Introduction to Parallel Computing

Solution Manual

Ananth Grama
Anshul Gupta
George Karypis
Vipin Kumar

Contents

Preface

This instructors guide to accompany the text ”Introduction to Parallel Computing” contains solutions to selected problems.

For some problems the solution has been sketched, and the details have been left out. When solutions to problems are available directly in publications, references have been provided. Where necessary, the solutions are supplemented by figures. Figure and equation numbers are represented in roman numerals to differentiate them from the figures and equations in the text.

CHAPTER 1

Introduction

1 At the time of compilation (11/02), the five most powerful computers on the Top 500 list along with their peak GFLOP ratings are:

1. NEC Earth-Simulator/ 5120, 40960.00.
2. IBM ASCI White, SP Power3 375 MHz/8192 12288.00.
3. Linux NetworX MCR Linux Cluster Xeon 2.4 GHz -Quadrics/ 2304, 11060.00.
4. Hewlett-Packard ASCI Q - AlphaServer SC ES45/1.25 GHz/ 4096, 10240.00.
5. Hewlett-Packard ASCI Q - AlphaServer SC ES45/1.25 GHz/ 4096 10240.00.

2 Among many interesting applications, here are a representative few:

1. Structural mechanics: crash testing of automobiles, simulation of structural response of buildings and bridges to earthquakes and explosions, response of nanoscale cantilevers to very small electromagnetic fields.
2. Computational biology: structure of biomolecules (protein folding, molecular docking), sequence matching for similarity searching in biological databases, simulation of biological phenomena (vascular flows, impulse propagation in nerve tissue, etc).
3. Commercial applications: transaction processing, data mining, scalable web and database servers.

3 Data too fluid to plot.

4 Data too fluid to plot.

CHAPTER 2

Models of Parallel Computers

1 A good approximation to the bandwidth can be obtained from a loop that adds a large array of integers:

```
for (i = 0; i < 1000000; i++)
  sum += a[i];
```

with `sum` and array `a` suitably initialized. The time for this loop along with the size of an integer can be used to compute bandwidth (note that this computation is largely memory bound and the time for addition can be largely ignored).
To estimate L1 cache size, write a 3-loop matrix multiplication program. Plot the computation rate of this program as a function of matrix size n. From this plot, determine sudden drops in performance. The size at which these drops occur, combined with the data size ($2n^2$) and word size can be used to estimate L1 cache size.

2 The computation performs 8 FLOPS on 2 cache lines, i.e., 8 FLOPS in 200 ns. This corresponds to a computation rate of 40 MFLOPS.

3 In the best case, the vector gