \otimes b_{i,t-i}$, is located at processor $P(s,(t-i)f(n))$. Once $b_{i,0} \otimes \cdots \otimes b_{i,n/f(n)-1}$ reaches processor $P(s,n-f(n))$, the result keeps moving right in the mesh along the wire connections so as not to interfere with the new results arriving at that processor in future time steps. Note that the last values in the pipeline to be computed are $b_{s+f(n)-1,0} \otimes \cdots \otimes b_{s+f(n)-1,n/f(n)-1}$. Since the last value to reach $P(0,0)$ is $b_{f(n)-1,0}$, taking $f(n)-1$ time steps, and the time it takes over the optical connections is $n/f(n)-1$ time steps, the total number of steps to perform this part of the operation is $f(n)+n/f(n)-2$, which is in $\Theta(f(n)+\sqrt{N}/f(n))$. Linear energy usage and linear slowdown give $\Theta(\frac{1}{r}+f(n)+\sqrt{N}/f(n))$ time with rN peak power.

A column reduction uses the same movements as before except the mesh is rotated 90 degrees. This can be done due to the symmetry of the mesh and thus is also achieved in $\Theta(\frac{1}{r}+f(n)+\sqrt{N}/f(n))$ time. □

Proposition 4.3. *The global reduction of an $n \times n$ mesh with fixed-length optical connections of length $f(n)$ can be computed in $\Theta(\frac{1}{r}+f(n)+\sqrt{N}/f(n))$ time using rN peak power.*

Proof. This is easily achieved by finding the semigroup operation applied over each row as before, then finding the semigroup operation over each result of the n rows by applying the semigroup operation on all processors in a column. □

This operation can also be extended to meshes of higher dimensions by using a divide-and-conquer paradigm over the dimensions. Once slices of the mesh in lower dimensions are solved in parallel individually, the semigroup operation can be applied to the result of each slice to get the semigroup operation applied to the next higher dimension. Only n values need to be combined using semigroup operations going from a lower dimension to the next higher dimension, so the optics provide enough bandwidth to compute this in $\Theta(\frac{1}{r}+f(n)+\sqrt[d]{N}/f(n))$ time and rN peak power.

4.1.3 Scan

Proposition 4.4. *The global scan over N values stored in an $n \times n$ mesh with fixed-length optical connections of length $f(n)$ can be computed in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time using rN peak power.*

Proof. Processors are labeled in row-major order, where processor P_i initially contains a_i, for $0 \leq i \leq N-1$. This is accomplished with the following steps:

1. Divide the mesh into submeshes of size $f(n) \times f(n)$ and index strips of $f(n)$ columns $0 \leq s \leq n/f(n) - 1$. Solve each row in each submesh so that processor P_i in row j and strip s knows $a_{jn+sf(n)} \otimes a_{jn+sf(n)+1} \otimes \cdots \otimes a_i$.

2. Using movement similar to that in the semigroup operation, send information along each row using the optical connections so that processor P_i in row j knows $a_{jn} \otimes a_{jn+1} \otimes \cdots \otimes a_i$.

3. Move values such that processor P_{jn}, for $0 \leq j \leq n-1$, knows the value stored in processor $P_{(j+1)n-1}$, that is, $a_{jn} \otimes \cdots \otimes a_{(j+1)n-1}$.

4. Divide the mesh into strips of $f(n)$ rows and index the strips $0 \leq s \leq n/f(n) - 1$, and index each row in the strip $0 \leq k \leq f(n) - 1$. Move data down column 0 so that processor $P_{(k+sf(n))n}$ knows $a_{sf(n)n} \otimes a_{sf(n)n+1} \otimes \cdots \otimes a_{(k+sf(n))n-1}$.

5. Each processor $P_{(k+sf(n))n}$ broadcasts $a_{sf(n)n} \otimes \cdots \otimes a_{(k+sf(n))n-1}$ to all processors in its row so that each processor can update its value to $a_{sf(n)n} \otimes a_{sf(n)n+1} \otimes \cdots \otimes a_i$.

6. Move the value stored at processor $P_{(s+1)f(n)n-1}$, that is, $a_{sf(n)n} \otimes \cdots \otimes a_{(s+1)f(n)-1}$, to $P_{sf(n)n}$, which are the processors in column 0 that have optical connections.

7. Move data down column 0 using optical connections so that each processor with an optical connection knows $V_s = a_0 \otimes a_1 \otimes \cdots \otimes a_{sf(n)n-1}$.

8. Each processor $P_{sf(n)n}$ broadcasts V_s to all processors in rows $sf(n)$ to $(s+1)f(n)-1$ so that each processor P_i can update its value to $a_0 \otimes a_1 \otimes \cdots \otimes a_i$.

Steps 1 and 4 are mesh operations on a mesh of size $f(n)^2$, so they take $O(f(n))$ time. Step 7 is an operation on a mesh of size $(n/f(n))^2$, that is, the mesh consisting of optical connections, so it takes $O(\sqrt{N}/f(n))$ time. All the other steps are just movement and broadcast operations, which have already been shown to take $\Theta(f(n) + \sqrt{N}/f(n))$ time. This can also be accomplished with $\Theta(N)$ energy. Therefore, the total running time to compute all prefixes is $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ using rN peak power. □

Proposition 4.5. *The row and column scans of an $n \times n$ mesh with fixed-length optical connections of length $f(n)$ can be computed in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time using rN peak power.*

Proof. When a scan needs to be computed for only each row independently, like in the distance transform algorithm in Section 4.4, only steps 1 and 2 are needed and they determine the prefix for every processor in a row in $\Theta(\frac{1}{r} + f(n) + \sqrt{N}/f(n))$ time. □

This is a good illustration of how optical connections help send small amounts of information to solve a problem. We have this paradigm of combining the results of smaller problems to solve the bigger problem. To compute a scan, the problem is first solved for chunks of $f(n)$ processors, then for each row of n processors, then for $f(n)$ rows of processors, and then finally for all N processors in the mesh together.

This operation can be extended to higher dimensions as well by adding steps to solve slices in smaller dimensions individually and then using the optics to combine and broadcast results to all the slices and strips of the higher-dimensional mesh. This is a kind of multidimensional divide-and-conquer.

4.1.4 Sorting n Items

Given n comparable items, the following algorithm can be used to order them in sorted order. Note that the communication diameter gives an $\Omega(N^{1/4})$ lower bound.

Proposition 4.6. *n comparable items stored one per processor along the diagonal of an $n \times n$ mesh with fixed-length optical connections of length $n^{1/2}$ can be sorted in $\Theta(N^{1/4})$ time using $N^{3/4}$ peak power.*

Proof. Note that these n items just need to be spread out enough so that the bisection bandwidth of the network of just optics does not limit the movement. For example, the n elements could be stored in a row or column in the mesh. There will be enough bandwidth for items to reach the optics and enough bandwidth in the optical mesh as long as there are at most x items in every $x \times x$ submesh. Also, n items stored in any $\sqrt{n} \times \sqrt{n}$ submesh can also be sorted in $\Theta(N^{1/4})$ time by simply running a standard sorting algorithm on the items in a submesh of size n.

Each item has some comparable value. This value is broadcast to every processor in the row it is in. Then, each row i individually determines how many values in that column are greater than the value at (i,i) using a column reduction. Account for ties by looking at the original index of the value if there is a tie. This number is the position of the item in sorted

order. Now each item can be permuted to its desired destination in sorted order in $O(N^{1/4})$ time. □

4.2 Graph Component Labeling and Minimum Spanning Forest

The problem of finding a minimum spanning forest is well known in graph theory. Given the weighted edges of a graph, we want to find a spanning forest of these edges of minimum weight. The connected components of a graph can be determined from the minimum spanning forest of the graph because the sets of vertices contained in each tree in the resulting spanning forest partition the graph into its components. The applications of this problem are many and include areas like networking. Given a graph with n vertices and m edges, algorithms with various running times are known: between $\Omega(m)$ and $O(m \cdot \alpha(m,n))$, where α is a version of the inverse Ackermann function, for serial machines [53], $\Theta(\log n)$ on an EREW PRAM [19], and on a regular mesh, $\Theta(n)$ if the input is an adjacency matrix [7] and $\Theta(\sqrt{m})$ if the input is a list of edges [63].

Theorem 4.7. *Given the $n \times n$ weighted adjacency matrix of an undirected graph stored one element per processor on an $n \times n$ mesh with fixed-length optical connections of length $n^{1/2}$, the connected components and a minimum spanning forest can be determined in $O(\frac{1}{r}\log N + N^{1/4}\log N)$ time using rN peak power.*

Proof. Note that this is a linear time-power tradeoff for values of peak power up to $N^{3/4}$, and thus, it suffices to give an algorithm using $N^{3/4}$ peak power. The main idea here is that in each step of the minimum spanning forest algorithm, the amount of data being worked with is reduced to an amount that can be sped up by the use of the optical mesh.

Like most other parallel algorithms for finding the minimum spanning forest, the algorithm follows the idea of building a tree by merging smaller trees together in parallel. Each vertex has a label, which is the component it is in. Each processor knows the two vertices it connects and knows the labels of the vertices. At the start of the algorithm, each processor contains the weight of its edge, if it exists, and each vertex is labeled as its own index. The algorithm is a series of $O(\log n)$ stages, where the number of components in the graph is reduced by at least half each stage. At each stage, the smallest edge weight between two different components stored in each row is found and these n edges are moved to the n processors that have optical connections, one edge per processor. Now using only the optical connections, a minimum spanning forest of these n edges is found and vertices are labeled

to reflect the new connected components. Edges that were used in the minimum spanning tree are also marked. This repeats until there is only one component left.
Repeat $\lg n$ times:

1. For each component, find the edge of minimum weight that is incident to a vertex in the component and incident to a vertex not in the component. Call the set of minimum edges A.

2. Move A to the optical mesh.

3. Find minimum spanning forest of A.

4. Update vertex labels to new components.

Note that the size of A is $O(\sqrt{N})$ and the optical mesh consists of $\sqrt{N}$ processors as optics of length $n^{1/2}$ create an optical mesh of size $N^{1/4} \times N^{1/4}$.

Each of these four steps takes $\Theta(N^{1/4})$ time. Step 1 requires only the minimum weight edge stored in each column of the adjacency matrix, i.e., a column reduction. This basic operation on an optical mesh takes $\Theta(N^{1/4})$ time. Each block of $n^{1/2}$ columns contains at most $n^{1/2}$ edges of A that need to be moved to the optical mesh, which takes $\Theta(N^{1/4})$ time using the basic data movement operations. Step 3 uses the mesh algorithm from [49, 63], which finds a minimum spanning forest given a list of $O(N^{1/2})$ edges in $\Theta(N^{1/4})$ time using $N^{1/2}$ power. In step 4, the new component labels are moved from the optical mesh to a processor in each column. The new label is broadcast to each processor in the column. Similarly, new labels are broadcast along rows to update vertex labels. This requires only basic movement and broadcast operations that take $\Theta(N^{1/4})$ time. In addition, all the operations can be accomplished using $N^{3/4}$ peak power. □

Extending to three dimensions, if there are n^2 values of the adjacency matrix, then we will need n^2 processors in an $n^{2/3} \times n^{2/3} \times n^{2/3}$ mesh. In three dimensions, we will do the same as in two dimensions, that is, create a mesh within the mesh made out of only optical connections, but since a mesh in three dimensions has much more space to place optical connections that do not cross, we can actually fit multiple copies of the optical mesh in the mesh. In fact, it is possible to fit $f(n)$ copies of a three-dimensional optical mesh of size $(N^{1/3}/f(n))^3$ by just interleaving the optical meshes.

The layout for three dimensions can easily be extended for d dimensions when $d > 3$. $f(n)$ copies of an optical mesh with optical connections of length $f(n)$ can fit in a d-dimensional mesh of size n^d.

Theorem 4.8. *Given the $n \times n$ weighted adjacency matrix of an undirected graph stored*

one element per processor on an $n^{2/3} \times n^{2/3} \times n^{2/3}$ mesh with fixed-length optical connections of length $n^{1/3}$, the connected components and a minimum spanning forest can be determined in $O(\frac{1}{r}\log N + N^{1/6}\log N)$ time using rN peak power.

Proof. The $O(N^{1/6}\log N)$ time is achieved when $N^{5/6}$ power is available, so as in the two-dimensional an algorithm achieving this time will be presented and the linear tradeoff follows from stepwise simulation of the algorithm. Like in the two-dimensional case, the algorithm for finding the connected components and minimum spanning forest of a graph is iterative. The only additional important thing to describe is how the data must be initially distributed on the mesh. Let the $n \times n$ adjacency matrix of the graph be stored in the three-dimensional mesh in the following way: the matrix is divided into $n^{2/3}$ $n^{2/3} \times n^{2/3}$ subsquares, where each subsquare is located on a different layer of the mesh. The layers are stacked in the z direction in a row-major order. The first $n^{2/3}$ rows in the matrix are located in the first $N^{1/6}$ layers of the mesh and so on.

Row and column minimums can be found in $\Theta(N^{1/6})$ time and $N^{5/6}$ power. Even though we do not need to know how to find column minimums in the minimum spanning forest algorithm, the description of how the algorithm works is included. For row minimums, each block of $N^{1/6}$ layers in the mesh contains $n^{2/3}$ rows, so minimums of elements in the same row along the same z-coordinate are found first. After that, the problem is reduced to finding the row minimums of an $n^{2/3} \times n^{2/3}$ matrix, as finding the minimum along the z-axis puts the minimums of $n^{1/3}$ rows in the same layer of the mesh. Since there exists an $N^{1/6} \times N^{1/6}$ optical mesh on each layer, we know the minimums can be found in $\Theta(N^{1/6})$ time and $N^{1/2}$ power. For column minimums, each column is divided into $n^{1/3}$ parts spread out in the mesh, separated by a distance of $N^{1/6}$ in the z-axis. For each part stored in one layer of the mesh, the minimum can be found in $\Theta(N^{1/6})$ time using the optic column minimum algorithm, since there is an $N^{1/6} \times N^{1/6}$ optical mesh on each layer. Afterwards, the minimum of these minimums must be found over the layers that are spread $N^{1/6}$ apart. This is easily achieved in $\Theta(N^{1/6})$ time by simply using the optical connections along the z-axis. There are $N^{1/3}$ optical connections between any two layers spaced $N^{1/6}$ apart in the mesh, so there is enough bandwidth to transfer the $n^{1/3}$ minimums across the mesh. The minimums of each layer in the mesh can be found in parallel; thus, column minimum finding takes $\Theta(N^{1/6})$ time.

We see that step 1 takes $\Theta(N^{1/6})$ time. Step 2 takes $\Theta(N^{1/6})$ time because each processor is at most $N^{1/6}$ away from an optical mesh. Step 3 takes $\Theta(N^{1/6})$ time by [49]. Step 4 takes $\Theta(N^{1/6})$ time using basic movement and broadcasting operations. In addition, all four steps use $O(N^{5/6})$ peak power. □

The same algorithm can be extended to a d-dimensional hypercube mesh of side length $N^{1/d}$, given the $n \times n$ adjacency matrix with $n^2 = N$ and $d \geq 1$. The basic operations work in any dimension with the same amount of energy, as does the edge input minimum spanning forest algorithm described in [49], and thus each step of the algorithm takes $\Theta(N^{\frac{1}{2d}})$ time, for a total of $O(\frac{1}{r}\log N + N^{\frac{1}{2d}}\log N)$ time.

Finding the connected components and minimum spanning forest of a graph is often a key step in many other graph algorithms. Algorithms for problems such as finding biconnected components, bridge edges, and articulation points follow almost immediately. See [6, 7] for details of the algorithms.

Corollary 4.9. *Given the $n \times n$ adjacency matrix of an undirected graph stored one element per processor on an $n \times n$ mesh with fixed-length optical connections of length $n^{1/2}$, the biconnected components, bridge edges, and articulation points can be found in $O(\frac{1}{r}\log N + N^{1/4}\log N)$ time using rN peak power on a mesh with fixed-length optical connections.* □

In the case where the input graph is given as a list of edges, the optical mesh cannot help much with communicating many edges across the mesh quickly, but algorithms for component labeling and minimum spanning forest given a fixed power usage are a special case of the algorithms presented for the optical pyramid described in Section 6.7.

4.3 Image Component Labeling

An image of size N is an $n \times n$ image stored one pixel per processor, where each pixel has a color. White pixels are considered the background of an image, and the pixels that are not white form connected components. We say that two pixels are adjacent to each other if they share an edge, and two pixels are in the same component if there is a path of adjacent pixels of the same color between them. To determine the connected components of an image is to give each component a unique label.

Image component labeling is important in the field of computer vision and has many applications in areas such as video processing, biology, and medicine [45]. Parallel image component-labeling algorithms have been studied [3], and for an $n \times n$ image, the problem can be solved in $O(N)$ time on a serial machine [16, 17], $O(\log N)$ time on an EREW PRAM [20], and $\Theta(\sqrt{N})$ time on a regular mesh [49].

Theorem 4.10. *Given an $n \times n$ digital image stored one element per processor on an $n \times n$ mesh with fixed-length optical connections of length $n^{1/2}$, the connected components of an image can be determined in $O(\frac{1}{r}\log N + N^{1/4}\log N)$ time using rN peak power.*

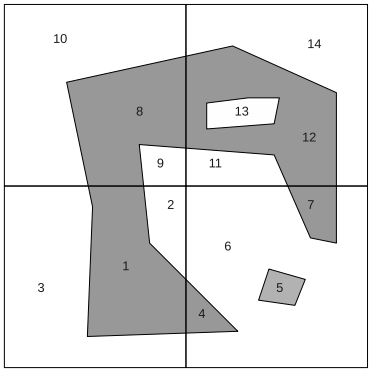


Figure 4.2: Labeling four subimages of an image.

Proof. Like with graph component labeling, the $O(N^{1/4}\log N)$ running time is achieved with $N^{3/4}$ power. This algorithm is a divide-and-conquer algorithm. To label the components of a square image, first divide the square into four subsquares and solve each subsquare individually with one-fourth of the available power in parallel. Now that each of these four subsquares is solved, the only figures that have incorrect labels in the whole square are those that cross the boundaries between the four subsquares. See Figure 4.2 for an illustration. The edges of figures that cross the borders are moved to a submesh and are labeled consistently using the edge input component-labeling algorithm [49]. Optics help in this algorithm because it exploits the fact that the amount of pertinent information is reduced from the area of the figure to the perimeter of the figure.

Consider a step of the algorithm where the subimage has side length m. In this $m \times m$ submesh, optics form an $m/\sqrt{n} \times m/\sqrt{n}$ optical mesh. We use the $m/\sqrt{n} \times m/\sqrt{n}$ optical mesh to simulate a $\sqrt{m} \times \sqrt{m}$ mesh, which will be the submesh the edge input component-labeling algorithm will be run on to label the $\Theta(\sqrt{m})$ edges. To do this, a $\sqrt{n/m} \times \sqrt{n/m}$ submesh located around each processor with optical connections simulates a $\sqrt{n/m} \times \sqrt{n/m}$ submesh of the $\sqrt{m} \times \sqrt{m}$ mesh. Adjacent submeshes are located around adjacent processors on the optical mesh. There is overhead when simulating communication between the borders of these submeshes, and at most $\Theta(\sqrt{n/m})$ data, which is the data on the border of the submesh, must be transferred to the processor with optical connections and over an optical connection to get to an adjacent submesh. This takes $O(\sqrt{n/m})$ time because $\Theta(\sqrt{n/m})$ data is being transferred over one optical connection. Since the edge input component-labeling algorithm takes $O(\sqrt{m})$ time on a $\sqrt{m} \times \sqrt{m}$ mesh and there is $O(\sqrt{n/m})$ overhead to simulate it on an $m/\sqrt{n} \times m/\sqrt{n}$ optical mesh, the total time is $O(\sqrt{m} \cdot \sqrt{n/m}) = O(\sqrt{n}) = O(N^{1/4})$. Therefore, each recursive step takes $O(N^{1/4})$ time. Once the size of the submesh is $\Theta(N^{1/2})$, the energy-efficient image component la-

beling algorithm of [65] is run with $\Theta(N^{1/4})$ power, which takes $\Theta(N^{1/4}\log N)$ time. As there are $\lg n - \lg n^{1/2} = \frac{1}{2}\lg n$ recursive steps, the total running time is $O(N^{1/4}\log N)$. □

We now look at the problem in three dimensions.

Theorem 4.11. *The connected components of an $n \times n$ image stored on an $n^{2/3} \times n^{2/3} \times n^{2/3}$ mesh with fixed-length optical connections of length $n^{1/3}$ can be labeled in $O(\frac{1}{r}\log\log N + N^{1/6}\log\log N)$ using rN peak power.*

Proof. The reason a faster speedup is achieved in dimensions greater than 2 is that there is much more room for optics, which allows for more data movement with optics.

Like before, we will divide the image into subimages and recursively merge the subimages together by computing the connected components of the border data of the subimages. The recursion is a series of $\lg\lg n$ stages. Index the stages i, for $0 \le i \le \lg\lg n - 1$. At each stage i, there is an invariant that says the connected components of each subimage of size $n^{1-1/2^i} \times n^{1-1/2^i}$ will have been solved. To do this, at each stage i, we merge $n/n^{1-1/2^i} = n^{1/2^i}$ of these $n^{1-1/2^i} \times n^{1-1/2^i}$ subimages by moving the border data of each subimage to an $n^{1/3} \times n^{1/3} \times n^{1/3}$ submesh and using the edge input component-labeling algorithm on just that wire submesh. As a result, this finds the connected components of the subimages merged together, which is a subimage of size $n^{1/2^i}(n^{1-1/2^i})^2 = n^{2-1/2^i}$ and dimensions $n^{1-1/2^{i+1}} \times n^{1-1/2^{i+1}}$. After stage $i = \lg\lg n - 1$ is done, the subimage side length will be $n^{1-1/2^{\lg\lg n}} = n/2$. At this point, there are only four subimages left to merge to get the whole image. The border data of these subimages is moved to the optical mesh and labeled consistently using the edge input component-labeling algorithm.

Note that we assume the image data stored in each submesh in the algorithm is a contiguous part of the whole image. In order for this to be the case, we need to specify how the image is initially stored in the mesh. We will recursively divide the image into rectangles so that each rectangle fits in a submesh. A cube can be divided into 8 smaller cubes, so when we divide the image, we alternate between cutting up the image into 8 vertical strips and cutting the image up into 8 horizontal strips. This keeps the part of the image being worked in a submesh contiguous and roughly square by a constant factor. Another option would be to cut the image up into 64 equal squares at each recursive step and divide the mesh into 64 cubes at each step, as 64 is both a square and a cube.

Since there are $O(N^{1/3})$ optical connections going in each $N^{1/6} \times N^{1/6} \times N^{1/6}$ submesh, it takes $O(N^{1/6})$ time to move $O(n)$ data from anywhere into an $N^{1/6} \times N^{1/6} \times N^{1/6}$ submesh. Therefore, the movement of the $O(n)$ border data to an $N^{1/6} \times N^{1/6} \times N^{1/6}$ submesh in each stage takes $O(N^{1/6})$ time. Finding the connected components of the border data in an $N^{1/6} \times N^{1/6} \times N^{1/6}$ submesh takes $O(N^{1/6})$ time [49]. Therefore, each stage takes

$O(N^{1/6})$ time. There are $\lg\lg n$ stages and each stage can be done in $O(N^{1/6})$ time with $N^{5/6}$ power, so the total running time is $O(\frac{1}{r}\log\log N + N^{1/6}\log\log N)$. □

The algorithm can be extended to d dimensions, for $d \geq 3$. The same algorithm used in the three-dimensional case is used, and the d-dimensional edge input image component-labeling algorithm is used in each recursive step. Using a d-dimensional mesh with a side length of $N^{1/d}$ with N processors, it is possible to do image component labeling on an $n \times n$ image in $O(N^{\frac{1}{2d}}\log\log N)$ time using $N^{1-\frac{1}{2d}}$ power, where implied constants depend on d.

4.4 Distance Transforms

For simplicity, we assume that images are composed of only black and white pixels. An object pixel is something in the foreground, that is, a black pixel. The ℓ_p metric is a distance measure where the ℓ_p distance between two points from (a,b) to (c,d) is $(|a-c|^p - |b-d|^p)^{1/p}$, for $1 \leq p < \infty$. The ℓ_∞ distance is defined as $\max\{|a-c|, |b-d|\}$. To find the distance transform of an image with respect to a metric is to find the closest object pixel for every pixel in the image.

Like image component labeling, finding distance transforms has applications in image processing and computer vision [23]. The distance transform of an $n \times n$ image can be determined in $\Theta(N)$ time on a serial machine [13], $O(1)$ time on a CRCW PRAM [76], and $\Theta(\sqrt{N})$ time on a mesh [49]. Here we only give an algorithm with the ℓ_1 metric. The algorithm for finding distance transforms according to the ℓ_∞ metric is similar.

Theorem 4.12. *Given an $n \times n$ digital image stored on an $n \times n$ mesh with fixed-length optical connections of length $n^{1/2}$, finding the object pixel closest to every pixel in the image according to the ℓ_1 metric can be determined in $\Theta(\frac{1}{r} + N^{1/4})$ time using rN peak power.*

Proof. The ℓ_1 distance between coordinates (x_1, y_1) and (x_2, y_2) is defined as $|x_1 - x_2| + |y_1 + y_2|$. There are two steps to the distance transform algorithm.

1. Every processor determines the closest object pixel in its row and stores the coordinates of that object pixel.

2. Every processor determines the closest object pixel by finding the closest object pixel stored in all the processors in its column.

Each of these two steps can be accomplished using two executions of a scan operation. For the first step, each processor finds the object pixel closest to the left and to the right of itself

(including itself) and stores the closest one of the two. To do this, each processor must store its position in the row and whether it contains an object pixel or not. Then define an operation that takes this information stored in two different processors and outputs the one that has an object pixel and a greater position. If neither contains an object pixel, it outputs either. We can do a row scan with this operation to find the closest object pixel to the left of every processor. Finding the closest object pixel to the right is similar. For the second step, we do a column scan, but this time, the operation has to compare the distance from the current processor to its closest object pixel so far to the closest pixel to the pixel above it. Each processor contains a record (p,d,r), where p is the pixel closest to the processor, initially the closest pixel in the processor's row, d is the distance from the processor to its current closest pixel, and r is the index of the row the processor is in. To find the closest pixel in the same row or higher, the associative operation is defined as

$$(p_0,d_0,r_0)\otimes(p_1,d_1,r_1)=\begin{cases}(p_0,r_1-r_0+d_0,r_1) & \text{if } r_1-r_0+d_0<d_1\\ (p_1,d_1,r_1) & \text{otherwise}\end{cases}.$$

Doing this in both directions will find the closest object pixel above the current pixel and below the current pixel.

We can prove that this works by assuming for the sake of contradiction that a processor P does not find the closest object pixel p. This would imply that there was an object pixel p' different from p that was closer to a processor P', the processor in the same column as P and in the same row as p. Let P'' be the processor in the same column as P and same row as p'. p' must be the object pixel closest to P'' by the ℓ_1 metric, which contradicts P' choosing p' as its closest object pixel. □

This can be extended to higher-dimensional images on higher-dimensional meshes. For example, to find the object pixel closest to every pixel in a three-dimensional image in a three-dimensional mesh with optical connections, first solve each two-dimensional slice of the image perpendicular to some axis individually in parallel and then merge the slices by doing a scan operation in the dimension the slices are stacked in. This uses a kind of dimensional divide-and-conquer algorithm, where solutions to a lower dimension of the problem can be used to solve a higher-dimensional version of the problem. Therefore, we can compute the ℓ_1 distance transform for a d-dimensional image of size $N=n^d$ on a d-dimensional mesh of size N with optical connections in $\Theta(\frac{1}{r}+N^{\frac{1}{2d}})$ time.

A convenient property about the ℓ_1 and ℓ_∞ metrics is that to find the pixel closest to any part of an image, it is enough to know only the object pixel closest to every pixel on the border of that part to compute the closest object pixels of the rest of the image. This implies that for any $m\times m$ part of an image, $O(m)$ data about the pixels on the border needs to be communicated to all the other processors in the mesh. This motivates the possibili-

ties of algorithms using optical connections, since only the square root of the information contained in any submesh is needed to be moved.

There is some evidence that it may be difficult for optical connections to help with ℓ_2, also called the Euclidean metric. If we want to find the Euclidean distance transformation of an image, for any set of processors Q that form a border around a processor P, there may exist a pixel that is closest to P but not closest to any processor in Q. Therefore, we do not have the property of needing only the information about the pixel closest to every processor on the border of a subimage. Thus, optics do not help when using the Euclidean metric. Consider an image on a mesh of size N, where the image is just a digitized circle of circumference $\Theta(\sqrt{N})$. It is possible that every pixel of the circle is the closest pixel to a pixel within an $N^{1/4} \times N^{1/4}$ subimage. Specifically, they would be the closest pixel to the pixel at the center of the circle. By a bandwidth argument, there are $\Theta(\sqrt{N})$ pixels in the image, each possibly needing its information sent within the $N^{1/4} \times N^{1/4}$ submesh, which has only a constant number of optical connections connecting it with the rest of the mesh. This means using optics cannot help, as it will take $\Omega(\sqrt{N})$ time to get the information about the pixels into the $N^{1/4} \times N^{1/4}$ submesh. However, it may be possible to get an approximate answer, such as the nearest pixel within some error. It is an open problem as to how well the Euclidean distance transform can be approximated in $o(\sqrt{N})$ time.

4.5 Sorting

In this section we briefly examine algorithms for permutation and sorting. While optics cannot improve the running time, energy consumption can be improved. First, we look at a sorting algorithm in the simple case of a one-dimensional mesh. The ideas used here will give a sense of the more complex algorithms for sorting on meshes of higher dimensions.

In two dimensions, it is possible to show that the optical mesh with optics of length $n^{1/2}$ achieves the optimal running time for sorting using peak power $O(\sqrt{N})$ out of all possible layouts of optics in one layer. In the case of $\omega(\sqrt{N})$ available peak power, shorter optical connections can be used to achieve the optimal running time for sorting over any possible layout of fixed length optical connections. The details are omitted here, but algorithms for the optical pyramid in Chapter 6 can be easily modified and applied to the optical mesh in situations with $\omega(\sqrt{N})$ power and optics of shorter length. Chapter 6 also shows optimality of an optical network with variable-length connections for sorting.

4.5.1 Sorting on a One-Dimensional Mesh with One Layer of Optics

In this section, we present a result about energy usage for sorting on a one-dimensional mesh and a proof that the optical layout used is optimal. The following two theorems show that, given a one-dimensional mesh of $N = n^2$ processors, the optimal way to add one layer of optical connections is to connect processor $i \cdot n$ with processor $(i+1)n$ for $i \in \{0, \ldots, n-2\}$, that is, connect processors with optics of length n.

Theorem 4.13. *Given N items on a one-dimensional mesh of N processors with optical connections connecting processor $i \cdot n$ with processor $(i+1)n$ for $0 \leq i \leq n-2$, they can be sorted in $O(N)$ time and $O(N^{3/2})$ energy using $\sqrt{N}$ peak power.*

Proof. We first need an algorithm to sort $n^{3/2}$ items located in a contiguous block of processors in $O(n^{3/2})$ time and $O(n^{5/2})$ energy. To do this:

1. Divide into $\sqrt{n}$ smaller blocks of size n, and sort each block one at a time with a standard one-dimensional sort such as bubble sort.

2. Let the set of pivots be the set comprised of the items at every $\sqrt{n}^{\text{th}}$ processor in the full block. For each pivot, broadcast its value to all items and then perform a reduction to determine how many items have a value less than it. Then, move each pivot to its correct destination.

3. Broadcast each pivot value and its location to each item so that each item knows between which pivots it belongs. Redistribute items so that they end up between the proper pivots.

4. Sort each block of items between pivots.

The standard one-dimensional sort on n items takes $\Theta(n)$ time and $\Theta(n^2)$ energy [65], so the first step uses $\Theta(n^{5/2})$ energy. In the second step, a broadcast and reduction is performed for each pivot, which takes $\Theta(n^{5/2})$ energy. Pipelining these operations takes $O(n)$ time. The third step also takes $\Theta(n^{5/2})$ energy since there is a broadcast of each pivot and redistributing the remaining $n^{3/2} - n$ using optics takes $O(n)$ energy per item. The broadcasts are pipelined in $O(n)$ time and redistributing the other items takes $O(n^{3/2})$ time. Note that the number of items between pivots is at most $n - \sqrt{n}$ because in the worst case, $\sqrt{n} - 1$ items in each of initial $\sqrt{n}$ blocks could fall between two pivots. Therefore, the last step takes $O(n^{5/2})$ energy and $O(n)$ time.

Now, here is an algorithm to sort N items:

1. Divide the processors into $\sqrt{n}$ contiguous blocks of size $n^{3/2}$, and sort each block using the preceding algorithm.

2. Let each item at a processor with optical connections, that is, each n^{th} item, be a pivot. For each pivot, determine its final destination and move it there.

3. Redistribute items so that they end up between the proper pivots.

4. Sort items between pivots using the preceding sorting $n^{3/2}$ items algorithm.

Note that the number of items between successive pivots is at most $n^{3/2} - \sqrt{n}$, as there are $\sqrt{n}$ blocks and a pivot at every n^{th} item within a block.

It is clear that all the steps take $O(N)$ time. Step 2 can easily be accomplished by first broadcasting each pivot so that each item in the mesh can determine which two pivots it lies between. (It could also be the case that the item lies before the first pivot or after the last pivot, in which case only the one closest pivot is recorded.) Then a reduction operation is executed for each pivot so that each pivot knows how many items are less than it. This is exactly the final destination of the pivot. In step 3, this information is then broadcast to each item so that each item knows a block where its destination lies delimited by the pivots it falls between. Finally, each item moves in a pipelined manner to an available location between the pivots where its destination lies.

Step 1 uses $O(N^{3/2})$ energy because the $O(n^{5/2}) = O(N^{5/4})$ energy algorithm is used $N^{1/4}$ times. Step 2 uses $O(N^{3/2})$ energy. Step 3 can be accomplished with $O(N^{3/2})$ energy by using the optical connections to move items between the pivots. Step 4 also uses $O(N^{3/2})$ energy. Therefore, this algorithm uses $O(N^{3/2})$ energy. In step 2, n pivots are chosen because partitioning with n pivots takes $O(N^{3/2})$ energy. □

The following theorem shows that even if optics of different lengths are allowed, sorting takes $\Omega(N^{3/2})$ total energy. Since the bisection bandwidth shows that sorting takes $\Omega(N)$ time, our layout is optimal in terms of time and energy.

Theorem 4.14. *Given a one-dimensional mesh of size N with one layer of optical connections, a permutation of N items stored one per processor requires $\Omega(N^{3/2})$ total energy.*

Proof. For the sake of contradiction, assume permutation can be accomplished with less than $N^{3/2}/18$ total energy. Processors with optical connections divide the mesh into sections of processors with no optical connections. A contiguous block of processors between processors at the endpoints of an optical connection is called an *optic section*, and the size of an optic section is the number of processors in the optic section. Without loss of generality, the mesh is partitioned into optic sections.

This implies the following two statements:

- At least $N/2$ items are within optic sections of size less than $\sqrt{N}$.

  Because otherwise, at least $N/2$ items are in optic sections of size greater than or equal to $\sqrt{N}$. This implies there are at least $\sqrt{N}/2$ of these optic sections. It is possible to construct a permutation in which all items need to move out of their initial optic sections. The total distance these $N/2$ items must move is minimized if all long sections are length $\sqrt{N}$, giving $\sqrt{N}/2$ sections. In a section, the total distance items move to get to the optical endpoints is minimized if half go the left and half to the right. Thus the total distance over all sections is $\sqrt{N}/2(N/2+\sqrt{N}/2) = N^{3/2}/4+N/4$.

- Less than $N/2$ items are within optic sections of size less than $\sqrt{N}$.

  Otherwise, at least $N/2$ items are in optic sections of size less than $\sqrt{N}$. Consider only the processors and connections in sections of size less than $\sqrt{N}$ and divide them into thirds. There are at least $N/6$ processors in the middle third. Therefore, at least $\sqrt{N}/6$ optic sections of size less than $\sqrt{N}$ are in the middle third. It is possible to construct a permutation in which all items in an outer third need to move across the middle third to the other side. Moving the $N/3$ items in the outer thirds across the at least $\sqrt{N}/6$ optic sections in the middle third takes at least $N^{3/2}/18$ energy.

These two statements cannot both be true. □

Note that the optimal layout with optical connections of length n also gives an optimal communication diameter of $\Theta(\sqrt{N})$.

4.5.2 Sorting on a Two-Dimensional Mesh with Fixed-Length Optical Connections

In two dimensions, it is not possible to sort faster than the $\Theta(\sqrt{N})$ time to sort on a standard mesh without optics. However, in some limited peak power situations, the addition of fixed-length optics can improve the running time over the standard mesh. The algorithms are more complex than the one-dimensional case and they are deferred to Chapter 6 due to the sorting algorithm for fixed-length optics being almost a special case of the sorting algorithm for the optical pyramid. The following lemma and theorem are analogous to Lemma 6.6 and Theorem 6.7 for the optical pyramid algorithms described in Section 6.5.

Lemma 4.15. *On a mesh of size N with fixed-length optical connections of length $n^{1/2}$, N items can be permuted in $\Theta(\frac{1}{r}N^{1/4}+N^{3/4})$ time using rN peak power.* □

Theorem 4.16. *On a mesh of size N with fixed-length optical connections of length $n^{1/2}$, N items can be sorted in $\Theta(\frac{1}{r}N^{1/4}+N^{3/4})$ time using rN peak power.* □

CHAPTER 5

Two Layers of Variable-Length Connections (Optical Mesh of Trees)

It is natural to consider what happens if more than just one noncrossing layer of optics is added to the mesh. In fact, two layers of optics is enough to simulate any constant number of layers of optics because all the vertically aligned optics can be placed in one layer and all the horizontally aligned optics can be placed in the second layer. Making sure that the optical connections are sufficiently spread out so that there are no overlaps, it is evident that there are no crossings in either layer. In addition, layouts with optics that are not parallel to an axis can be simulated by a layout with axis-aligned optics where optics are decomposed into a constant number of vertical connections and a horizontal connections. This stepwise simulation introduces only a constant factor overhead to the running time.

In the following layout, algorithms often achieve faster running times than the standard mesh, even with peak power restrictions. We will define the layout for a mesh with two optical layers as a mesh-of-trees network [40] except that the leaves of the trees are spaced $\lg n$ apart. Despite this, it will be called the *optical mesh of trees*. The layouts of the two layers are exactly the same except one layer is the other layer rotated 90 degrees.

First divide the mesh into $\lg n \times n$ blocks, which are groups of $\lg n$ rows. Label the blocks A_j, for $0 \leq j \leq \left\lfloor \frac{n}{\lg n} \right\rfloor - 1$. Now, for each row i in A_j, for $1 \leq i \leq \lg n - 1$, and for each $0 \leq k \leq \frac{n}{2^{i+1}} - 1$, connect processors $P(i, k \cdot 2^{i+1})$ and $P(i, k \cdot 2^{i+1} + 2^i)$ with an optical connection. Therefore, row i within the block A_j contains optical connections of length 2^i. Similarly, we label the $n \times \lg n$ blocks B_j, for $0 \leq j \leq \left\lfloor \frac{n}{\lg n} \right\rfloor - 1$. Figure 5.1 illustrates one layer, with the other being a 90-degree rotation.

Note that the communication diameter in each block is $\Theta(\log N)$ because the optics can simulate a balanced binary tree of height $\lg n$ and every processor is within $\lg n$ steps of a leaf of the tree.

We know that the communication diameter of the mesh with optics is $\Omega(\log N)$. The mesh of trees layout shows that the communication diameter is also $O(\log N)$ because each

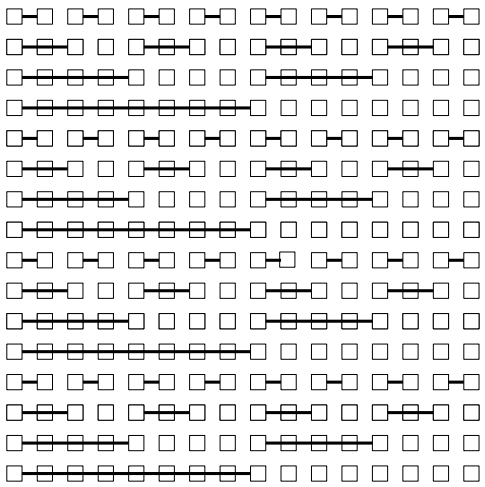


Figure 5.1: One of two layers of an optical mesh of trees on a 16×16 mesh.

processor can reach any processor in $O(\log N)$ steps by first traveling using the tree network in its row to the submesh B_j in which its destination lies in $O(\log N)$ time and then using the tree network in B_j to reach its destination in $O(\log N)$ time. Therefore, the communication diameter of this network is $\Theta(\log N)$.

The communication diameter gives a lower bound of $\Omega(\log N)$ for any operation that requires global communication. We will also see that the optical mesh of trees layout gives the optimal running time of $\Theta(\frac{N \log N}{S})$ for sorting when peak power S is $O(\sqrt{N})$.

5.1 Basic Operations

In this section, detailed descriptions of the execution of basic operations are given. The operations of broadcast, reduction, and scan can be accomplished with $O(N \log N)$ energy in the optimal $\Theta(\log N)$ time, giving a $\Theta(\frac{1}{r} \log N)$ running time using peak power rN.

5.1.1 Broadcast

Since we showed in Section 3.2 that there is a lower bound of $\Omega(\log N)$ for the communication diameter, there is a lower bound of $\Omega(\log N)$ time to do a broadcast. We can achieve this running time on a mesh with an optical mesh of trees.

Proposition 5.1. *$O(\log N)$ data stored in any $\lg n \times \lg n$ submesh of an $n \times n$ mesh with an optical mesh of trees can be broadcast to all processors in the mesh in $O(\frac{1}{r} \log N)$ time using rN peak power.*

Proof. We will first look at the movement of data between any two processors in a $\lg n \times n$ or $n \times \lg n$ submesh. These strips of $n \lg n$ processors all have identical optical connections, so the following movement algorithms will describe them in the first $\lg n$ rows, that is, rows 0 to $\lg n - 1$. It is easy to extend the movements to other strips by adding multiples of $\lg n$ to the column coordinate or rotating the algorithm 90 degrees.

Suppose processor $P(0,0)$ wants to send data to processor $P(0,j)$. To accomplish this, we look at the binary representation of j to see which optical connections to use when transmitting this data. Examining the most significant bit to the least, if bit a of j is a 1, then the optical connection in row a within the $\lg n \times n$ strip is used. The following pseudocode describes the movement from processor to processor. Denote $|j|$ as the number of bits in the binary representation of j and $(j)_a$ as the bit at index a of j.

```
1:  i' ← 0, j' ← 0
2:  for a = |j| − 1 to 0 do
3:      if (j)_a = 1 then
4:          j' ← a
5:          MoveUsingWiresTo(i', j')
6:          i' ← i' + 2^a
7:          MoveUsingOpticsTo(i', j')
8:      end if
9:  end for
10: MoveUsingWiresTo(0, j)
```

In the preceding pseudocode, the variables i' and j' keep track of the coordinates at which the data is located and a denotes the index of the current bit of j we are looking at. The body of the loop is executed only when bit a is 1, so in effect, we move along the rows using only optical connections corresponding to the 1 bits. It is easy to see that this movement takes $O(\log N)$ time. The movement using wires at lines 5 and 10 is the only movement along columns that is performed and takes $2(|j| - 1) = 2\lfloor \lg j \rfloor$ steps total because the first iteration of the loop moves the data from row 0 to row $|j| - 1$ and the remaining movements along the columns move the data back to row 0. The only other movement operation at line 7 moves data over only one optical connection, which takes one time step each time it is executed. As line 7 is executed at most $|j|$ times, that is, the number of iterations of the loop, it takes at most $\lfloor \lg j \rfloor + 1$ steps. Thus, we see this whole movement operation takes $O(\log N)$ time.

Now suppose processor $P(i,0)$ wants to send data to processor $P(i',j)$ with $i \leq \lg n - 1$ and $i' \leq \lg n - 1$. We first move the data using wires from $P(i,0)$ to $P(0,0)$, then use the previous movement algorithm to move from $P(0,0)$ to $P(0,j)$, then move using wires from

$P(0,j)$ to (i',j). This also takes $O(\log N)$ time, as the additional movements on wires take at most $2(\lg n-1)$ steps.

If the source processor is not in the first column, we first look at the problem when processor $P(0,j)$ wants to send data to processor $P(0,j')$, where $j<j'$ and there is an a, $a \leq |j'|$, where j is equal to j' with the a least significant bits set to 0. The movement of this operation is the original movement operation generalized to work on any section of the strip where the optical connections are in the same layout as the ones that will be used. The same exact movement as from $(0,0)$ to $(j'-j,0)$ is done except shifted over by j. Note that the optical connections are in the same relative position, as the movement uses only rows 0 to $|j'-j|-1 \leq a-1$. By definition of how the optical connections are laid out, the pattern of optical connections in rows 0 to $a-1$ repeats in the mesh every 2^a columns. The a least significant bits of j are 0, so j is a multiple of 2^a. Thus, it is possible to use the same movement along the optical connections as if it started at $P(0,0)$, except that it is shifted over j columns, and in $O(\log N)$ time as before. By symmetry, this movement operation can be mirrored so that data can be moved from $P(0,j)$ to $P(0,j')$, where $j>j'$ and there is an a, $a \leq |j'|$, where j is equal to j' with the a least significant bits set to 1.

Generalizing the movement more, suppose processor $P(0,j)$ wants to send data to processor $P(0,j')$, for any j and j'. First assume that $j<j'$. To do this, find the largest a such that $(j)_a \neq (j')_a$, that is, the most significant bit that differs between j and j'. Let j'' be equal to j' with the a least significant bits set to 0. To move from $P(0,j)$ to $P(0,j')$, we first move data from $P(0,j)$ to $P(0,j'')$ using the movement operation defined previously and then move data from $P(0,j'')$ to $P(0,j')$ using the movement operation defined previously. This takes $O(\log N)$ total time. If $j>j'$, we just use the same path that the data takes to move from $P(0,j)$ to $P(0,j')$, except backwards. Extending this as before to send information between any two processors $P(i,j)$ and $P(i',j')$ within a strip can also be done in $O(\log N)$ time.

We now know how to move information between any two processors in an $n \times \lg n$ or $\lg n \times n$ strip, so what remains is moving data between different strips. This is where we have to use both layers of optical connections. Let us look at how one processor sends information to another processor at an arbitrary position in the mesh. Suppose processor $P(i,j)$ wants to send information to processor $P(i',j')$. Imagining the mesh divided into $\lg n \times \lg n$ submeshes, the idea is to first move the data to the strip of $\lg n$ columns that contains the destination using optical connections, then move it to the destination within that strip of columns using the movement operation described earlier. This movement operation will be using only the optical connections along rows in the strip of $\lg n$ rows that contain $P(i,j)$ and the optical connections moving data along columns in the strip of $\lg n$

columns that contain $P(i', j')$. Each move in the two strips used takes $O(\log N)$ time, and thus the total time taken to move data between any two arbitrary processors in the mesh is $O(\log N)$. Pipelining data movements, we can also show that it is also possible to move $O(\log N)$ data stored in a submesh of size $O(\log^2 N)$ anywhere within the strip it is located in.

Now consider the operation of broadcasting a single value from a processor to all other processors in its row. Without loss of generality, we can assume that the value is stored initially at processor $P(0,0)$, since we can move any single value there in $O(\log N)$ time using the movement operations. The tree-like structure of the optical connections allows us to efficiently broadcast a single value to all processors in a row. We can imagine each optical connection as a node in a binary tree, the longest optical connection in row $\lg n - 1$ as the root node, with the children of each node being the two optical connections in the next row in the same columns of the mesh, and each of the positions in row 0 as leaf nodes. If we move the value to the processor at coordinates $(0, \lg n - 1)$, we can easily broadcast to all the processors in row 0 by a parallel traversal of the tree to all the children. The number of steps a broadcast takes is proportional to the number of steps to traverse a balanced binary tree with n leaf nodes in parallel, and thus it takes $O(\log N)$ time. If a single value is to be broadcast to all processors in a single strip, the value is first broadcast to all processors in a row, taking $O(\log N)$ time, and then each processor in that row broadcasts the value to the other $\lg n - 1$ processors in its column within the strip using the standard broadcast operation on a wire mesh in $O(\log N)$ time. If a value has to broadcast to all processors in the mesh, then first the value is broadcast to all processors in its column so that at least one processor in each $\lg n \times n$ strip knows the value. Then each $\lg n \times n$ strip has one of these processors within it to broadcast to all other processors in the strip in parallel. All of these operations take $O(\log N)$ time. In addition, peak power usage does not exceed $\frac{N}{\log N}$ so the running time is $O(\frac{1}{r} + \log N)$ using rN peak power.

In fact, it is possible to broadcast any $O(\log N)$ amount of data stored in any $\lg n \times \lg n$ submesh to all processors in $\Theta(\log N)$ time by pipelining the broadcast operation for each value being broadcast. We can see that the broadcast operation can be pipelined because no processor has to remember any information in a broadcast operation and values never visit a processor more than a constant number of times. The reason this has to be true is that if a processor had to store information about data it has seen, then there would not be enough memory to store this information for all $O(\log N)$ values, and since no value visits a processor more than a constant number of times, each processor can pass on all $O(\log N)$ values in $O(\log N)$ time. This adds up to a logarithmic factor in total energy usage since $O(\log N)$ more data needs to be moved. This gives the $O(\frac{1}{r} \log N)$ running time. □

Note that we can simulate a mesh with fixed-length optics with $O(\log N)$ overhead because each movement over optical connections in a mesh with fixed-length optical connections can be simulated in $\Theta(\log N)$ time.

5.1.2 Scan

Here we look at finding only the row scan. An algorithm for global scan follows easily.

Proposition 5.2. *The row and column scans of an $n \times n$ mesh with an optical mesh of trees can be computed in $\Theta(\frac{1}{r}\log N)$ time.*

Proof. We first need to describe how to find the prefix for each processor in a single row. We already know that movement along the optics in a strip can be directly mapped to traversal along the binary tree defined by the optical connections, so it will be enough to describe how to compute the scan of all the leaf nodes of the tree using tree traversals to compute the scan of all the elements in a row. Intuitively, this follows a standard parallel prefix algorithm where each subtree computes the operation applied to all of its values then receives the prefix of the previous values from its parent node.

Associated with each node, we define a variable *lefttree*, which will store some intermediate values during execution of the operation. Initially, *lefttree* of all leaf nodes in the tree is set to the value at the node. First, an operation similar to the semigroup operation over the row is performed. Data is moved up the tree from the children, and each node stores the data sent from the left subtree to *lefttree* and computes the semigroup operation applied to the values sent by its two children. The result is sent to its parent in the tree. Once the root node is reached, data is broadcast down the tree in the following manner. The root node sends nothing to its left child and *lefttree* to its right child. For all other nodes, if it receives nothing from its parent, then it sends nothing to its left child and *lefttree* to its right child. If a node receives a value a from its parent, it sends a to its left child and $a \otimes$ *lefttree* to its right child. The result for each leaf node is what the node would have sent to its right child, if it had a right child. By induction on the height of the tree, we can show that for any tree, the root knows the semigroup operation applied to all the values stored in the leaf nodes, and each subtree in the tree is given the semigroup operation applied to all nodes to the left of it in the tree, or nothing if there is nothing to the left. Thus, each leaf node receives all the information it needs. The base case of a single node works, and the information received from a parent node is always the semigroup operation applied to all nodes to the left in the tree, or nothing if it is the leftmost node in the tree. Note that movement of data up and down this tree can be done in parallel just like broadcast and semigroup operations, so the running time of a scan is $\Theta(\log N)$.

Now, consider the problem of computing the scan of each row. Label processors in the standard way as $P(i, j)$. It is not immediately clear that the scan operation can be efficiently pipelined since each node of the tree has to store *lefttree* for each row in the strip during the operation and there is not enough memory in a processor to hold $\lg n$ values in one processor. It turns out we will still be able to pipeline and compute the scan of all rows in $O(\log N)$ time. The key idea is that each node of the tree needs to store only $\lg n$ data while pipelining the scan operation. There are $n-1$ total nodes, which is a total of $(n-1)\lg n$ data that needs to be stored. Since a strip contains $n \lg n$ processors, there is enough space in this strip to store all this data. In order to store each *lefttree* value for each row in each node, each optical connection (which represents a node in the tree) is designated a column in which its values are stored. Since there are $\lg n$ processors in a column, the column has enough space to store the *lefttree* value of each row. Recall that in a row i within a strip, optical connections connect processors $P(i, k \cdot 2^{i+1})$ and $P(i, k \cdot 2^{i+1} + 2^i)$, for $0 \leq k \leq \frac{n}{2^{i+1}} - 1$. Each optical connection is assigned column $k \cdot 2^{i+1} + 2^i$ to store *lefttree*. We can see that each optical connection is assigned a unique column, because the column that is assigned is not used by any greater row, as no greater row has a processor with an optical connection in that strip in that column. So, at each step within the operation, every time a node of the tree has to store a *lefttree*, it rotates the values in its assigned column using standard mesh rotation operations to store the value. These values stored in the columns are rotated along the column and pipelined so that the processor will have the corresponding *lefttree* value of a row when it is needed in the algorithm. □

5.1.3 Semigroup Operations

Both the papers of Beame [10] and of Li and Yesha [41] show that any variant of the PRAM model with a polynomial number of processors in n requires time $\Omega(\log n)$ to compute operations such as the addition of n numbers. This gives us a lower bound of $\Omega(\log n)$ for computing the semigroup operation of values stored in a mesh, since a mesh can be simulated by a PRAM. It happens that there is an algorithm that can also achieve this lower bound.

Proposition 5.3. *The row, column, and global reductions of an $n \times n$ mesh with an optical mesh of trees can be computed in $\Theta(\frac{1}{r}\log N)$ time using rN peak power.*

Proof. This follows from combining Propositions 5.1 and 5.2. A scan operation computes a reduction on all the values in each row or column as part of the output. These values can be broadcast to each processor. To compute a global reduction, we can extend row and column reductions to the computation of the semigroup operation applied to all values stored in a

two-dimensional or higher-dimensional mesh. A column reduction can be computed using the resulting values of each row. This is exactly the semigroup operation applied to all values in a two-dimensional mesh. For higher dimensions, we continue this dimensional divide-and-conquer paradigm, where all possible slices of the mesh in lower dimensions are solved and then combined using a single semigroup operation over the resulting values along the dimension the slices are stacked in. These operations take $O(\frac{1}{r}\log N)$ time using rN peak power. □

5.1.4 Diagonal Broadcast and Diagonal Scan

Sometimes it may be useful to be able to compute information about data stored in diagonals of the mesh rather than in rows or columns. One application is computing distance transforms of images with respect to the ℓ_∞ metric. Let $\otimes$ be a semigroup operation and $a_{i,j}$ be entries of a matrix stored on the mesh. To compute a diagonal scan is for processor $P(i,j)$ to determine

$$\begin{cases} \bigotimes_{k=0}^{i} a_{k,j-i+k} & \text{if } i \le j \\ \bigotimes_{k=0}^{j} a_{i-j+k,k} & \text{if } i \ge j \end{cases}$$

If the scan is performed on the anti-diagonals, processor $P(i,j)$ determines

$$\begin{cases} \bigotimes_{k=0}^{j} a_{i+j-k,k} & \text{if } i+j \le n-1 \\ \bigotimes_{k=0}^{n-1-i} a_{n-1-k,j-(n-1-i)+k} & \text{if } i+j \ge n-1 \end{cases}$$

Scans performed in the reverse directions are also possible.

A diagonal broadcast is when a processor in each diagonal sends data to all other processors in its diagonal. For example, processor $P(i,0)$ with data $a_{i,0}$ sends it to processors $P(i+j,j)$ and processor $P(0,i)$ with data $a_{0,i}$ sends it to processors $P(j,i+j)$, for all i such that $0 \le i \le n-1$ and for all $0 \le j \le n-1-i$.

Proposition 5.4. *The diagonal broadcast on an $n \times n$ mesh with an optical mesh of trees can be computed in $O(\frac{1}{r}\log^2 N)$ time.*

Proof. The idea is to divide up the mesh into blocks of size $2\lg^2 n$, each containing a part of $2\lg n$ diagonals, and use the movement operation along optics to move $\log n$ data between

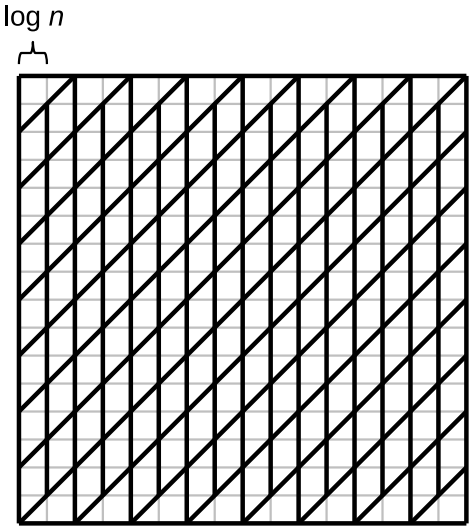


Figure 5.2: Dividing the mesh with anti-diagonals.

blocks in a way such that data movements going on at the same time do not interfere with each other. Once each block gets the data, regular mesh movement operations can broadcast the information to all the processors in the block.

Let $m = n/(2\lg n)$. We will look at only the upper triangular half of the mesh because the other half is symmetric. We can label the diagonals of blocks from 0 to $m/2-1$. Within each diagonal of blocks, the blocks are also indexed in order. The movement follows.

for i from $\lfloor \lg \frac{m}{2} \rfloor$ to 0 **do**
  for all j, $j \geq 2^i$ **do**
    for all k, $0 \leq k \leq \lfloor \frac{j+1}{2^i+1} \rfloor - 1$ **do**
      on diagonal j, move from block $j + k2^{i+1}$ to block $j + k2^{i+1} + 2^i$
      on diagonal j, move from block $j - k2^{i+1}$ to block $j - k2^{i+1} - 2^i$
    end for
  end for
end for

For the first move in the loop, the data is moved down to the same row as the destination using a column of optics and then moved to the destination using the row optics. The second move just moves data in the other half, in the row first and then the column. The distances moved along the row and column are equal since movement occurs only between blocks in the same diagonal of blocks. Note that at step i, data is moved 2^i blocks along a row of blocks and each diagonal that uses the same row for movement is spaced 2^i blocks apart. The same is true with movement in the columns, so none of the parallel movement interferes with each other.

The outer loop is executed $O(\log N)$ times. The inner two loops are executed in parallel. It takes $O(\log N)$ time to do the movement. Once each block receives the data, it takes an additional $O(\log N)$ time to broadcast the data to everything inside the block. This gives a total of $O(\log^2 N)$ time using N power. □

Proposition 5.5. *The diagonal scans of an $n \times n$ mesh with an optical mesh of trees can be computed in $O(\frac{1}{r}\log^2 N)$ time using rN peak power.*

Proof. To compute diagonal scans, again the mesh is divided into blocks as in the diagonal broadcast. The diagonal scan of the values of each diagonal in a block is computed. Now each diagonal within a block needs to use only the operation applied to all the values in that block to compute the scan of the whole diagonal. Like the regular scan operation, data is moved around in a tree one level at a time. This uses the exact same movement as the diagonal broadcast, so this is accomplished in $O(\log^2 n)$ time. □

While the communication diameter gives an $\Omega(\log N)$ running time lower bound for computing a diagonal broadcast or a diagonal scan, it is an open problem whether this can be achieved with the optical mesh of trees layout.

5.1.5 Sorting n Items

Proposition 5.6. *n comparable items stored one per processor along the diagonal of an $n \times n$ mesh with an optical mesh of trees can be sorted in $O(\frac{1}{r}\log N)$ time using rN peak power.*

Proof. As in the case of fixed-length optics, this depends on only broadcast and reduction operations, so this is achieved in $\Theta(\log n)$ time when all processors can be active at any time. □

Note that the n elements do not have to be on the diagonal; it is required only that the number of items in every square submesh is no more than the side length of the submesh. For example, the n elements could be stored in a row or column in the mesh.

5.1.6 Simulating PRAM Algorithms

Extending the idea behind sorting n items to permuting data, we can apply this to simulating PRAM algorithms.

Given a mesh of size N with an optical mesh of trees, it is possible to simulate algorithms for EREW PRAM machines that use n processors and $O(n)$ memory with $O(\log N)$

overhead. The idea is that given data a_i stored at processor P_i for $0 \leq i \leq n-1$, if we can permute the data with any permutation π so that a_i is stored at processor $P_{\pi(i)}$, then we can simulate every memory access in the EREW PRAM algorithm. In addition, it is possible to simulate algorithms that use more than $O(n)$ memory, depending on the memory accesses in the algorithm. For example, it is possible to simulate a PRAM algorithm using $O(n \log n)$ memory with the same $O(\log N)$ overhead, as each processor on the border of the mesh can reach $O(\log n)$ processors inside the mesh for additional memory in $O(\log n)$ time. In addition, it is possible to use this method to simulate algorithms on the arbitrary CRCW PRAM model. For other CRCW or CREW PRAM models, it is sometimes possible to simulate algorithms, but this requires careful analysis of memory accesses in the algorithm to ensure there are no conflicts if many processors want to access the same memory location.

Let there be a two-dimensional mesh of size N with an optical mesh of trees. Label the processors $P(i, j)$ according to their position in the mesh for $0 \leq i \leq n-1, 0 \leq j \leq n-1$.

Proposition 5.7. *Let the value a_i be initially stored in $P(i,0)$, for $0 \leq i \leq n-1$, and let π be a permutation on $\{0,1,\ldots,n-1\}$. For all i, $0 \leq i \leq n-1$, $P(\pi(i),0)$ can view value a_i in $\Theta(\log N)$ time using $\sqrt{N}$ peak power.*

Proof. This process involves three steps. In the first step, the processors are divided up into $\lg n \times n$ strips. Label the strips in order from 0 to $n/\lg n - 1$. Move the $\lg n$ data stored in processors $P(j \lg n, 0), \ldots, P((j+1)\lg n - 1, 0)$ (the data stored in strip j) to processors $P(j \lg n, j \lg n), \ldots, P((j+1)\lg n - 1, j \lg n)$. In the second step, we move each piece of data to the strip the destination is in. The data that is stored in $P(j \lg n + k, j \lg n)$ is moved to $P(j' \lg n + k, j \lg n)$, where j' is the index of the destination strip. In the third step, we move all the data to its destination. Each strip contains $\lg n$ data, and this data is moved to its destination.

The first step uses the standard movement operation of $\lg n$ data in a $\lg n \times n$ strip on the mesh. The second step can be done in $O(\log N)$ time by moving each of the $\lg n$ pieces of data to its destination strip individually but also pipelining the movement. The third step is like the previous step, where each piece of data in a strip is moved using the standard movement operation, but these $\lg n$ moves can be pipelined to achieve $O(\log N)$ running time. Each of these three steps takes $O(\log N)$ time, so the total time to permute n elements is $O(\log N)$. □

It is also possible to simulate a CRCW PRAM machine using $O(n \log n)$ memory, but we need to take care of how concurrent reads and concurrent writes are simulated. A concurrent read can be done like a broadcast along a strip of optics, and a concurrent write

can be done like a reduction on a strip of optics. The simulation can be extended to three dimensions using only $N^{3/4}$ processors to simulate $O(n)$ PRAM processors and memory.

Let there be a three-dimensional mesh of size $N^{3/4}$ with variable-length optical connections. Label the processors $P_{(i,j,k)}$ according to their positions in the mesh for $0 \leq i,j,k \leq \sqrt{n}-1$. Let data $a_{i,j}$ be stored at processor $P_{(i,j,0)}$. Each $a_{i,j}$ can be moved to processor $P_{\pi((i,j,0))}$ for all $0 \leq i,j \leq \sqrt{n}-1$ and any permutation π on $\{(i,j,0)|0 \leq i,j \leq \sqrt{n}-1\}$ in $O(\log N)$ time.

5.2 Graph Component Labeling and Minimum Spanning Forest

The following algorithms are based on the general minimum spanning forest algorithm described by Awerbuch and Shiloach [9]. Let the input graph $G = (V,E)$ have n vertices labeled 0 to $n-1$. A vector F representing the parent function of each vertex is stored one element per processor somewhere on the mesh. Initially, $F(i) = i$ for every vertex i. Vertices are labeled from 0 to the number of vertices minus 1. A star is a tree of height 1 with edges defined by F.

The Awerbuch and Shiloach minimum spanning forest algorithm is as follows. It is assumed that edge weights are distinct because if there are edges of the same weight, ties can be broken by the labels of its endpoints.
Repeat until F doesn't change:

Pointer Jumping. For every i, set $F(i) \leftarrow F(F(i))$.

Star Hooking. For every root of a star v, that is, v is in a star and $v = F(v)$, find the minimum weight edge (i,j) such that i is in the star rooted at v and j is not in the star rooted at v, that is, $F(i) = v$ and $F(i) \neq F(j)$. Mark (i,j) as an edge in the minimum spanning tree. Then, set $F(v) \leftarrow F(j)$.

Awerbuch and Shiloach showed that the marked edges form a minimum spanning forest and that the algorithm terminates after these two steps are repeated $O(\log n)$ times.

Theorem 5.8. *Given the $n \times n$ weighted adjacency matrix of an undirected graph stored one element per processor on an $n \times n$ mesh with an optical mesh of trees, the connected components and a minimum spanning forest can be determined in $O(\frac{1}{r}\log^2 N)$ time using rN peak power.*

Proof. The following is an implementation of the algorithm described by Awerbuch and Shiloach [9] for finding a minimum spanning forest. We note that the pointer jumping

implementation on the mesh uses the same technique in [25]. It is enough to show that both pointer jumping and star hooking can be accomplished in $O(\frac{1}{r}\log N)$ time since each is executed at most $O(\log N)$ times.

To initialize, F has n elements and is stored on the diagonal of the mesh, that is, $F(i)$ is at processor $P(i,i)$ with $F(i) = i$. $inStar(i)$ is also stored at processor $P(i,i)$, for $0 \leq i \leq n$.

Pointer Jumping: Let Q and R be $n \times n$ matrices where $Q_{i,j} = F(i)$ and $R_{i,j} = F(j)$. $F(F(i))$ is $R_{i,j}$, where j is the unique j such that $Q_{i,j} = j$. This takes $O(\frac{1}{r}\log N)$ time using standard broadcasts and reductions.

Star Hooking:

1. For all i, compute *inStar*, the Boolean vector that stores for each vertex True if the vertex is in a star and False if not:
   (a) Compute $F(F(i))$.
   (b) Set $inStar(i)$ to True for all i.
   (c) If $F(i) \neq F(F(i))$:
      i. Set $inStar(i)$ to False.
      ii. Set $inStar(F(F(i)))$ to False.
   (d) Set $inStar(i)$ to $inStar(F(i))$.
2. For all i such that $inStar(i) =$ True, find the minimum weight edge (i,j) such that $F(i) \neq F(j)$ and mark it as a candidate edge.
3. Find the minimum weight candidate edge incident to a vertex in each star.
4. Update F for vertices that were hooked onto other vertices.

Determining $F(F(i))$ uses the algorithm for pointer jumping and the result is stored in processor $P(i,i)$ for each i.

Step 1(d) is required so that vertices that have siblings with children are updated correctly.

To accomplish step 1(c)ii, for each i such that $F(i) \neq F(F(i))$, processor $P(i,i)$ sends a message to processor $P(F(F(i)),i)$. Then for each i, processor $P(i,i)$ checks if any of the processors in row i received a message using a row reduction and sets $inStar(i)$ to False if such a processor was found. Step 1(d) is accomplished similarly except processor $P(F(i),F(i))$ sends the value of $inStar(F(i))$ to processor $P(i,F(i))$ so processor $P(i,i)$ can do a row reduction to get the new value of $inStar(i)$. Each of these steps take $O(\frac{1}{r}\log N)$ time.

To accomplish step 2, F and *inStar* are broadcast using row and column broadcasts so that each processor $P(i,j)$ knows the values of $inStar(i)$, $F(i)$, and $F(j)$. Now the candidate edges can be marked using row reductions, as each edge incident to i is stored in row i. After this step, there may be more than one candidate edge incident to each star, so we need to find the minimum of these candidate edges for each star. The candidate edge in each row i is sent to processor $P(i,F(i))$. The minimum over all the candidate edges in column $F(i)$ is the minimum candidate edge incident to the tree rooted at $F(i)$ and is added to the minimum spanning tree. Each chosen edge then sends the value of its new parent to the root of the star it previously was in to update F. This is all done in $O(\frac{1}{r}\log N + \log N)$ time. □

Note that while a different algorithm was used in the case of fixed-length optics in Section 4.2, the Awerbuch and Shiloach minimum spanning forest algorithm can be applied to fixed-length optics, and achieve the same $O(\frac{1}{r}\log N + N^{1/4}\log N)$ running time.

Knowing how to find a minimum spanning forest, we again have the following corollary using algorithms from [7, 67].

Corollary 5.9. *The biconnected components, bridge edges, and articulation points of an undirected graph can be found in $O(\frac{1}{r}\log^2 N)$ time using rN peak power on a mesh with an optical mesh of trees.* □

5.2.1 Minimum Spanning Forest with Arbitrary Edge Input

Consider when a graph $G = (V,E)$ is stored one edge per processor on an $n \times n$ mesh, without the structure of an adjacency matrix. We assume G has no isolated vertices i.e., every vertex is an endpoint, so G has $N^{\frac{1}{2}+\varepsilon}$ vertices for some $0 \leq \varepsilon \leq \frac{1}{2}$. Note that a smaller ε corresponds to a denser graph. Since it could be possible for edges incident to $N^{\frac{1}{2}+\varepsilon}$ different vertices to be in a $\sqrt{N^{\frac{1}{2}+\varepsilon}} \times \sqrt{N^{\frac{1}{2}+\varepsilon}}$ submesh, which all need to be moved out of that submesh, the bisection bandwidth gives an $\Omega\left(\sqrt{N^{\frac{1}{2}+\varepsilon}}\right)$ running time for computing a minimum spanning forest. Note that when there are $\Omega(N)$ vertices, the following algorithm does not help since a standard mesh algorithm can solve this case in $\Theta(\sqrt{N})$ time. Since the vertices are numbered 0 to $N^{\frac{1}{2}+\varepsilon}-1$, ε can be thought of as part of the input.

Theorem 5.10. *Given the N weighted edges of an undirected graph with $N^{\frac{1}{2}+\varepsilon}$ vertices stored one edge per processor on an $n \times n$ mesh with an optical mesh of trees, a minimum spanning forest can be determined in $O\left(\sqrt{N^{\frac{1}{2}+\varepsilon}}\log^{3/2} N\right)$ time.*

Proof. We again follow the outline of the Awerbuch and Shiloach minimum spanning forest algorithm. The vector F has $N^{\frac{1}{2}+\varepsilon}$ elements and is stored one element per processor

within a submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$ in row-major-like order except the values are spread out a little with $\sqrt{\log N}$ distance between elements. That is, processor $P(i\sqrt{\log N}, j\sqrt{\log N})$ in the submesh contains $F\left(i\sqrt{N^{\frac{1}{2}+\varepsilon}}+j\right)$ for $0 \le i, j \le \sqrt{N^{\frac{1}{2}+\varepsilon}}-1$. These processors also store *inStar*, the Boolean vector that indicates whether each vertex is in a star or not.

Pointer Jumping: $F(F(i))$ can be determined using standard mesh algorithms on the submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$ in which the parent data is stored in $O\left(\sqrt{N^{\frac{1}{2}+\varepsilon}\log N}\right)$ time.

Star Hooking: The idea is, for a vertex, to find the minimum weight edge over all possible incident edges within each submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$ and then combine information in all the submeshes.

1. Compute *inStar*.
2. Broadcast F and *inStar* to each submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$.
3. Determine the minimum weight edge incident to each star in each submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$.
4. Compute the minimum weight edges among all the minimum edges found in the submeshes.
5. Update F.

Step 1 can be done in $O\left(\sqrt{N^{\frac{1}{2}+\varepsilon}\log N}\right)$ time using a regular mesh algorithm on the submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$. In step 2, the submesh that stores F and *inStar* broadcasts its values so that each submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$ has a copy. This is done in two steps, first broadcasting all the information along the rows and then broadcasting the information down the columns. Each strip of optics is able to broadcast the $\sqrt{N^{\frac{1}{2}+\varepsilon}\log N}$ values in its strip to all processors in its strip in $O\left(\sqrt{N^{\frac{1}{2}+\varepsilon}\log N}\right)$ time using pipelining. Step 3 is done in $O\left(\sqrt{N^{\frac{1}{2}+\varepsilon}\log N}\right)$ time using standard mesh algorithms. In Step 4, the data is combined together in a single submesh of size $N^{\frac{1}{2}+\varepsilon}\log N$ using reductions and pipelining on the optics similar to the data movement in step 2 backwards in $O\left(\sqrt{N^{\frac{1}{2}+\varepsilon}\log N}\right)$ time. Once the minimum edge incident to each star is known in that submesh, standard mesh data movement algorithms can be used to update F.

Step 3 requires all processors to be active in order to achieve the minimum running time, so the whole algorithm requires N peak power. □

5.3 Image Component Labeling

Theorem 5.11. *Given an $n \times n$ digital image stored one element per processor on an $n \times n$ mesh with an optical mesh of trees, the connected components of an image can be determined in $O(\frac{1}{r}\log N + \log^3 N)$ time using rN peak power.*

Proof. Like in the algorithm using fixed-length optics, this algorithm uses a divide-and-conquer. At each recursive step of the algorithm, we combine edge data by simulating the PRAM edge component-labeling algorithm. Note that there are fewer optics when the current submesh being worked on is smaller, but since the submesh is smaller, there are fewer steps to simulate a PRAM algorithm, so it balances out. Each recursive step takes $O(\frac{1}{r} + \log^2 N)$ time.

The recursion stops when the size of the submesh is $\log^2 N$. At this point, the problem in all submeshes can be solved by simulating a standard mesh algorithm in $O(\frac{1}{r}\log N)$ time. □

5.4 Distance Transforms

The following theorem follows from being able to compute row and column scans in $O(\frac{1}{r}\log N)$ time.

Theorem 5.12. *Given an $n \times n$ digital image stored on an $n \times n$ mesh with an optical mesh of trees, finding the closest object pixel to every pixel in the image according to the ℓ_1 metric can be determined in $\Theta(\frac{1}{r}\log N)$ time using rN peak power.*

Proof. As in the fixed-length optical connection length case, we accomplish computing the distance transform with respect to the ℓ_1 metric using scans along each dimension, achieving a $\Theta(\log N)$ running time for an $n \times n$ image of size on a mesh of size N. □

5.5 Sorting

In this section, we show that for $r \in O\left(\frac{1}{\sqrt{N}}\right)$ (equivalently, $O(\sqrt{N})$ peak power), permutation and sorting can be performed efficiently on an optical mesh of trees with linear speedup. The running time achieved is optimal for both permutation and sorting among all layouts with a fixed number of optical layers with respect to available peak power.

Lemma 5.13. *On a mesh of size N with an optical mesh of trees, N items can be permuted in $\Theta\left(\frac{1}{r}\log N + \sqrt{N}\log N\right)$ time using rN peak power.*

Proof. The lower bound of $\Omega(N \log N)$ total energy has already been shown, and the bisection bandwidth of the mesh still gives an $\Omega(\sqrt{N})$ running time. In the best case, work is evenly distributed among the rN active processors, so permutation takes $\Omega(\frac{1}{r}\log N)$ time, for $r \leq \frac{\log N}{\sqrt{N}}$.

The algorithm for permutation in $O(\frac{N \log N}{S})$ time follows the same outline as in the case of the optical pyramid, which is presented in detail in Section 6.5.

Moving S items at a time:

- Count the number of items with a destination in each $\lg n \times n$ submesh A_i.
- Determine the optic columns used to route items to prevent bottlenecks.
- Route items so they are distributed evenly in the A_i that contains its destination.

Then, in parallel, S items, in groups of $\lg n$ items per optic row, are moved to their destinations using pipelining until all items have reached their destinations.

Scans and reductions of S items can be computed in $O(\log N)$ time using peak power S since only the processors that have items need to be on and communicating in the tree network, so each iteration of loop takes $O(\log N)$ time. □

Theorem 5.14. *On a mesh of size N with an optical mesh of trees, N items can be sorted in* $\Theta\left(\frac{1}{r}\log N + \sqrt{N}\log N\right)$ *time using rN peak power.*

Proof. It is enough to give the sorting algorithm for $S = \sqrt{N}$. When peak power is less than $\sqrt{N}$, we simulate the $\sqrt{N}$ peak power algorithm with an optimal peak power vs. time tradeoff.

The outline of the algorithm is the same as for the sorting algorithm for one layer of optics:

Step 1. Divide the mesh into $N^{1/4}$ submeshes of size $N^{3/8} \times N^{3/8}$, and recursively sort each submesh.

Step 2. Each $N^{1/4\text{th}}$ item in the mesh is designated as a splitter. Recursively sort the $N^{3/4}$ splitters.

Step 3. Determine the rank of each splitter and calculate the rank of each item with respect to the splitters. Redistribute the items into $N^{3/4}$ parts delimited the splitters.

Step 4. Recursively sort each part.

We can recursively sort because each block is able to simulate multiple blocks of a smaller mesh with $O(\log N)$ overhead. The time to redistribute items dominates and uses the permutation algorithm just like in the one layer of optics case, which takes $O(\frac{N \log N}{S})$ time. □

5.6 d Dimensions, $d \geq 3$

We can extend many of the previous algorithms to run on meshes of higher dimensions. Let $N = n^d$. An $\underbrace{n \times \cdots \times n}_{d \text{ times}}$ mesh-connected computer is a d-dimensional mesh of size N. Each processor has wire connections to $2d$ neighbors. In dimensions higher than two, there is no notion of layers of optics, as there is much more volume to fit connections between processors. Therefore, by using an optical multidimensional mesh of trees, with an optical mesh of trees in each $n \times n$ slice, we can attain a communication diameter of $\Theta(\log N)$, which is also the running time of basic operations. Further, given any d-dimensional cube of m items, $m < N/2$, the bandwidth into and out of it is $\Theta(m^{(d-1)/d})$.

For the broadcast, reduction, and scan operations, the $\Theta(\log N)$ running time in two dimensions can also be achieved on a d-dimensional mesh, for $d > 2$, using a dimensional divide-and-conquer algorithm where the problem is solved in lower-dimensional slices first and then combined.

The sorting algorithm that obtains the following is basically the same as that in Theorem 5.14: items are recursively moved to lower-dimensional meshes in which their destinations lie.

Corollary 5.15. *On a d-dimensional mesh of size N, $d \geq 2$, with a optical multidimensional mesh of trees, N items can be sorted in $\Theta\left(\frac{1}{r}\log N + N^{1/d}\log N\right)$ time using rN peak power, where the constants depend on d.* □

5.7 Improving the Mesh of Trees Interconnection Layout

Note that the previous algorithm for sorting on the optical mesh of trees does not achieve an optimal linear speed up for sorting when $r \in \omega\left(\frac{1}{\sqrt{N}}\right)$ due to the fact that there are not enough long optics to move data when power is that high.

It is also possible to add some more optics to the original mesh of trees layout in order to achieve an optimal running time, that is, achieving a linear speedup up to $\Theta(\sqrt{N}\log N)$ power. Therefore, this gives $\Theta(\sqrt{N})$ time to sort using $\sqrt{N}\log N$ peak power, which is optimal since sorting requires $\Theta(N\log N)$ energy and it is not possible to do better than $\Theta(\sqrt{N})$ time by the bisection bandwidth of the mesh. This adds a little more complexity to the layout of optics in order to remove a logarithmic factor from the running time of sorting.

In addition to the optics in the optical mesh of trees, optics connecting processors $n/\lg n$ apart are added in each row and column. It is important to note that processors with optical

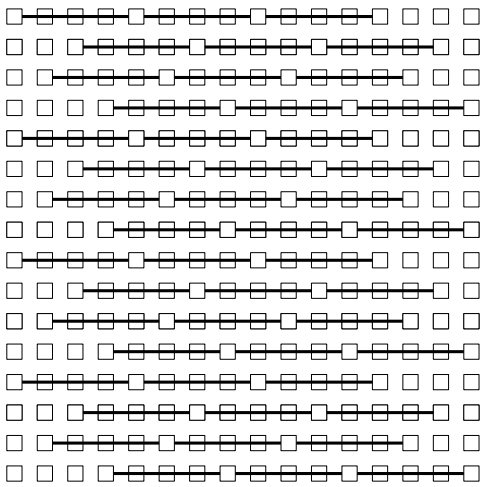


Figure 5.3: Optics added to one of the two layers in a 16×16 mesh to achieve an optimal time-power tradeoff for sorting.

connections are spaced as far apart as possible. Another interpretation of this is to add $n/\lg n$ optical meshes, each of size $\lg n \times \lg n$ to the mesh. Formally, divide the mesh into $n/\lg n \times n$ blocks, which are groups of $n/\lg n$ rows. Then for each row $i = a + b \cdot \sqrt{\frac{n}{\lg n}}$, with $0 \leq a \leq \sqrt{\frac{n}{\lg n}} - 1$ and $0 \leq b \leq \sqrt{\frac{n}{\lg n}} - 1$, in each block, connect processor $P(i, k \cdot \frac{n}{\lg n} + a \cdot \sqrt{\frac{n}{\lg n}} + b)$ and $P(i, (k+1) \cdot \frac{n}{\lg n} + a \cdot \sqrt{\frac{n}{\lg n}} + b)$, for $0 \leq k \leq \lg n - 2$. The other layer is the same layout rotated 90 degrees. Figure 5.3 illustrates the added connections in one of the layers.

The advantageous property of these additional optical connections is that there are enough optics to move $\Theta(\sqrt{N} \log N)$ pieces of data, each a distance of $\Theta(\sqrt{N})$, in $\Theta(\log N)$ time using $\Theta(\sqrt{N} \log N)$ power. Another property used by sorting is that the there is one processor with optical connections within every $\sqrt{n/\lg n} \times \sqrt{n/\lg n}$ submesh. This gives enough bandwidth for data to be moved in and out of the added optical meshes.

Lemma 5.16. *On a mesh of size N with an optical mesh of trees and the additional connections illustrated in Figure 5.3, N items can be permuted in* $\Theta\left(\frac{1}{r} \log N + \sqrt{N}\right)$ *time using rN peak power.*

Proof. It is enough to show how to redistribute the items using $n \lg n$ power such that each item is within the $n/\lg n \times n/\lg n$ submesh that its destination is in. Once this is achieved, the permutation algorithm on the optical mesh of trees can be executed with $n/\lg n$ power in each submesh in parallel.

The outline is similar to the original permutation algorithm. Items are first moved to the $n/\lg n \times n$ submesh in which its destination lies. Afterwards, items are moved to the

$n/\lg n \times n/\lg n$ submesh in which its destination lies. Like before, the initial movement to the destination $n/\lg n \times n$ submesh requires computing an intermediate destination for each item and the route it takes to get there so that no optical connection is used more than $\Theta(n)$ times.

The algorithm for the initial movement is a series of iterations where $n\lg n$ items are moved concurrently. The mesh can be divided into submeshes of size $n/\lg n$, each containing a processor with optical connections. At each iteration, $\lg n$ items from each of these submeshes is moved to an intermediate location such that the items are divided evenly among each $n/\lg n \times n/\lg n$ submesh in the destination $n/\lg n \times n$ submesh. The intermediate location and route to get there is computed like before at the beginning of each iteration using a scan operation. The mesh of trees network is used for communication within each $n/\lg n \times n/\lg n$ submesh to move items to an available processor.

Once each item is in the correct $n/\lg n \times n$ submesh, items can be moved to the correct $n/\lg n \times n/\lg n$ submesh using the optics of length $n/\lg n$ in each row. $\lg n$ items use each row concurrently and are moved to an available processor in their destination $n/\lg n \times n/\lg n$ submesh using the mesh of trees network. This takes $\Theta(\log n)$ time for each item. □

Theorem 5.17. *On a mesh of size N with an optical mesh of trees and the additional connections illustrated in Figure 5.3, N items can be sorted in* $\Theta\left(\frac{1}{r}\log N + \sqrt{N}\right)$ *time using rN peak power.*

Proof. This uses the same algorithm before, except peak power can be up to $O(\sqrt{N}\log N)$. Since the running time is still dominated by the permutation algorithm, the running time is the same as the running time for permutation. □

Again, this can be extended to sorting with a d-dimensional mesh in $\Theta(N^{1/d})$ time using $N^{\frac{d-1}{d}}\log N$ peak power.

Corollary 5.18. *On a d-dimensional mesh of size N,* $d \geq 2$*, with a optical multidimensional mesh of trees and the additional connections illustrated in Figure 5.3, N items can be sorted in* $\Theta\left(\frac{1}{r}\log N + N^{1/d}\right)$ *time using rN peak power, where the constants depend on d.* □

CHAPTER 6

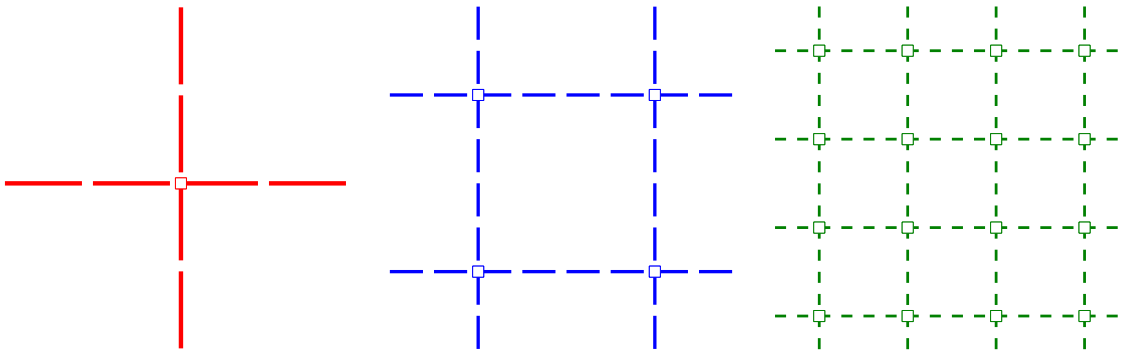
One Layer of Variable-Length Connections (Optical Pyramid)

In the following model, called the *optical pyramid*, we assume that only one layer of optics is allowed and optics cannot cross in this layer. The difference between this model and the fixed-length optics in Chapter 4 is that here we allow optics of any length in the interconnection layout (lengths can range from 1 to $n-1$). While this is not as powerful as two layers of optics as described in Chapter 5, one layer of connections is not as difficult to build in hardware and this model admits significant unique results regarding time-power usage for various problems. The most interesting result it an optimal sorting algorithm with a sublinear slowdown.

Recall the optical connection layout in the optical mesh. Given optics of a specific length, say of length n/k, a $k \times k$ mesh consisting of only optical connections is on top of the wire mesh. Processors $P(i, j)$, for $i, j \in \{0, n/k, 2n/k, \ldots, (k-1)n/k)\}$ will be part of this $k \times k$ optical mesh. Figure 4.1 illustrates an example of an optical mesh on an 8×8 mesh with $k = 4$.

In the more general case where we allow optics of any length, multiple mesh-like optical networks of different sizes can be embedded and a pyramid-like network can be achieved within one layer of optics, described as follows: let a be the index of the least significant 1 bit of i. For each row i such that $2 \leq a \leq \lg n - 1$, connect the processor in column $b \cdot 2^{a-1}$ to the processor in column $(b+1) \cdot 2^{a-1} - 1$, for $0 \leq b \leq n/2^{a-1} - 1$. Likewise, for each column i such that $2 \leq a \leq \lg n - 1$, connect the processor in row $b \cdot 2^{a-1}$ to the processor in row $(b+1) \cdot 2^{a-1} - 1$, for $0 \leq b \leq n/2^{a-1} - 1$. Note that optics are of length $2^{a-1} - 1$ and it is easy to check that none of these optical connections cross.

Alternatively, a recursive definition may be easier to understand: on an $n \times n$ mesh, place optical connections of length $n/4 - 1$ so that the following eight pairs of processors are connected:



(a) 1×1 optical mesh (optics of length 7)

(b) 2×2 optical mesh (optics of length 3)

(c) 4×4 optical mesh (optics of length 1)

Figure 6.1: Different levels of the layout of optical connections for a 32×32 mesh.

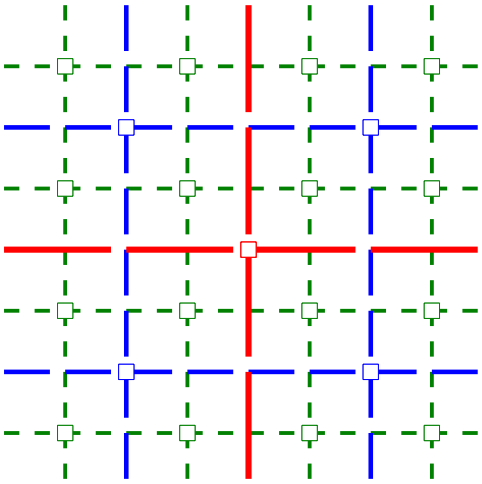


Figure 6.2: Optical connections and processors in optical meshes of a 32×32 mesh with optical pyramid.

$P(\frac{n}{2}, 0)$, $P(\frac{n}{2}, \frac{n}{4}-1)$ $\quad$ $P(\frac{n}{2}, \frac{n}{4})$, $P(\frac{n}{2}, \frac{n}{2}-1)$

$P(\frac{n}{2}, \frac{n}{2})$, $P(\frac{n}{2}, \frac{3n}{4}-1)$ $\quad$ $P(\frac{n}{2}, \frac{3n}{4})$, $P(\frac{n}{2}, n-1)$

$P(0, \frac{n}{2})$, $P(\frac{n}{4}-1, \frac{n}{2})$ $\quad$ $P(\frac{n}{4}, \frac{n}{2})$, $P(\frac{n}{2}-1, \frac{n}{2})$

$P(\frac{n}{2}, \frac{n}{2})$, $P(\frac{3n}{4}-1, \frac{n}{2})$ $\quad$ $P(\frac{3n}{4}, \frac{n}{2})$, $P(n-1, \frac{n}{2})$

Then recursively place eight optics in each of the four $n/2 \times n/2$ submeshes until reaching the base case of a 4×4 mesh after a total of $\lg n - 1$ levels of recursion.

Figures 6.1 and 6.2 illustrate the layout of the optical connections. In Figure 6.2, only processors that act as processors in the optical meshes are shown. That is, processors in the "base" of the pyramid are not shown.

0	3	4	5	58	59	60	63
1	2	7	6	57	56	61	62
14	13	8	9	54	55	50	49
15	12	11	10	53	52	51	48
16	17	30	31	32	33	46	47
19	18	29	28	35	34	45	44
20	23	24	27	36	39	40	43
21	22	25	26	37	38	41	42

Figure 6.3: 8×8 mesh Hilbert curve ordering.

Recall that $S = rN$. For convenience, let $s = \sqrt{S}$ and assume that s is a power of 2. This simplifies notation as we will frequently refer to the $s \times s$ optical mesh. While the statements of the results are in terms of r, their proofs typically are in terms of S.

6.1 Useful Properties

The pyramid layout of optical connections has some advantageous properties. It has a communication diameter of $\Theta(\log N)$, which is far smaller than the $\Theta(\sqrt{N})$ communication diameter of the mesh. Each processor can communicate with any processor in $O(\log N)$ time because processor $P(\frac{n}{2}, \frac{n}{2})$ can reach any processor in the mesh in at most $6(\lg n - 1)$ time steps by moving from the longest-length optics to successively shorter optics, using at most six time steps on each optical mesh, and always moving towards the quadrant the destination processor is in. Thus small amounts of data can be moved across the mesh quickly.

A particularly useful property is the recursive definition of the layout of the optical pyramid: the layout of the optical connections in any square submesh is also a logical optical pyramid. That is to say, an optical pyramid over any square submesh can be simulated with a constant factor overhead. In situations where recursive algorithms are applied to ordered data, the Hilbert space-filling curve ordering (see Figure 6.3) is useful as it keeps local data in close proximity. In fact, for items ordered in a Hilbert curve, every set of k consecutive items is contained within a square submesh with a logical optical pyramid of size at most $4k$.

Another property is that optics of the same length compose a network that can simulate

an $n/2^{a+1} \times n/2^{a+1}$ optical mesh with a constant factor overhead, for $2 \leq a \leq \lg n - 1$. This is accomplished using processors evenly spaced 2^{a+1} apart in the entire mesh, that is, processors $P(i, j)$, for $i, j \in \{2^a, 2^a + 2^{a+1}, \ldots, 2^a + (n/2^{a+1} - 1)2^{a+1}\}$, are processors in the optical mesh. Specifically, it takes eight time steps to communicate between adjacent processors in an optical mesh; processors need only to send data over four wire connections and four optical connections to reach the next processor. For convenience, we will refer to these as optical meshes, ignoring the constant factor overhead contributed by the gaps between optics. Note that communication from a processor on one optical mesh to a processor on the next smaller or next large optical mesh takes six time steps.

This model is named the optical pyramid because the connections are almost the same as those in the classical pyramid computer model [14, 49]. Figure 6.4 shows the structure and communications links of a pyramid computer. In fact, a pyramid computer can be simulated by an optical pyramid with just a constant factor overhead. The pyramid network is a fundamental layout that has been considerably studied in the past, appearing in VLSI design as well as being a model of parallelism studied for problems involving images, adjacency matrices, etc. [14, 28, 34, 49, 71, 78]. Algorithms for the pyramid computer can be run on the optical pyramid using stepwise simulation with a constant factor overhead, though such simple usage in general has a linear tradeoff of time vs. peak power and is of less interest here. No previous work on pyramid computer algorithms considered energy usage, that is, there was no penalty for having all processors running all the time. Further, for communication-intensive problems such as sorting, simple bandwidth arguments show that the pyramid with all processors active is no faster than the base mesh; therefore, the pyramid has no advantages for problems which require sorting. This suddenly changes when peak power is limited, though this requires pyramid algorithms quite different from previous ones. The following results utilize the pyramid in new ways; most significantly, several of the power aware algorithms that will be presented achieve a sublinear increase in time as the peak power decreases, a property unachievable if only the base mesh is utilized.

For permutation, for peak power S, $1 \leq S \leq N$, the lower bound on permutation energy shows that any optical layout needs at least $\Omega(\frac{N \log N}{S})$ time. For the optical pyramid, another lower bound is $\Omega(N/\sqrt{S})$, which is more than $\Omega(\frac{N \log N}{S})$ when $S > \log^2 N$. To prove this bound, first observe that on a mesh with no optics, transposing the data has $\Omega(N)$ processors with a destination at distance $\Omega(N^{1/2})$, for a total distance of $\Omega(N^{3/2})$ covered. To maximize distance at each time step on a mesh with an optical pyramid, the longest optical connections must be used. Note that the lengths of optics are $2^{a-1} - 1$, for $2 \leq a \leq \lg n - 1$, and that there are $n^2/2^{2a-1}$ optical connections of that length. The optical connections can be re-indexed from longest to shortest to get 2^{2i+1} optical connections of length $n/2^{i+1} - 1$,

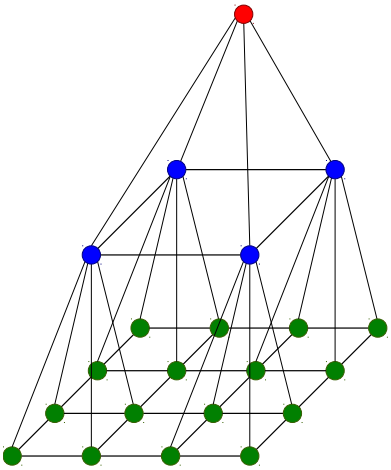


Figure 6.4: A classical pyramid computer model of size 16.

for $1 \leq i \leq \lg n - 2$. As the lengths get shorter, the number of optics of that length grows exponentially, so if the peak power is S, the number of different lengths of optics that must be used to maximize movement is at most $\lg s$. Therefore, the most total movement possible per time step is $\sum_{i=1}^{\lg s} 2^{2i+1} \cdot (n/2^{i+1} - 1) = 2ns - 2n - \frac{8s^2}{3} + \frac{8}{3} \in O(ns)$. Since time is total distance divided by distance per time step, time is equal to $\Omega(\frac{n^3}{ns}) = \Omega(N/\sqrt{S}) = \Omega(\frac{1}{\sqrt{r}}\sqrt{N})$.

6.2 Lower Bound for Sorting

A lower bound for permutation on the optical pyramid layout was just given, but we can make a stronger statement and show this bound applies to any possible layout of noncrossing optics. We first prove a lower bound on the crossing number of a complete bipartite graph, which will be used in the proof of the lower bound for sorting. The crossing number $\mathrm{cr}(G)$ of a graph is defined to be the minimum number of crossing pairs of edges, over all drawings of G in the plane. Crossing numbers of graphs have been studied extensively [21, 24, 36]. In general, determining the crossing number of a graph is NP-complete, but it is possible to show various bounds for specific classes of graphs. Here we consider a lower bound only in terms of asymptotic limits. The following proofs regarding crossing number follow ideas found in [36], but Leighton [40] claims these techniques are well-known. The complete bipartite graph with a bipartition into two sets of vertices, one of size M and the other of size N, is denoted $K_{M,N}$.

Lemma 6.1. *The crossing number of* $K_{3,N}$ *is* $\Omega(N^2)$.

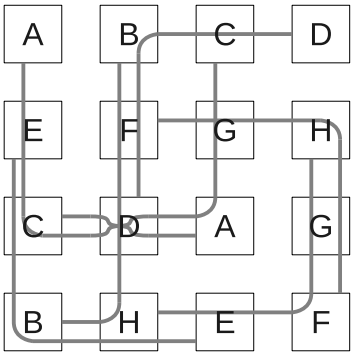


Figure 6.5: A drawing of an execution of a permutation where pairs of processors with the same label exchange data.

Proof. We can assume that no two edges that share an incident vertex cross because such a crossing can be removed and still produce a valid drawing.

Given a drawing of $K_{3,N}$, consider the N subdrawings where one of the vertices in the part with N vertices is removed along with its incident edges. Each crossing in the drawing of $K_{3,N}$ exists in $N-2$ of these subdrawings, that is, all of the subdrawings except the two that removed one of the edges in the crossing. The number of crossings in each subdrawing must be at least the crossing number of $K_{3,N-1}$. Therefore, $(N-2)\operatorname{cr}(K_{3,N}) \geq N\operatorname{cr}(K_{3,N-1})$. We also know that $\operatorname{cr}(K_{3,3}) = 1$ since $K_{3,3}$ is not planar and can be drawn with one crossing. This implies $\operatorname{cr}(K_{3,N}) \geq \frac{N}{N-2} \cdot \frac{N-1}{N-3} \cdot \frac{N-2}{N-4} \cdots \frac{6}{4} \cdot \frac{5}{3} \cdot \frac{4}{2} = \frac{N(N-1)}{6} \in \Omega(N^2)$ □

Lemma 6.2. *The crossing number of $K_{M,N}$ is $\Omega(M^2N^2)$.*

Proof. The argument is similar to the proof of the previous lemma. Consider M subdrawings of $K_{M,N}$ where one of the vertices in the part of the graph with M vertices is removed along with its incident edges. Each crossing in the drawing of $K_{M,N}$ exists in $M-2$ of these drawings. The number of crossings in each subrawing must be at least the crossing number of $K_{M-1,N}$. Therefore, $(M-2)\operatorname{cr}(K_{M,N}) \geq M\operatorname{cr}(K_{M-1,N})$. Applying the previous lemma, this implies $\operatorname{cr}(K_{M,N}) \in \Omega(\frac{M}{M-2} \cdot \frac{M-1}{M-3} \cdot \frac{M-2}{M-4} \cdots \frac{5}{3} \cdot \frac{4}{2} \cdot N^2) = \Omega(M^2N^2)$. □

Specifically, we will use the fact that $\operatorname{cr}(K_{N/2,N/2}) \in \Omega(N^4)$ in the following theorem.

Theorem 6.3. *Given a mesh of size N with one layer of optical connections and rN peak power, a permutation of N items stored one per processor requires $\Omega\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time.*

Proof. A permutation can be represented by an N vertex graph, where each processor is represented by a vertex and an edge exists between two vertices if there is an item with

initial and final positions that correspond to those two vertices. We may exclude items that do not move as they do not require any energy to reach their destination.

Every possible execution of a permutation can be represented by a planar drawing of the edges of the graph corresponding to the permutation. To represent an execution by a drawing, vertices are placed as an $\sqrt{N} \times \sqrt{N}$ lattice in the plane and an edge is drawn through a vertex if that vertex is used in the route taken by the item corresponding to that edge. Note that edges can be drawn such that edges share points at discrete locations and they only share a point when their corresponding routes cross. See Figure 6.5 for an example of a drawing that corresponds to the execution of a permutation on a 4×4 mesh. Whenever two edges share a common point in such a drawing, other than at an endpoint, one of the corresponding routes in the execution must have incurred one unit of energy usage. While the drawing of an edge does not specify what parts of the route are using wires or optics, it cannot be the case that both routes passed over the common processor using optics as optics cannot cross.

In a given execution, it may be the case that the routes of many items cross at a common location. In the drawing, this would correspond to edges of these routes sharing a common point in the plane. We call a point in the plane that is shared by more than one edge an *intersection*.

Let processors be labeled $P_0, \ldots, P_{N-1}$. Consider $N/2$ permutations, where permutation i moves the item located at P_j to $P_{N/2+(j+i \bmod N/2)}$, for $0 \leq i \leq N/2-1$ and $0 \leq j \leq N/2-1$. An execution of one of these $N/2$ permutations corresponds to a drawing of the graph that represents the permutation. The union of these $N/2$ drawings is a drawing of the complete bipartite graph $K_{N/2,N/2}$ and such a drawing has $\Omega(N^4)$ crossings.

To prove the theorem, it is enough to show that computing all $N/2$ permutations takes a total of $\Omega(N^2/\sqrt{S})$ time. We first assume that computing these $N/2$ permutations in total can be accomplished in T time. Therefore, every location in the mesh is used at most T times, and thus any point in a drawing may be shared by at most T edges. This implies $O(T^2)$ pairwise crossings of edges may be located at an intersection.

The total energy required for any execution must be at least the sum of the number of times each intersection is traversed. We will show this is at least the product of the number of intersections and the maximum number of times each intersection is used. Intuitively, the minimum energy is achieved when the number of points of intersection is minimized and each intersection is used the maximum possible number of times.

Let there be m intersections in the planar drawing of these $N/2$ permutations. For each intersection i, let x_i be the number of edges which go through the intersection. We know $x_i \leq T$, energy is at least $\sum_{i=1}^{m} x_i$, and the total number of pairwise crossings is $\sum_{i=1}^{m} \frac{x_i(x_i-1)}{2} \in$

$\Omega(N^4)$. Using the method of Lagrange multipliers, the minimum energy occurs when all x_i have the same value. Let this value be x. This implies that $m \in \Omega(N^4/x^2)$. Energy is at least the number of times each intersection is traversed, which is $\Omega((N^4/x^2) \cdot x) = \Omega(N^4/x)$. Time is at least energy divided by peak power, so $T \in \Omega((N^4/x)/S)$. Combining this with the fact that $x \leq T$, T must be at least $\Omega(N^2/\sqrt{S})$. □

Since permutation can be accomplished by sorting using each item's destination as the key, sorting must also take $\Omega\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time. Combining this with the linear speedup lower bound for sorting, we have the following theorem.

Theorem 6.4. *Given a mesh of size N with one layer of optical connections and rN peak power, sorting N items requires* $\Omega\left(\frac{1}{\sqrt{r}}\sqrt{N} + \frac{1}{r}\log N\right)$ *time.* □

6.3 Basic Operations

The following operations on the optical pyramid can be accomplished using less time and power compared to a mesh without optics.

6.3.1 Routing

If just one processor needs to send a word of data to another processor, routing the data takes $O(\log N)$ time simply by using the tree structure of the pyramid. If k processors have data that needs to be sent to other processors, the values at those processors are first moved to the $k \times k$ optical mesh in $O(k + \log N)$ time using the tree structure of the pyramid. Then all values can be routed on the optical mesh to their $n/k \times n/k$ submesh destination. Using pipelining, this can all be accomplished in $O(k + \log N)$ time. More generally, S processors can send data to other processors in $O(\sqrt{S} + \log N)$ time using the $s \times s$ optical mesh as long as the bandwidth on the optical mesh between processors and their destinations is $\Omega(\sqrt{S})$.

6.3.2 Broadcast and Reduction

The $\Theta(\frac{1}{r} + \sqrt{N})$ running time on a standard mesh can be lowered to $\Theta(\frac{1}{r} + \log N)$ on the optical pyramid. To do this, the value to be broadcast is moved to processor $P(n/2, n/2)$ in $O(\log N)$ time. Then, using the tree network embedded in the pyramid, the value is broadcast to each $n/s \times n/s$ submesh in $O(\log N)$ time. In each submesh, 1 unit of energy per time step is used to broadcast to all processors in the submesh, taking $\Theta(\frac{1}{r})$ time.

Since the data movement in a reduction is the reverse of a broadcast operation, this can also be accomplished in $\Theta(\frac{1}{r} + \log N)$ time.

6.3.3 Scan

If the values are ordered in a Hilbert or z-order curve in the mesh with optical pyramid, a scan can be computed in $\Theta(\frac{1}{r} + \log N)$ time using the tree network that is a subset of the optical pyramid. If the values are in row-major order, the $\sqrt[4]{N} \times \sqrt[4]{N}$ optical mesh can be used to compute the scan in $\Theta(\frac{1}{r} + \sqrt[4]{N})$ time.

6.4 Image Component Labeling

Since the optical pyramid can simulate any pyramid algorithm, the image component labeling algorithm for a pyramid computer of [49] can be used. However, energy usage and time-power tradeoffs of such algorithms have not been analyzed in any previous work. The following achieves linear speedup using $N^{3/4}$ peak power.

Theorem 6.5. *Given an $n \times n$ digital image stored one element per processor on a mesh of size N with optical pyramid, the connected components of an image can be determined in $\Theta(\frac{1}{r} + N^{1/4})$ time using rN peak power.*

Proof. This result is similar to in the case of fixed length optics, but the optical pyramid has the extra power to remove a logarithmic factor from the running time. With fixed length optics, there is only one length of optical connections so optical meshes of different sizes must be simulated in recursive applications of the algorithm. In the case of the optical pyramid, the additional optical meshes allow the algorithm to be executed without this simulation overhead.

As in the fixed length optics case, the algorithm recursively solves the problem in four submeshes then runs a graph component labeling algorithm to consistently label components spanning more than one of the submeshes. When the current size of the mesh being worked on is $m \times m$, the optical pyramid provides a $\sqrt{m} \times \sqrt{m}$ optical mesh in which the graph component labeleing is run on to perform the relabeling.

Using $N^{3/4}$ power, relabeling in each $m \times m$ submesh at a given level of recursion can be accomplished in parallel in $O(\sqrt{m})$ time using m power per submesh, for $m \geq N^{1/4}$. When $m < N^{1/4}$, $N^{3/4}/m$ submeshes are executed in parallel at a time. Since there are N/m^2 submeshes total, this takes $O(\frac{N}{m^2} / \frac{N^{3/4}}{m} \cdot \sqrt{m}) = O(N^{1/4}/\sqrt{m})$ time. The base case occurs at a constant size submesh which can be solved by a standard mesh algorithm. The total time is $\Theta(N^{1/4})$ using $N^{3/4}$ power, which gives $\Theta(\frac{1}{r} + N^{1/4})$ time with rN power. □

6.5 Sorting

In this section, we show that permutation and sorting can be accomplished in $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time using the optical pyramid, as opposed to the $\Theta(\frac{1}{r}\sqrt{N})$ time required on a standard mesh, for $r \in \Omega\left(\frac{\log^2 N}{N}\right)$. Thus the lower bounds shown previously can be achieved. Again, we emphasize that this is a sublinear slowdown. We first give an algorithm for permutation and then use it within the sorting algorithm. Note that, due to the issue of many items clustered together not having enough bandwidth to be spread out in the desired running time, the algorithm is not as simple as moving S items at a time to their destinations.

Lemma 6.6. *On a mesh of size N with an optical pyramid, N items can be permuted in $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}+\frac{1}{r}\log N\right)$ time using rN peak power.*

Proof. Our algorithm uses the $s \times s$ optical mesh to move data across the mesh. We conceptually partition the mesh horizontally into $n/s \times n$ submeshes, referred to as *optic rows*, labeled $A_0,\ldots,A_{s-1}$ and also partition the mesh vertically into $n \times n/s$ submeshes, referred to as *optic columns*, labeled $B_0,\ldots,B_{s-1}$. Every A_i contains a single row of the $s \times s$ optical mesh, and every B_i contains a single column, for $0 \leq i \leq s-1$. Each processor on the optical mesh belongs to a unique pair (A_i, B_j) of submeshes.

Figure 6.6 is an outline of the algorithm. In more detail, the following is repeated N/S times:

- Within each B_j, s items that have not yet been moved are chosen. A copy of each of these s items is moved along the vertical optical connections in B_j to the processor on the optical mesh in $A_{i_{\text{dest}}}$, the optic row with the item's destination. Each processor on the optical mesh has a counter that keeps track of the number of items that were moved to it. Every time an item is moved to a processor on the optical mesh, the counter is incremented by one and the item is discarded. The result of this is that each processor on the optical mesh in (A_i, B_j) knows $count(i, j)$, the number of items in B_j that have a destination in A_i.

- A scan operation is performed on the counters on the optical mesh that determines the number of items at processors in columns numbered less than each processor's column number in the same row. A reduction is also performed in each A_i to determine the number of items in that row. With this information, each item is tagged with the column number, j_{int}, of the optical connections it must use in order for items to be distributed as evenly as possible among the s processors on the optical mesh in each submesh A_i. That is, each item is assigned an $n/s \times n/s$ submesh, in $A_{i_{\text{dest}}}$ and $B_{j_{\text{int}}}$, as

for all items; S at a time, s per optic column **do**
  $i_{\text{int}} \leftarrow$ index of optic column of origin
  $i_{\text{dest}} \leftarrow$ index of optic row of destination
  for $i, j \leftarrow 0, s$ **parallel do**
    $count(i, j) \leftarrow$ number of items from B_j with $i = i_{\text{dest}}$
    $x(i, j) \leftarrow \sum_{j'=0}^{j-1} count(i, j')$
    $y(i) \leftarrow \sum_{j'=0}^{s-1} count(i, j')$
    $z \leftarrow$ index within items moved to $(A_{i_{\text{dest}}}, B_j)$
    $j_{\text{int}} \leftarrow \left\lfloor \frac{x(i,j)+z}{y(i)} \cdot s \right\rfloor$
  end for
  Move item to $A_{i_{\text{int}}}$
  Move item to $B_{j_{\text{int}}}$
  Move item to $A_{i_{\text{dest}}}$
end for
for $i \leftarrow 0, s$ **parallel do**
  for all items in A_i; s at a time **do**
    Move item to destination
  end for
end for

Figure 6.6: Permutation algorithm.

an intermediate location. Specifically, for an item starting in B_j with a destination in $A_{i_{\text{dest}}}$, if x is the number of items with a destination in the same submesh $A_{i_{\text{dest}}}$ and in a submesh $B_{j'}$, $j' < j$, y is the number of items with a destination in the same submesh $A_{i_{\text{dest}}}$ and z is the index of the item out of those from B_j with destination $A_{i_{\text{dest}}}$, then the item has an intermediate location in column $j_{\text{int}} = \lfloor \frac{x+z}{y} \cdot s \rfloor$ of optics. To tag each item, the items move to the processors on the optical mesh as in the previous step, but instead of being discarded once it reaches the optical mesh processor, it gets tagged and reverses its movement and returns to its original location.

- Items are moved to the diagonal of the mesh, that is, row $i_{\text{int}} = j$ of the optical mesh, which is the intermediate row items move in before moving to their destination row. Then, each item moves to its intermediate column j_{int}, then to its destination row i_{dest}. Once each item is moved to its destination row, it is spread out so that each $n/s \times n/s$ submesh in the row has an equal number of items that have been moved to that row so far.

Now each A_i contains only items that have destinations within A_i. For each A_i, s active processors are used to move s items at a time to their correct destinations.

At each iteration, there are never more than s items moving at a time in each A_i or B_j, so there is enough bandwidth to accomplish each iteration in $O(\sqrt{S})$ time. Since it takes $O(\log N)$ time to reach a processor on the optical mesh and there are N/S iterations, the running time is $\Theta(N/\sqrt{S}+\frac{N\log N}{S}) = \Theta(\frac{1}{\sqrt{r}}\sqrt{N}+\frac{1}{r}\log N)$. □

As an example, consider the transposition permutation where each item $a_{i,j}$ in processor $P(i,j)$ has a destination of $P(j,i)$. Assume $S = N^{1/4}$ peak power is available. At each iteration of the algorithm, $N^{1/4}$ items are moved, $N^{1/8}$ items per optic column. Item $a_{i,j}$ has $i_{\text{int}} = i_{\text{dest}} = \left\lfloor \frac{j}{N^{7/8}} \right\rfloor$ since the index of the origin column of each item is also the index of the destination row in a transpose. Each processor in the $N^{1/8} \times N^{1/8}$ optical mesh computes *count*, which is equal to $N^{1/8}$ for processors on the diagonal and 0 for all other processors. Since there are exactly $N^{1/8}$ items moved to each optic column, j_{int} is also equal to $\left\lfloor \frac{j}{N^{7/8}} \right\rfloor$. Once $N^{1/8}$ items are moved to each optic row, they are uniformly distributed among the $N^{3/8} \times N^{3/8}$ submeshes, that is, at the end of the iteration, each submesh contains exactly one of these items.

A generalization of the permutation algorithm is needed for the sorting algorithm. We will call it *redistribution*. Instead of a unique destination processor for each item, the mesh is partitioned into approximately square contiguous blocks and each item has a destination block. Within its destination block, an item can be assigned to an arbitrary processor as long as each processor ends up with a single item. Within a block, the processor of smallest index is designated as the representative destination location of the block. For simplicity, we assume block sizes are a multiple of N/S. For block sizes less than or equal to N/S, items moving to these blocks use the same data movement as in the permutation algorithm. For blocks of size N/S, there is no change to the permutation algorithm except items designate their destination as the representative processor of their destination block and, in the last step, items just fill in each $n/s \times n/s$ submesh as they arrive by moving the first processor in row-major order of the block without an item yet. For blocks of size greater than N/S, each processor on the $s \times s$ optical mesh has to keep track of whether all the processors in its $n/s \times n/s$ submesh has received an item yet. Before each iteration of the first loop in the permutation algorithm, the S items being moved are moved on the $s \times s$ optical mesh. Each processor on the optical mesh determines how many of the S items will be moved to its submesh, and if it is full, it changes the item's destination to the next available submesh. This takes $\Theta\left(N/\sqrt{S}+\frac{N\log N}{S}\right)$ time.

Theorem 6.7. *On a mesh of size N with an optical pyramid, N items can be sorted in* $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}+\frac{1}{r}\log N\right)$ *using rN peak power.*

```
procedure SORT(M, S):

if SIZE(M) = S then
    Standard sort M
else
    Partition M into submeshes M_i of size √(NS)
    for all M_i do                                  ▷ step 1
        SORT(M_i, S)
    end for
    Select every S^th item as splitter
    Standard sort N/S splitters                     ▷ step 2
    Redistribute into N/S submeshes M'_i            ▷ step 3
    for all submeshes M'_i do                       ▷ step 4
        SORT(M'_i, S)
    end for
end if
```

Figure 6.7: Sorting algorithm for mesh M of size N with peak power S, for $N^{1/4} \leq S \leq N$.

Proof. The permutation lower bound gives the lower bound for sorting. The following algorithm sorts items into a Hilbert space-filling curve order. All sorting in any submesh using the standard mesh algorithm or a recursive call is in terms of a Hilbert space-filling curve. If another order is desired, one can switch to any other sorted order by a simple permutation. Figure 6.7 gives an outline of the recursive algorithm that sorts N items with S peak power, for $N^{1/4} \leq S \leq N$.

The base case of the algorithm occurs when the submesh is of size S, when a standard mesh algorithm [49] can sort the $s \times s$ mesh in $O(\sqrt{S})$ time using peak power S. When $S = N$ this is just the standard mesh sorting algorithm that sorts in $\Theta(\sqrt{N})$ time. There are four steps:

Step 1. The mesh is partitioned into $\sqrt{N/S}$ submeshes of size $\sqrt{NS}$, and each submesh is individually sorted one at a time with S peak power in $O(\sqrt{N})$ time, for a total of $O(N/\sqrt{S})$ time.

Step 2. Every S^{th} item in each of the submeshes sorted in step 1 is designated as a splitter and moved to the $n/s \times n/s$ optical mesh, where they are sorted using a standard mesh sorting algorithm. Since $S \geq N^{1/4}$, this takes $O(N/\sqrt{S})$ time.

Step 3. The data is partitioned along a Hilbert curve. Each splitter must determine its correct position in the Hilbert curve ordering, and each item must determine which part it belongs in. To do this, S copies of the splitters in parallel are distributed so that each submesh of size N/S has a copy of the splitters. In each submesh, there is one active

processor at any given time and the splitters are merged with the items to determine which part each item belongs in and the number of items from that submesh that belong in each part. When $S > N^{1/3}$, the number of splitters, N/S, is less than the size of an individual submesh, $\sqrt{NS}$, so there are extra copies of the splitters that can be disregarded. Each item individually can determine its part in $O(\log N)$ time by just searching the copy of the splitters. Then a reverse movement happens so that the total number of items in each part for the whole mesh is determined. This data is sent to all the items in the mesh so each item knows the location of the part it needs to move to. Then, the redistribution algorithm is used to move each item in its correct part. This takes $\Theta(N/\sqrt{S})$ time.

Step 4. In the worst case, the size of each part is $O(\sqrt{NS})$, but each part of size $O(\sqrt{NS})$ takes $O(\sqrt{N})$ time to sort, so this step is accomplished in $O(N/\sqrt{S})$ time.

In the case where $S < N^{1/4}$, a few modifications to the algorithm must be made. For steps 1 and 2, the sorting algorithm using peak power $\sqrt{N}$ is simulated. Therefore, in step 1, the items are partitioned into $N^{1/4}$ submeshes of size $N^{3/4}$ and each is recursively sorted with S power. For step 2, the simulation of sorting the $\sqrt{N}$ splitters is run on the $s \times s$ optical mesh, that is, the wire $\sqrt{n}/s \times \sqrt{n}/s$ mesh around each processor in the optical mesh that acts as a submesh of size $\sqrt{N}/S$ part of the $\sqrt{N}$ splitters. Since the total energy required to sort $\sqrt{N}$ items on a mesh is $N^{3/4}$, step 2 takes $O(N^{3/4}/S)$ time, which is within the required time. No other changes are required for the remaining steps, where the redistribution algorithm and recursive calls with S peak power are used, which takes $\Theta(N/\sqrt{S} + \frac{N \log N}{S})$ time. □

There are many algorithms that depend on sorting or permutation that can achieve similar time-power tradeoffs on the optical pyramid simply by using this sorting algorithm. We examine a few of these algorithms in the next few following sections. One feature these algorithms have in common is that they all recursively solve subproblems in parallel on submeshes with power uniformly distributed among the submeshes. We call this *divide and conquer with uniform power distribution*. Once the available power per submesh falls below $\lg^2 N$ power, a serial algorithm is simulated to solve the whole submesh. The exact value of $\lg^2 N$ is somewhat arbitrary; we use it for simplicity We will only consider $\Omega(\log^8 N)$ available peak power, because when peak power is close to 1, algorithms are more serial in nature and are less interesting.

6.6 All-Nearest-Neighbors

Given a set A of points in d-dimensional space, the all-nearest-neighbors problem is to determine, for every point $p \in A$, the closest point in $A \setminus \{p\}$, where distance is measured via an L_p metric, $1 \leq p \leq \infty$. It is well known that this fundamental problem can be solved in $\Theta(N \log N)$ time serially [72].

Theorem 6.8. *Given N or fewer points in d-dimensional space, distributed one per processor on a mesh of size N with an optical pyramid, the all-nearest-neighbors problem can be solved in $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time using rN peak power, for $r \in \Omega\left(\frac{\log^8 N}{N}\right)$, where the implied constants depend upon d.*

Proof. We present the algorithm for $d = 2$. The algorithm for higher dimensions is the same, with only the various constants changing (e.g., number of slabs at each step, number of points that need to be broadcast) as functions of d. The algorithm follows the outline of solving all-nearest-neighbors on the mesh in [48].

The points are first partitioned into five disjoint, linearly separable vertical slabs, with each slab containing $N/5$ points. The all-nearest-neighbors problem is solved within each vertical slab. Likewise, the points are divided into five horizontal slabs and the problem is recursively solved in each slab. By a lemma proven in [49], there are at most 8 points in each rectangular region determined by the intersection of a vertical slab and a horizontal slab that has not determined its true closest neighbor and these 8 points can be identified efficiently. A broadcast of these 8 points from each of the 25 rectangular regions is then used to determine the true nearest neighbors of these points.

In order for the problem to be recursively solved in a square submesh for each of the slabs, the points are sorted using Hilbert curve ordering. Points are sorted by x-coordinate for vertical slabs and by y-coordinate for horizontal slabs. Determining the 8 points in each rectangular region that may not know their true nearest neighbor can be accomplished by sorting the points in each region and performing a reduction operation. We execute a divide and conquer with uniform power distribution, so the available power is divided evenly among the recursive calls so that recursive calls can be executed in parallel. Therefore, the running time obeys the recurrence $T(N') = T_{\text{sort}}(N') + 2T(\frac{N'}{5})$, where $T_{\text{sort}}(N')$ is the time to sort N' items with $S/\frac{N}{N'}$ power.

At the base case of our algorithm when the power is $\log^2 N$, there are $\frac{S}{\log^2 N}$ submeshes running in parallel, each of size $\frac{N \log^2 N}{S}$, with $\log^2 N$ available peak power per submesh. Now, a serial algorithm is simulated to solve the all-nearest-neighbors problem. Since the optical pyramid reduces the communication diameter of the mesh to the logarithm of the

size of the mesh, a serial algorithm on an input of size M can be simulated on a mesh of size M with $O(\log M)$ overhead. Thus the base case is $T(\frac{N\log^2 N}{S}) \in O\left(\frac{N\log^2 N}{S}\log^2(\frac{N\log^2 N}{S})\right)$.

Since there is $\Omega(\log^8 N)$ peak power, $T_{\text{sort}}(N') \in \Theta\left(\sqrt{\frac{N'N}{S}}\right)$. Therefore, the total running time is $\Theta(N/\sqrt{S}) = \Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$. □

6.7 Minimum Spanning Forest

The following theorem comes from simulating the minimum spanning forest algorithm on the pyramid in [49].

Theorem 6.9. *Given the $n \times n$ weighted adjacency matrix of an undirected graph stored one element per processor on a mesh of size N with an optical pyramid, the connected components and a minimum spanning forest can be determined in $O(\frac{1}{r}\log N + N^{1/4})$ time.*□

This shows that adding an optical pyramid to the mesh improves the running time for higher amounts of peak power over a standard mesh for finding a minimum spanning forest. For the minimum spanning forest problem on a standard mesh, [65] gives $\Theta(\frac{N\log N}{S})$ time for $S \in O(N^{1/2}\log N)$ and $\Theta(N^{1/2})$ time for all larger S. In contrast, the optical pyramid can efficiently utilize power as long as $S \in O(N^{3/4}\log N)$.

Often, parallel algorithms for a graph given as an adjacency matrix are faster than those for when they are given as a set of edges, and this holds true for finding a minimal spanning forest on a mesh with optics. However, for large graphs or sparse graphs, a more natural input format is to be given the graph as a set of edges.

Theorem 6.10. *Given N weighted edges of an undirected graph arbitrarily distributed one edge per processor on a mesh of size N with an optical pyramid, a minimum spanning forest can be determined in $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time using rN peak power, for $r \in \Omega\left(\frac{\log^8 N}{N}\right)$.*

Proof. For simplicity, each edge is represented twice so that an edge between vertices u and v is stored in one processor as (u,v) and in another as (v,u). Also assume that every vertex has an edge to itself as a way of ensuring it is represented.

The algorithm uses a series of recursive steps, commonly called *Borůvka steps*, where for each vertex an incident edge of smallest weight is selected. The resulting subgraph consists of edges in the minimum spanning forest, and they form trees which are *supervertices*, i.e., vertices for the following stages. For each tree, one of the vertices is chosen and its label becomes the label for the supervertex. Then some of the original edges in the graph become edges between supervertices, where the edge between supervertices U and V is the

one having minimal weight among all edges connecting a vertex in U with one in V. This is known as *vertex reduction*.

1. Do vertex reduction five times. The number of vertices is now no more than $1/32$ of the original number.

2. In each quadrant of the mesh, recursively solve the problem using only edges in the quadrant. The number of edges selected in each quadrant is proportional to the number of supervertices, so for all the quadrants combined, the number of edges is at most $4*(1/32) = 1/8$ the number of original vertices.

3. Move these edges to a submesh of size $N/8$, and recursively solve the problem in this submesh. This uses the fact that a minimum spanning forest of the entire graph is a minimum spanning forest of the union of the subgraphs.

We divide and conquer using uniform power distribution. If T_{MSF} is the time to find a minimum spanning forest, T_{VR} is the time to do a vertex reduction, and T_{sort} is the time to sort (to move edges to a submesh of size $N/8$), then

$$T_{\text{MSF}}(N,S) = 5T_{\text{VR}}(N,S) + T_{\text{MSF}}(N/4,S/4) + T_{\text{MSF}}(N/8,S) + T_{\text{sort}}(N,S)$$

Vertex reductions are done recursively, using upward tree reductions at each step, which are themselves done recursively. In an upward tree reduction, there is a directed tree with edges pointing toward the root. Each vertex has a value, and the result of the value is a semigroup operation applied to all of these values. See [7,49,64] for an explanation of how these operations are used.

If T_{UT} is the time for doing upward tree reduction, then

$$T_{\text{VR}}(N,S) = T_{\text{UT}}(N,S) + T_{\text{VR}}(N/2,S) + T_{\text{sort}}(N,S)$$

where

$$T_{\text{UT}}(N,S) = T_{\text{sort}}(N,S) + T_{\text{UT}}(N/4,S/4)$$

Similar to the all-nearest-neighbors algorithm, before the power available to a recursive call becomes too small (less than the square of the logarithm of the size of mesh), a serial algorithm is simulated to solve the lowest levels of recursion. Since a minimum spanning forest can be computed in $O(N \log N)$ time serially, $T_{\text{MSF}}(N,S) \in \Theta(N/\sqrt{S}) = \Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$. □

Given a tree, it is possible to determine a rooted tree and the preorder numbering of its vertices in the same time and energy as sorting. The following seems to be a known result, but it does not seem that a proof is explicitly stated anywhere.

Lemma 6.11. *Given N edges of a tree stored one per processor in an optical pyramid, the preorder numbering of the vertices in the tree can be computing in the same time and energy as sorting.*

Proof. The algorithm follows by combining two algorithms:

1. Use the CHAIN-RANK algorithm of Attalah and Hambrusch [6] to compute a rooted tree and the number of descendants of each vertex.

2. Use the string formation algorithm of Stout [62] to order vertices by their preorder numbering.

The dominating running time of these two steps both come form sorting, which gives us the desired running time. □

Using this lemma and the finding the minimum spanning forest of a graph is often a key step in many other graph algorithms. Algorithms for finding connected components, biconnected components, bridge edges, and articulation points follow [6, 7].

Corollary 6.12. *The connected components, biconnected components, bridge edges, and articulation points of a graph with N edges can be found in* $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ *time using rN peak power, for* $r \in \Omega\left(\frac{\log^8 N}{N}\right)$*, on a mesh of size N with an optical pyramid.*

Proof. The algorithms are not completely trivial, but they follow known algorithms while using the tree algorithms in [49] for some operations. A short description of these algorithms on a single connected component of the graph follows.

Checking if a graph is bipartite is the easiest. After a rooted spanning tree is found, each vertex i determines $depth(i)$, its distance from the root in the spanning tree in the $N^{1/4} \times N^{1/4}$ optical mesh. This information is then distributed to the processors in the base so that each processor $P(i, j)$ that contains an edge (i, j) knows $depth(i)$ and $depth(j)$. The graph is bipartite if for every edge (i, j), the parity of $depth(i)$ is not equal to the parity of $depth(j)$, which can be determined with a reduction operation. Note that a graph is bipartite if and only if $depth(i) - depth(j)$ is odd for every edge (i, j). The forward direction is true because otherwise, there is an odd cycle. The backward direction is true because the vertices can be partitioned into two sets, one containing vertices at an even

level and the other containing vertices at an odd level; thus, no two vertices in a set are adjacent.

Bridge edges can be computed with slight modifications to a serial algorithm [66]: after finding a rooted spanning tree and a postorder labeling, $post(v)$, for each vertex v in the tree, each vertex computes $\mathrm{ND}(v)$, the number of descendants of v (including v itself) in the $N^{1/4} \times N^{1/4}$ optical mesh. Then, using pyramid matrix read operations, each vertex v computes:

$$\begin{aligned} S_L(v) &= \min\{\{post(v) - \mathrm{ND}(v) + 1\} \cup \{post(w) | (v,w) \text{ is an edge}\}\} \\ S_H(v) &= \max\{\{post(v)\} \cup \{post(w) | (v,w) \text{ is an edge}\}\} \end{aligned}$$

Then, $L(v)$ and $H(v)$ are computed, where $L(v) = \min\{S_L(w) | w \text{ is a descendant of } v\}$ and $H(v) = \min\{S_H(w) | w \text{ is a descendant of } v\}$. As Tarjan showed, an edge (v,w) directed away from the root of the spanning tree is a bridge edge if and only if $H(w) \leq post(w)$ and $L(w) > post(w) - \mathrm{ND}(w)$. All of these calculations use mesh algorithms from [49]. Computing articulation points is similar. □

6.8 Convex Hull

The convex hull of a set of points U is the smallest convex polygon P for which each point of U is in the interior or on the boundary of P. A point $p \in U$ is an extreme point if and only if p is not in the convex hull of $U \setminus \{p\}$.

It is known that the extreme points of a set of N points can be found in $\Theta(N \log N)$ time on a serial computer [8, 26] and $\Theta(\log N)$ time on an EREW PRAM [47].

We present the following theorem for points in two-dimensional space.

Theorem 6.13. *Given a set U of N or fewer planar points, distributed one per processor on a mesh of size N with optical pyramid, the extreme points of U can be identified in $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time using rN peak power for $r \in \Omega\left(\frac{\log^8 N}{N}\right)$.*

Proof. The algorithm to determine the extreme points follows a divide-and-conquer strategy [48] with uniform power distribution as the previous algorithms. The following steps are executed:

1. Sort the points by x-coordinate (where ties are broken by y-coordinate) into a Hilbert curve.

2. Recursively solve the four quadrants of the mesh in parallel using one quarter of the available power for each quadrant.

3. Determine the extreme points using the extreme points from the recursively solved quadrants.

The base case of the recursive algorithm occurs when the size of the problem being solved is $O(\frac{1}{r}\log^2 N)$. At this point a serial algorithm is simulated, which takes at most $O\left(\frac{1}{r}\log^2 N \log^2(\frac{1}{r}\log^2 N)\right)$ time.

In order to combine the solutions of the four quadrants into one, we use the fact that the convex hull of the union of two sets of points is the convex hull of the union of their convex hulls. Note that the points in each quadrant are linearly separable. The extreme points of the union of the first two and last two quadrants are found, then the extreme points of the whole mesh is found.

Assuming we want to find the convex hull of the union of two sets of points where all the points in the convex hull of one set are to the left of all the points in the other set. Call the left set of points A and the other set of points B. To find the convex hull of the union, it is enough to determine the two points in each set that make up the tangent line segments that connect the two convex hulls.

Given the extreme points of each of the two sets in clockwise order, a binary search can be performed to find the upper and lower tangent lines. Initially, the leftmost and rightmost points of A are examined. For the points between the leftmost and rightmost points in clockwise order, a binary search starting from a point between the leftmost and rightmost points for the point where the line through that point and its predecessor is above all points in B and the line through that point and its successor is not. Similarly, this method is used to find the bottom tangent point. All this requires an initial sort to determine the clockwise ordering of points and a logarithmic number of broadcasts and reductions.

The analysis is similar to the previous sections, so in total, this takes $\Theta(N/\sqrt{S}) = \Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time. □

6.9 Intersection Detection of Geometric Objects

The following results on detecting an intersection among geometric objects use ideas from the plane-sweep tree technique of [5], which builds off initial work from [1] and a serial line segment intersection detection algorithm [18]. The definition of the plane-sweep tree data structure is given. Note that these algorithms only detect whether any intersections exists and reports one if there exists an intersection. It is not possible to enumerate all possible intersections in the given time as N objects may have $\Theta(N^2)$ intersections in total.

We will assume that no two endpoints share the same x-coordinate. The cases where

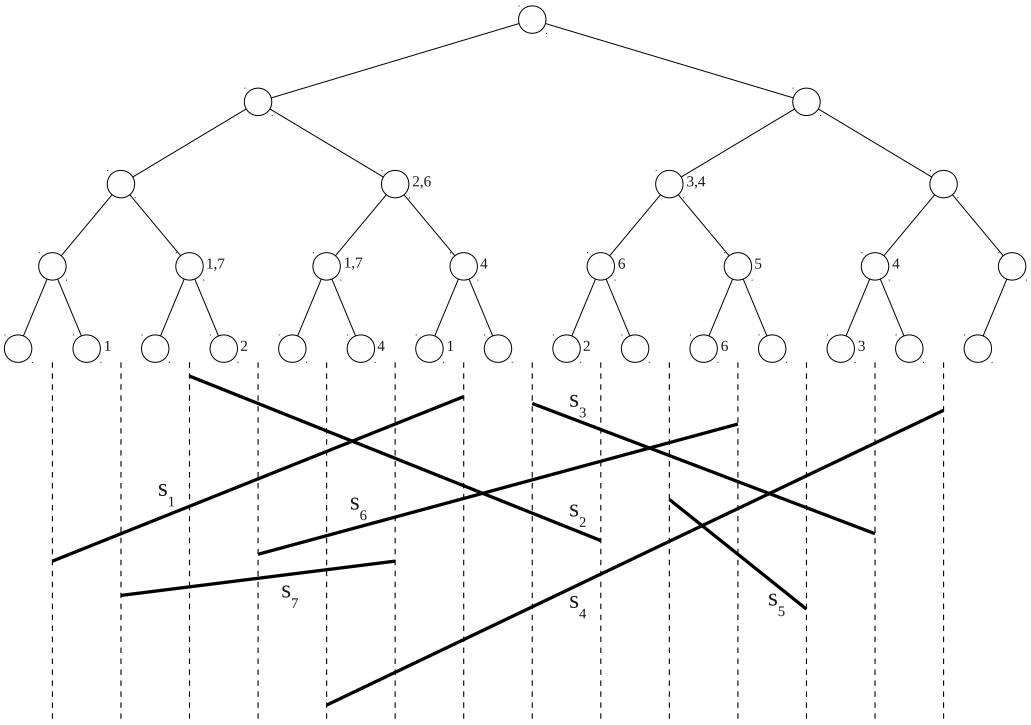


Figure 6.8: A plane-sweep tree of 7 line segments. Nodes are labeled with segments it is covered by.

there are endpoints that share an x-coordinate are easily handled, but require special treatment.

A plane-sweep tree T (Figure 6.8) is a tree that stores information about a set of line segments in the plane. Given a set of line segments $S = \{s_1, \ldots, s_N\}$, with none of the endpoints sharing an x-coordinate, T is a complete binary tree where each leaf node corresponds to one of the $2N+1$ intervals formed by projecting the endpoints to the x-axis. Each non-leaf node $v \in T$ corresponds to an interval $[a_v, b_v]$, which is the union of the intervals that correspond to the descendants of v. Let $\Pi_v = [a_v, b_v] \times (-\infty, \infty)$. A segment s_i *covers* a node $v \in T$ if it spans Π_v but not Π_z, where z is the parent of v. For each node $v \in T$, define $H(v) = \{s_i : s_i \text{ covers } v\}$ and $W(v) = \{s_i : s_i \text{ has at least one endpoint in } \Pi_v\}$.

The general idea behind using the plane-sweep tree is to check if there are any intersections among $H(v) \cup W(v)$ for each node v. A key result from [18] says that in order to detect an intersection in S, it is enough to check, for all $v \in T$, that there are no intersections between line segments in $H(v)$ and also that there are no intersections between a line segment in $W(v)$ and a line segment in $H(v)$.

While the whole plane-sweep tree may be too large to fit in the mesh at one time, each level of the tree contains $O(N)$ data and so we process the tree one level at a time.

In addition, the information at each level can be computed using only the level below it (specifically, using only its children and sibling's children), thus, the tree can be computed bottom-up level-by-level.

To apply this algorithm to the mesh with an optical pyramid, the line segments are initially sorted by x-coordinate and arranged along a Hilbert space-filling curve so that as nodes in T are computed, the line segments corresponding to each node will be in a submesh proportional to its size without having to redistribute the data again. Divide and conquer with uniform power distribution is used. Note that the space required for each submesh is bounded by a constant multiple of the size of the subtree contained in it, so it is sufficient to distribute space evenly among the submeshes for a given level.

Using the plane-sweep tree, we can find an intersection in a set line segments.

Theorem 6.14. *Given N line segments in the plane, distributed one per processor on a mesh of size N with an optical pyramid, in $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time it can be determined if any two intersect using rN peak power, for $r \in \Omega\left(\frac{\log^8 N}{N}\right)$.*

Proof. A plane-sweep tree of the N line segments is built bottom-up. $H(v)$ and $W(v)$ are computed for each node v in the plane-sweep tree. An intersection in $H(v)$ can be detected by simply sorting the line segments by their y-coordinate at v_a and sorting by their y-coordinate at v_b and comparing the two sorted lists. To determine if any segment in $W(v)$ intersects with a segment in $H(v)$, a binary search is performed in $H(v)$ for each line segment in $W(v)$

As the computations for each node in T require sorting, we use the divide and conquer with uniform power distribution paradigm. Note that the bottom $\lg\left(\frac{N\lg^2 N}{S}\right)$ levels of the tree are composed of $\frac{S}{\lg^2 N}$ separate subtrees of size $O\left(\frac{N\log^2 N}{S}\right)$. It is sufficient to take the union of all the line segments to be considered in each node and check if there are any intersections among them instead of computing each node in each subtree individually. The line segments to be checked for intersections in a subtree rooted at node v are $A(v) = \{s_i : s_i \text{ covers } v \text{ or } s_i \text{ has at least one enpoint in } \Pi_v\}$. Note that the number of line segments considered for each subtree is proportional to the size of the subtree. An intersection can be detected in a set of N line segments in $O(N\log N)$ time on a serial computer [59], so this can be simulated with $O\left(\log\left(\frac{N\log^2 N}{S}\right)\right)$ overhead with one unit of power. Therefore, these first $\lg\left(\frac{N\lg^2 N}{S}\right)$ levels can be computed in $O\left(\frac{N\log^2 N}{S}\log^2(\frac{N\log^2 N}{S})\right)$ time, which is $o\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$.

If no intersections are found in the bottom levels, the algorithm continues using standard merging and sorting operations to compute the remaining levels up the tree. Since divide

and conquer with uniform power distribution is used, this takes $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ time. □

The plane-sweep tree technique used in line segment intersection detection can also be used to detect an intersection in a set of circles or an overlap in a set of disks.

Theorem 6.15. *Given N circles or disks, distributed one per processor on a mesh of size N with an optical pyramid, in* $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ *time it can be determined if any two intersect or overlap using rN peak power, for* $r \in \Omega\left(\frac{\log^8 N}{N}\right)$.

Proof. Finding an intersection among circles will be described. Finding an overlap among disks uses the same idea.

The input is first transformed into a set of line segments. Construct a set S of line segments by taking each circle with center (x,y) and diameter d and adding to S a line segment with endpoints $(x-\frac{d}{2},y)$ and $(x+\frac{d}{2},y)$. The plane sweep tree is constructed using the line segments in S. Once the tree is constructed, it is enough to check, for all v in the tree, whether there are any intersections between the circles corresponding to the line segments in $H(v)$ and if there are any intersections between circles in $W(v)$ and $H(v)$. The algorithm is the same as the line segment intersection detection algorithm except for a change in how intersections are checked for each node v.

In order to detect intersections of circles in $H(v)$, it is enough to first sort the circles by y-coordinate, then determine if any pair of successive circles intersect. To detect an intersection between a circle in $W(v)$ and a circle in $H(v)$, for each circle in $W(v)$, it is enough to determine which circles in $H(v)$ are directly above and below it by y-coordinate, then determining if they intersect.

Note that an intersection among N circles can be detected in $\Theta(N\log N)$ time using serial sweep-line algorithm and so the running time recurrence still holds.

If any two circles intersect, their intersection will be detected by this algorithm. To see this, consider a point of intersection between two circles a and b. Let v be the node in T such that the intersection point is contained in the interval corresponding to v and a is in $H(v)$. Similarly, let w be the node corresponding to b. If $v = w$, then the intersection will be detected when scanning for intersections in $H(v)$. Otherwise, one of the nodes must be a descendant of the other as they both contain the point of intersection. In this case, the intersection will be found when determining intersections between circles in $H(v)$ and $W(v)$ (or $H(w)$ and $W(w)$ if v is a descendant of w). □

Another use of the plane-sweep tree is to detect an overlap among a set of polygons.

Theorem 6.16. *Given N line segments forming a set of simple polygons, distributed one per processor on a mesh of size N with an optical pyramid, in* $\Theta\left(\frac{1}{\sqrt{r}}\sqrt{N}\right)$ *time it can be determined if any two polygons overlap using rN peak power, for* $r \in \Omega\left(\frac{\log^8 N}{N}\right)$.

Proof. For simplicity, we assume that no two polygons share a common x-coordinate in any of their vertices. Extending this to the general case can be done but requires extra detail. In the case where there are two edges that intersect, this can be detected by just running the line segment intersection algorithm on the set of edges without their endpoints. Therefore, we assume that none of the edges intersect. This implies that if a pair of overlapping polygons exists, one must be contained in the other. We can also assume that the polygons are labeled and each edge knows the label of the polygon it is in. If not, the edges can be labeled using the minimum spanning forest algorithm to find the connected components.

The plane-sweep tree data structure containing the edges is constructed. To check for overlaps in $H(v)$, the edges are sorted by y-coordinate and then the their respective polygons are checked for intersections. It is enough to check that every successive pair of edges belongs to the same polygon because otherwise, one polygon will be contained in another according to the Jordan curve theorem. This same test can be done to check for overlaps with $W(v)$ as well. □

6.10 Pyramid Read

In some situations, the order of data accesses is determined during the execution of the algorithm and initially sorting all the data is not possible. A *pyramid read* of N^α items, for $0 \leq \alpha \leq 1$, is the operation in which up to N^α processors in the base mesh have to send data to level $(1-\alpha)\log_4 N$ of the pyramid.

If $\alpha = 0$, then only one value is being read, which takes $\Theta(\log N)$ time. If $\alpha = 1$, then all the movement is in the base and takes $\Theta(\sqrt{N})$ time if each processor must read an arbitrary value and the mesh is fully powered. The following theorem considers the case of $0 < \alpha < 1$.

Theorem 6.17. *On a mesh of size N with an optical pyramid, a pyramid read of* N^α *values, for constant* α, $0 < \alpha < 1$, *can be performed in* $O(N^{\alpha/2}\sqrt{\log N})$ *time using* N^α *power and requires* $\omega\left(N^{\alpha/2}\sqrt{\log N}\Big/\log\log N\right)$ *time no matter what power is available.*

Proof. The proof of the lower bound is given first. To simplify notation, $P_{i,j}$ denotes the processor at coordinates (i, j) within a specific level of the pyramid.

For the sake of contradiction, assume there is an algorithm to perform a pyramid read in $T = \frac{cN^{\alpha/2}\sqrt{\lg N}}{\lg\lg N}$ time, for some constant c. Further, assume that N^α data stored in processors

$P_{i,j}$, for $0 \leq i,j \leq N^{\alpha/2}-1$, in the base of the pyramid is being requested by the pyramid read operation.

Let $W = \lg\lg N + \lg c + 1 - \lg\lg\lg N$ and let M_k be the $2T \times 2T$ submesh consisting of processors $P_{i,j}$, for $0 \leq i,j \leq 2T-1$ at level $(1-\alpha)\log_4 N - kW$, for $1 \leq k \leq (1-\alpha)\log_4 N/W$. W is the difference in levels between successive levels containing M_k. Note that processors in the same level but not in M_k are not used in the algorithm, as the distance to these processors from the values being read in the base of the pyramid is greater than T.

Consider the path each value must take to reach the processor in level $(1-\alpha)\log_4 N$ of the pyramid that requested it. The path of each value must include at least one processor in each M_k.

Let $R = \frac{4cN^{\alpha/2}}{(1-\alpha)\sqrt{\lg N}}$ and let M'_k be the $R \times R$ submesh of M_k consisting of processors $P_{i,j}$, for $0 \leq i,j \leq R-1$. Let x_k be the number of values with paths that use processors in the same level as M'_k, but not in M'_k. Then $x_k \geq N^\alpha - R^2 - 4RT$ because there are at least $N^\alpha - R^2$ values with a destination not in the pyramid with M'_k as the base and the bandwidth to get out of that pyramid is at most $4R$. Therefore, at most $4RT$ of these items could have stayed inside of the pyramid at that level.

As k is at most $(1-\alpha)\log_4 N/W$, there is at least one value which takes a path that uses processors in the same level but not in M'_k, for at least half of the possible values of k. The distance between a processor in $M_k \setminus M'_k$ and $M_{k'} \setminus M'_{k'}$ for $k' \neq k$ is at least $R - R/2^W$. This implies that value moves at least $R - R/2^W$ steps at least $\frac{1}{2} \cdot \frac{(1-\alpha)\log_4 N}{W}$ times, which is $\left(1 - \frac{\lg\lg N}{2c\lg N}\right) \frac{cN^{\alpha/2}\sqrt{\lg N}}{\lg\lg N + \lg c + 1 - \lg\lg\lg N}$ total steps, which is a contradiction because this is greater than $T = \frac{cN^{\alpha/2}\sqrt{\lg N}}{\lg\lg N}$, for N sufficiently large.

Now an $O(N^{\alpha/2}\sqrt{\log N})$ time algorithm is presented. Without loss of generality, it is enough to show there is an algorithm that moves N^α values located in the base of the pyramid to one per processor in level $(1-\alpha)\log_4 N$ of the pyramid. Once the data is spread out one per processor at level $(1-\alpha)\log_4 N$, a mesh concurrent read operation can be performed in $O(N^{\alpha/2})$ time to get values to the processors that requested them.

First, given $N^\alpha/\lg N$ values stored one per processor in a mesh of a size at least $\frac{2N^\alpha}{\sqrt{\lg N}} \times \frac{2N^\alpha}{\sqrt{\lg N}}$, it is possible to spread out the values such that there is at most one value in every 2×2 submesh in $O(N^{\alpha/2}/\sqrt{\lg N})$ time. To do this, divide the mesh into $\frac{2N^{\alpha/2}}{\sqrt{\lg N}} \times \frac{2N^{\alpha/2}}{\sqrt{\lg N}}$ submeshes. Each submesh then spreads the data out using standard mesh movement operations in $\Theta(N^{\alpha/2}/\sqrt{\lg N})$ time. This is possible because there are at most $N^\alpha/\lg N$ values total.

The algorithm divides the N^α values into $\lg N$ groups of $N^\alpha/\lg N$ values each. Each group has a designated $\frac{N^{\alpha/2}}{\sqrt{\lg N}} \times \frac{N^{\alpha/2}}{\sqrt{\lg N}}$ submesh of level $(1-\alpha)\log_4 N$ of the pyramid as its

destination. These groups are moved up the pyramid, taking $O(N^{\alpha/2}/\sqrt{\log N})$ time per level, with groups moving in a pipelined manner so that all groups reach their respective destinations in $O(N^{\alpha/2}\sqrt{\log N})$ time even though each of the $\lg N$ groups individually takes $O(N^{\alpha/2}\sqrt{\log N})$ time to reach its destination.

To partition the values into groups, the base of the pyramid is partitioned into $\frac{N^\alpha}{\sqrt{\lg N}} \times \frac{N^\alpha}{\sqrt{\lg N}}$ submeshes. Values are grouped in an arbitrary manner within each $\frac{N^\alpha}{\sqrt{\lg N}} \times \frac{N^\alpha}{\sqrt{\lg N}}$ submesh and in row-major order over the submeshes. The destination $\frac{N^{\alpha/2}}{\sqrt{\lg N}} \times \frac{N^{\alpha/2}}{\sqrt{\lg N}}$ submesh of a group at level $(1-\alpha)\log_4 N$ is assigned by row-major order. The ordering ensures successful pipelining of groups to get to their destination after reaching level $(1-\alpha)\log_4 N$.

Each group moves up the pyramid in the following manner. At each level, the values in the group spread out so that at most one value is in each 2×2 submesh in $O(N^{\alpha/2}/\sqrt{\log N})$ time. Then, all the values in the group move up one level to the processor directly above their current locations in one time step. Once a group reaches level $(1-\alpha)\log_4 N$, values are moved to their destination submesh in a pipelined manner. Since there are $(1-\alpha)\log_4 N$ levels, the movement of $\lg N$ groups can be pipelined over the levels to achieve a total time of $O(N^{\alpha/2}\sqrt{\log N})$. □

Note that the pyramid read algorithm takes $\Theta(N^{3\alpha/2}\sqrt{\log N})$ energy, as all values are moving in parallel. In general, if each of the N values in the base of a pyramid must be read once and it is possible to read at least $S = N^\alpha$ in parallel at a time, then sequentially executing a pyramid read for S values at a time uses only a factor of $O(\sqrt{\log N})$ more time and energy than a single sort performed on all N data.

A similar lower bound on time is shown in [49], except that the lower bound is shown for moving data that is structured in a matrix. A pyramid read operation examines data stored at arbitrary locations whereas that result states that moving M words of data from the first column of the base of the pyramid to the last column takes $\Omega(\log N + M^{1/2})$ time.

The next section is an application of the pyramid read operation.

6.10.1 Stepwise Simulation of PRAMs

Through simple stepwise simulation, any CRCW PRAM algorithm using p processors and $O(p)$ memory can be simulated on a $\sqrt{p} \times \sqrt{p}$ mesh with $\sqrt{p}$ overhead per times step. In the case where the memory size is greater than the number of processors, CRCW PRAM algorithms can be simulated on the pyramid using pyramid read operations. This fact can sometimes be useful when trying to implement or convert PRAM algorithms to the pyramid computer model.

Proposition 6.18. *A CRCW PRAM with p processors and $O(p^k)$ memory, for some constant $k > 1$, can be stepwise simulated with $\Theta(\sqrt{p \log p})$ overhead per step on a mesh of size $O(p^k)$ with an optical pyramid, where the implied constants depend on k.*

Proof. The idea is that the level with the $\sqrt{p} \times \sqrt{p}$ mesh acts as the processors and the base of the pyramid acts as the memory. Each memory access in a CRCW PRAM can be simulated using a pyramid read at the base from the $\sqrt{p} \times \sqrt{p}$ mesh level. A reverse of the pyramid read is also needed so that processors can locate the memory being read. □

6.11 A Planar Separator Algorithm

A planar graph is a graph that can be drawn on the plane so that none of its edges cross. It was shown by Lipton and Tarjan [42] that for every planar graph with N vertices, there exists a set of $O(\sqrt{N})$ vertices that partitions the graph into two disconnected parts such that each part has at most $\frac{2}{3}N$ vertices. Given a list of the edges in a planar graph, they also showed that a planar separator can be found in $O(N)$ time on a serial computer. This can be generalized to the case where each vertex has an associated nonnegative cost and the separator of $O(\sqrt{N})$ vertices partitions the graph into parts, each of total cost at most two-thirds times the total cost of the graph. Such a separator is called a $\frac{1}{3}$-$\frac{2}{3}$ separator.

The Gazit-Miller planar separator algorithm (see [70]) is an algorithm that finds a planar separator of a graph with N vertices on an EREW PRAM in $O(\sqrt{N} \log N)$ time using $\sqrt{N}$ processors. The following algorithm is based on theirs, with some modifications for the pyramid model which include use of the pyramid read operation.

Theorem 6.19. *Given N edges of a planar graph $G = (V, E)$ with nonnegative vertex costs, stored one edge per processor on a mesh of size N with an optical pyramid, a planar separator can be found in $O(\frac{1}{r}N^{1/4}\sqrt{\log N} + N^{3/4}\sqrt{\log N})$ time using rN peak power.*

Proof. The dominant running time of algorithm comes from what Träff and Zaroliagis call an augmented breadth-first search. This search can be shown to take $O(N^{5/4}\sqrt{\log N}/S + N^{3/4}\sqrt{\log N})$ time in the optical pyramid model due to a series of successive pyramid read operations. In the worst case, a pyramid read of $O(\sqrt{N})$ vertices is executed for each of the $O(\sqrt{N})$ augmented breadth-first search levels, which gives $O(N^{3/4}\sqrt{\log N})$ time when power is maximized.

Define $wt(V)$ as the sum of the costs of the vertices in V. Without loss of generality, assume that $wt(V) = 1$. The output of the algorithm is a partition of V into three sets V_1, V_2, U, where $|U| \in O(\sqrt{N})$ is a separator of G and $wt(V_1), wt(V_2) \leq 2/3$.

A tree almost like a breadth-first search tree is constructed and each vertex is assigned a level. There exists a processor for each vertex that keeps track of whether the vertex has been assigned a level yet and the level it was assigned. Define $V(\ell)$ to be the set of vertices at level ℓ. Vertices will be assigned levels such that each $V(\ell)$ is a separator of G. The Gazit-Miller algorithm computes three levels, ℓ_0, ℓ_1, and ℓ_2, such that the following properties hold:

- $\ell_0 \leq \ell_1 < \ell_2$
- $wt(\bigcup_{\ell<\ell_1} V(\ell)) < 1/2$
- $|V(\ell_0)| \leq 2\sqrt{|V|}$
- $|V(\ell_2)| \leq \sqrt{|V|}$
- $\ell_1 - \ell_0 < \sqrt{|V|}$
- $\ell_2 - \ell_1 < \sqrt{|V|}$

Define three sets of vertices: $A = \bigcup_{\ell<\ell_0} V(\ell)$, $B = \bigcup_{\ell_0<\ell<\ell_2} V(\ell)$, and $C = \bigcup_{\ell>\ell_2} V(\ell)$. In the algorithm, a graph G_B, will be constructed by removing the vertices $C \cup V(\ell_2)$ from G and shrinking all vertices in $A \cup V(\ell_0)$ to a single vertex of cost 0. Given the endpoints of each edge marked with the set it is in, G_B can be computed using constant energy per edge and $O(N/S)$ total time.

Intuitively, this is similar to the original Lipton-Tarjan algorithm, where ℓ_1 is a middle level that most closely separates the graph into two parts of equal weight and ℓ_0 and ℓ_2 are levels that contain $O(\sqrt{N})$ vertices. In the case where neither $V(\ell_1)$ nor $V(\ell_0) \cup V(\ell_2)$ is a proper separator, a separator of just the vertices between ℓ_0 and ℓ_2 is found, which will be sufficient. The difference with the Gazit-Miller algorithm is that the levels that are computed may not be the same.

An array X initialized with $|V|$ elements is used as a stack-like data structure supporting push and pop operations that can push or pop multiple elements at once. These operations can be efficiently implemented on a pyramid. To do this, the array is stored in row-major order in the mesh and one processor keeps a record of which processor contains the current top element of the stack. Since the elements are stored in row-major order, a push or pop operation of x elements to another array in row-major order takes $O(x/\sqrt{S} + \sqrt{x} + \log N)$ time. Over the whole algorithm, $\sqrt{|V|}$ elements are popped off X $O(\sqrt{N})$ times; therefore, this contributes only $O(N/\sqrt{S} + N^{3/4})$ time to the algorithm.

Similarly, a scan over x elements indexed in row-major order is accomplished either by computing the scan of S elements at a time or moving elements to the $\sqrt{x} \times \sqrt{x}$ mesh

and computing the scan there, which takes $O(x/\sqrt{S}+\sqrt{x}+\log N)$ time. Like the stack operations, over the course of the algorithm, scans contribute only $O(N/\sqrt{S}+N^{3/4})$ time to the algorithm.

The following is the full algorithm as stated in [70]. The algorithm depends on three phases. Initialization occurs in INITIALIZATION PHASE, an augmented breadth-first search is performed in PHASE A to compute ℓ_0 and ℓ_1, and a breadth-first search is performed in PHASE B to compute ℓ_2.

Run INITIALIZATION PHASE
Run PHASE A
if $|V(\ell_1)| \leq 7\sqrt{|V|}$ **then**
    $U = V(\ell_1)$; $V_1 = \bigcup_{\ell<\ell_1} V(\ell)$; $V_2 = V - V_1 - U$;
else
    Run PHASE B;
end if
Let A, B, C, and G_B as defined previously;
if $wt(B) \leq 2/3$ **then**
    $U = V(\ell_0) \cup V(\ell_2)$; $V_1 = \max_{wt}\{A,B,C\}$; $V_2 = (A \cup B \cup C) - V_1$;
else
    Find a $\frac{1}{3}$-$\frac{2}{3}$ separator in G_B yielding partition W_1, W_2, U', $|U'| \leq 4\sqrt{|V|}$,
        and $wt(W_1) \geq wt(W_2)$;
    $U = V(\ell_0) \cup V(\ell_2) \cup U'$; $V_1 = W_1$; $V_2 = A \cup C \cup W_2$;
end if

Other than the three phases, we need to explain how the $\frac{1}{3}$-$\frac{2}{3}$ separator in G_B is found. While there are sorting-dependent mesh algorithms [33] that compute this in the same time as sorting N items, we can also simply say that the PRAM algorithm given in [46] using $\sqrt{N}$ processors can be simulated using pyramid reads in $O(N^{5/4}\sqrt{\log N}/S)$ time.

The implementation of the three phases follows.

INITIALIZATION PHASE:
Find a spanning tree T of G rooted at an arbitrary vertex s;
Compute the preorder numbering, $pre(\cdot)$, of the vertices in T;
for all $v \in T$ **do in parallel** $X[pre(v)] = v$;

All the steps of INITIALIZATION PHASE take $O(N/\sqrt{S})$ time.

PHASE A:

$\ell = 0$; $V(0) = \{s\}$;
while $wt(\bigcup_{\ell' < \ell} V(\ell')) < 1/2$ **do**
NEXT-LEVEL$(\ell, V(\ell))$;
$\ell = \ell + 1$; $j = 2\ell + 1$;
while $|V(\ell)| < j$ **and** $A \neq \varnothing$ **do**
Pop the top $\sqrt{|V|}$ vertices from X into X';
Mark vertices in X' that belong to any previous level $i < \ell$;
Remove marked vertices from X';
$R =$ number of remaining vertices of X';
$\rho = \min\{j - |V(\ell)|, R\}$;
Add the first ρ elements of X' to $V(\ell)$ and push the remaining $R - \rho$ back onto X;
end while
if $|V(\ell)| < 2\sqrt{|V|} + 1$ **then**
$\ell_0 = \ell$;
end if
end while
$\ell_1 = \ell$;

Removing vertices from X' uses a scan operation. A count of unmarked elements in an array of size $\sqrt{|V|}$ takes $O(\sqrt{|V|}/S + \log N)$ time. Checking if vertices have already been assigned a level is accomplished using a pyramid read of the $|V|$ processors that contain the information about how each vertex is labeled. It was shown in [70] that the inner while loop is executed $O(\sqrt{N})$ times; therefore, the total running time of PHASE A is $O(N^{5/4}\sqrt{\log N}/S + N^{3/4}\sqrt{\log N})$.

NEXT-LEVEL$(\ell, V(\ell))$:
Find a list of adjacent vertices for every $u \in V(\ell)$;
Remove vertices belonging to any level $i < \ell + 1$;
Assign remaining vertices level $\ell + 1$;

The procedure NEXT-LEVEL is a parallelization of one step of a breadth-first search and contributes to the dominating time of the algorithm as it uses a pyramid read operation to find adjacent vertices and assign levels. Since NEXT-LEVEL is called $O(\sqrt{N})$ times, the total running time contributed is $O(N^{5/4}\sqrt{\log N}/S + N^{3/4}\sqrt{\log N})$.

PHASE B:

$\ell = \ell_1$; $k = |\bigcup_{0 \le i \le \ell_1} V(i)|$;
while $|V(\ell)| \ge \sqrt{|V| - k}$ **do**
    NEXT-LEVEL$(\ell, V(\ell))$;
    $\ell = \ell + 1$
end while
if $V(\ell_1 + 1) = \varnothing$ **then**
    $\ell_2 = \ell_1 + 1$;
else
    $\ell_2 = \ell$;
end if

PHASE B is simply a continuation of a breadth-first search. Refer to [70] for the proof that the while loop of PHASE B iterates $O(\sqrt{N})$ times. This implies the running time of PHASE B is $O(N^{5/4}\sqrt{\log N}/S + N^{3/4}\sqrt{\log N})$. □

CHAPTER 7

Conclusion

Energy and peak power are becoming are increasingly important in parallel computing. E.g., the DOE report *Architectures and Technology for Extreme Scale Computing* [61] states:

> The primary design constraint for future HPC systems will be power consumption. ... Data movement will be a bigger factor for system energy consumption and cost than FLOP/s. ... Energy and performance costs should be reflected in abstract machine model.

Unfortunately few parallel algorithms address the energy consumption problem. It is addressed in some algorithms for sensor networks, but they are limited by the total energy available in their batteries, while parallel computers are limited by peak power which is supplied externally.

Our power aware algorithms address these issues, considering fundamental tradeoffs of time versus peak power for communication intensive problems. Our abstract model is based on ideas first expressed in von Neumann's finite autamata model which addressed physical locality and data movement. To this we added a model of on-chip optical connections, a capability which is rapidly becoming available and which offers the possibility of reducing time and/or energy. As the number of processors greatly increases, the asymptotic bounds of our algorithms are descriptive of the behavior of their running times.

The three different interconnects of the optical mesh, optical mesh of trees, and optical pyramid were presented, each with a different capability to reduce the time or energy required by the standard mesh-connected computer. The most significant results apply in the optical pyramid layout, where the optical interconnects form a pyramid which we use for problems quite unlike its previous roles in parallel computing. It is a fundamental layout appearing in VLSI design as well as being a model of parallelism studied for problems involving images, adjacency matrices, etc. [14,28,34,49,71,78]. However, simple bandwidth

arguments show that it cannot sort faster than the base mesh; therefore, the pyramid has no advantages for problems which require sorting. This suddenly changes when peak power is limited, though new approaches are needed.

We showed that the optical pyramid layout is optimal in terms of communication diameter and time-power tradeoff for sorting. Using the pyramid, we achieved a non-linear time/peak energy tradeoff, where if the peak power is cut in half then the time increases by only a factor of $\sqrt{2}$, instead of the factor of 2 that occurs with stepwise simulation. Similar results were obtained for problems where the input was an unstructured set of edges or points. The algorithms presented combine parallel divide-and-conquer approaches with stepwise simulation of serial algorithms when there is only one active processor per submesh.

For all of the optical models studied, our results explore a new perspective for modeling energy usage on massively parallel architectures and emerging capabilities. The actual implementations of these algorithms and realization of the model in hardware is another area of research, but it is abstract enough to be applied to moderately different computer architectures. For example, depending on the physical properties of the interconnection technology, it may be the case that the communication over some of the shorter optics in our model are more efficiently implemented using standard electrical wires. Nevertheless, the basic principles of routing data with numerous processors and power constraints shown in this work still hold. Further, they hold for any technology which can supply a layer of interconnections which can transmit information long distances with low power relative to standard wire interconnections.

Note that one energy-reducing hardware option, reducing the clock as the voltage is decreased, can be utilized in conjunction with our algorithms. Since the algorithms almost always have $S = rN$ processors active at any one time, one merely needs to introduce a multiplicative factor for a tradeoff of increasing peak power versus decreasing clock speed.

7.1 Future Work

The continued study of the interplay between time and energy usage of algorithms on parallel computers is necessary for the future. In [65] it is shown that, for some problems, peak power usage can be reduced, without increasing the time, on the standard mesh without optics. Depending on advancements of computer architecture and fabrication technology, we will continue to need the development of theory and models of computation.

In general, we have considered only the case where the size of the input is equal to the size of the mesh, but the algorithms can be easily extended when there is more than

one item per processor by simulating a larger size mesh. However, if there is a lot of data per processor, such as $\sqrt{N}$ per processor in a mesh of size N, then more sophisticated approaches are needed, such as running a serial algorithm part of the time. If there is less than one item per processor, time and energy decrease since less communication is required.

One open problem already mentioned was approximating Euclidean distance transforms. In addition, various other problems on this model can be studied, such as matrix multiplication, computing Voronoi diagrams, or selection. The planar separator algorithm is a very interesting result for planar graphs, a class of graphs that has not been studied before in detail for mesh-connected systems. This can be extended and applied to various other problems on planar graphs such as shortest paths. In addition, the planar separator algorithm can be used to recursively partition planar graphs so that locality of vertices in the graph is preserved in the mesh.

In general, improving running times and energy usage for problems or proving a matching lower bound is desired for all problems. Examples of results that might be improved include image component labeling and minimum spanning forest given adjacency matrix input on the different optical interconnect layouts, though for these specific problems it is likely they can only be improved by a logarithmic factor. Another example is that the lower bound running time for the pyramid read operation can likely be improved. It may also be possible to develop algorithms that sacrifice some time in order to use the minimal amount of energy possible. One example is to determine a spanning forest in linear energy and $O(N^{1/4+\varepsilon})$ time for an adjacency matrix input. Note that for serial algorithms, it is not known if there is a deterministic linear time algorithm for finding a minimum spanning tree given edge input. Perhaps there are also algorithms for random graphs that achieve linear energy. While algorithms for problems such as all-nearest-neighbors and convex hull were presented for $\Omega(\log^8 N)$ power, it is conjectured that these algorithms can be extended to match the running time of sorting for all possible values of available peak power.

Extensions to the model include analyzing algorithms on meshes of higher dimensions. A few results have been given, but there is much more to research. In three-dimensional meshes the problem of optical pathways crossing is eliminated, which allows for more optical connections and bandwidth on them. Further, the underlying three-dimensional mesh has a smaller diameter and larger bisection bandwidth than the two-dimensional mesh.

BIBLIOGRAPHY

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3(1–4):293–327, 1988.

[2] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53:86–96, May 2010.

[3] H. M. Alnuweiri and V. K. Prasanna. Parallel architectures and algorithms for image component labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:1014–1034, 1992.

[4] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[5] M. J. Atallah and M. T. Goodrich. Efficient plane sweeping in parallel. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, pages 216–225, New York, NY, USA, 1986. ACM.

[6] M. J. Atallah and S. E. Hambrusch. Solving tree problems on a mesh-connected processor array. *Information and Control*, 69(1-3):168–187, 1986.

[7] M. J. Atallah and S. R. Kosaraju. Graph problems on a mesh-connected processor array. *Journal of the ACM*, 31(3):649–667, July 1984.

[8] D. Avis. On the complexity of finding the convex hull of a set of points. *Discrete Applied Mathematics*, 4(2):81–86, 1982.

[9] B. Awerbuch and Y. Shiloach. New connectivity and MSF algorithms for shuffle-exchange network and PRAM. *IEEE Transactions on Computers*, 36:1258–1263, 1987.

[10] P. Beame. Limits on the power of concurrent-write parallel machines. *Information and Computation*, 76(1):13–28, 1988.

[11] K. Bernstein, P. Andry, J. Cann, P. Emma, D. Greenberg, W. Haensch, M. Ignatowski, S. Koester, J. Magerlein, R. Puri, and A. Young. Interconnects in the third dimension: Design challenges for 3D ICs. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 562–567, New York, NY, USA, 2007. ACM.

[12] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749. ACM, 2007.

[13] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:529–533, 1995.

[14] C. Cantoni and S. Levialdi, editors. *Pyramidal Systems for Computer Vision*, volume 25 of *NATO ASI Series F: Computer and Systems Sciences*. Springer-Verlag, 1986.

[15] J. Chan, G. Hendry, A. Biberman, and K. Bergman. Architectural exploration of chip-scale photonic interconnection network designs using physical-layer analysis. *IEEE/OSA Journal of Lightwave Technology*, 28(9):1305–1315, May 2010.

[16] F. Chang, C.-J. Chen, and C.-J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220, 2004.

[17] V. Chaudhary and J. K. Aggarwal. Parallel image component labeling for target acquisition. *Optical Engineering*, 37(7):2078–2090, 1998.

[18] B. Chazelle. Reporting and counting segment intersections. *Journal of Computer and System Sciences*, 32(2):156–182, 1986.

[19] K. W. Chong, Y. Han, and T. W. Lam. Concurrent threads and optimal parallel minimum spanning trees algorithm. *Journal of the ACM*, 48(2):297–323, Mar. 2001.

[20] R. Cypher, J. L. C. Sanz, and L. Snyder. An EREW PRAM algorithm for image component labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:258–262, 1989.

[21] P. Erdős and R. K. Guy. Crossing number problems. *The American Mathematical Monthly*, 80(1):52–58, 1973.

[22] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[23] R. Fabbri, L. da F. Costa, J. C. Torelli, and O. M. Bruno. 2D Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys*, 40(1):2:1–2:44, 2008.

[24] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[25] S. Goddard, S. Kumar, and J. F. Prins. Connected components algorithms for mesh-connected parallel computers. In S. N. Bhatt, editor, *Parallel Algorithms*, volume 30 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 43–58. American Mathematical Society, 1997.

[26] R. Graham. An efficient algorith [sic] for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[27] M. D. Grammatikakis, D. F. Hsu, M. Kraetzl, and J. F. Sibeyn. Packet routing in fixed-connection networks: A survey. *Journal of Parallel and Distributed Computing*, 54(2):77–132, 1998.

[28] R. I. Greenberg. The fat-pyramid and universal parallel computation independent of wire delay. *IEEE Transactions on Computers*, 43(12):1358–1364, 1994.

[29] M. Haurylau, G. Chen, H. Chen, J. Zhang, N. A. Nelson, D. H. Albonesi, E. G. Friedman, and P. M. Fauchet. On-chip optical interconnect roadmap: challenges and critical directions. *IEEE Journal of Selected Topics in Quantum Electronics*, 12(6):1699–1705, 2006.

[30] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, 41(7):33–38, 2008.

[31] Intel Corporation. *Intel®Turbo Boost Technology in Intel®Core™Microarchitecture (Nehalem) Based Processors*, Nov. 2008.

[32] Intel Corporation. *2nd Generation Intel®Core™vPro™Processor Family*, 2011.

[33] J. JáJá and S. R. Kosaraju. Parallel algorithms for planar graph isomorphism and related problems. *IEEE Transactions on Circuits and Systems*, 35(3):304–311, Mar. 1988.

[34] G. E. Jan, S.-W. Leu, and C.-H. Li. On the array embeddings and layouts of quadtrees and pyramids. *Journal of Information Science and Engineering*, 20(1):127–141, 2004.

[35] A. Joshi, C. Batten, Y.-J. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic. Silicon-photonic clos networks for global on-chip communication. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, pages 124–133, Washington, DC, USA, 2009. IEEE Computer Society.

[36] D. J. Kleitman. The crossing number of $K_{5,n}$. *Journal of Combinatorial Theory*, 9(4):315–323, 1970.

[37] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, and Others. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report, University of Notre Dame, CSE Dept., 2008.

[38] M. Kumar and D. S. Hirschberg. An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes. *IEEE Transactions on Computers*, 32:254–264, 1983.

[39] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal. ATAC: a 1000-core cache-coherent processor with on-chip optical network. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 477–488, New York, NY, USA, 2010. ACM.

[40] F. T. Leighton. *Complexity Issues in VLSI*. The MIT Press, 1983.

[41] M. Li and Y. Yesha. New lower bounds for parallel computation. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 177–187, New York, NY, USA, 1986. ACM.

[42] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[43] J. Liu, X. Sun, R. Camacho-Aguilera, L. C. Kimerling, and J. Michel. Ge-on-Si laser operating at room temperature. *Opt. Lett.*, 35(5):679–681, 2010.

[44] Y. Ma, S. Sen, and I. D. Scherson. The distance bound for sorting on mesh-connected processor arrays is tight. In *27th Annual Symposium on Foundations of Computer Science*, pages 255–263, Oct. 1986.

[45] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, 1990.

[46] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986.

[47] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. *IEEE Transactions on Computers*, 37(12):1605–1618, 1988.

[48] R. Miller and Q. F. Stout. Mesh computer algorithms for computational geometry. *IEEE Transactions on Computers*, 38(3):321–340, Mar. 1989.

[49] R. Miller and Q. F. Stout. *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*. The MIT Press, 1996.

[50] D. Nassimi and S. Sahni. Bitonic sort on a mesh-connected parallel computer. *IEEE Transactions on Computers*, 28:2–7, 1979.

[51] I. O'Connor and F. Gaffiot. On-chip optical interconnect for low-power. In E. Macii, editor, *Ultra Low-Power Electronics and Design*, pages 21–39. Springer US, 2004.

[52] K. Ohashi, K. Nishi, T. Shimizu, M. Nakada, J. Fujikata, J. Ushida, S. Torii, K. Nose, M. Mizuno, H. Yukawa, M. Kinoshita, N. Suzuki, A. Gomyo, T. Ishi, D. Okamoto, K. Furue, T. Ueno, T. Tsuchizawa, T. Watanabe, K. Yamada, S.-i. Itabashi, and J. Akedo. On-chip optical interconnect. *Proceedings of the IEEE*, 97(7):1186–1198, July 2009.

[53] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1):16–34, Jan. 2002.

[54] R. Ribando and K. Skadron. Many-core design from a thermal perspective. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 746–749, 2008.

[55] I. D. Scherson and S. Sen. Parallel sorting in two-dimensional VLSI models of computation. *IEEE Transactions on Computers*, 38(2):238–249, Feb. 1989.

[56] I. D. Scherson, S. Sen, and Y. Ma. Two nearly optimal sorting algorithms for mesh-connected processor arrays using shear-sort. *Journal of Parallel and Distributed Computing*, 6(1):151–165, 1989.

[57] C. P. Schnorr and A. Shamir. An optimal sorting algorithm for mesh connected computers. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 255–263, New York, NY, USA, 1986. ACM.

[58] J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In J. Palma, M. Daydé, O. Marques, and J. a. Lopes, editors, *High Performance Computing for Computational Science–VECPAR 2010*, volume 6449 of *Lecture Notes in Computer Science*, pages 1–25. Springer Berlin / Heidelberg, 2011.

[59] M. I. Shamos and D. Hoey. Geometric intersection problems. In *17th Annual Symposium on Foundations of Computer Science*, pages 208–215, Oct. 1976.

[60] Y. Song, B. Wang, Z. Shi, K. Pattipati, and S. Gupta. Distributed algorithms for energy-efficient even self-deployment in mobile sensor networks. *IEEE Transactions on Mobile Computing*, 2013.

[61] R. Stevens, A. White, and et al. Scientific grand challenges: Architectures and technology for extreme scale computing, 2009.

[62] Q. F. Stout. Topological matching. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 24–31, New York, NY, USA, 1983. ACM.

[63] Q. F. Stout. Optimal component labeling algorithms for mesh-connected computers and VLSI. *Abstracts of Papers Presented to the American Mathematical Society*, 5:148, 1984.

[64] Q. F. Stout. Tree-based graph algorithms for some parallel computers. In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 727–730. CRC Press, 1985.

[65] Q. F. Stout. Minimizing peak energy on mesh-connected systems. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 331–331, New York, NY, USA, 2006. ACM.

[66] R. E. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, Apr. 1974.

[67] R. E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.

[68] C. D. Thompson and H. T. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 2:263–271, Apr. 1977.

[69] T. Toffoli and N. Margolus. *Cellular Automata Machines: A New Environment for Modeling*. MIT Press, 1987.

[70] J. Träff and C. Zaroliagis. A simple parallel algorithm for the single-source shortest path problem on planar digraphs. In A. Ferreira, J. Rolim, Y. Saad, and T. Yang, editors, *Parallel Algorithms for Irregularly Structured Problems*, volume 1117 of *Lecture Notes in Computer Science*, pages 183–194. Springer Berlin / Heidelberg, 1996.

[71] L. Uhr, editor. *Parallel Computer Vision*. Academic Press, 1987.

[72] P. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4:101–115, 1989.

[73] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '10, pages 205–218, New York, NY, USA, 2010. ACM.

[74] Y. A. Vlasov. Silicon photonics for next generation computing systems. In *34th European Conference on Optical Communication, 2008, ECOC 2008*, pages 1–2, Sept. 2008.

[75] J. von Neumann and A. W. Burks. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.

[76] Y.-R. Wang and S.-J. Horng. An O(1) time algorithm for the 3D Euclidean distance transform on the CRCW PRAM model. *IEEE Transactions on Parallel and Distributed Systems*, 14:973–982, 2003.

[77] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[78] T. Yamada, N. Fujii, and S. Ueno. On three-dimensional layout of pyramid networks. In *APCCAS*, pages 159–164, 2002.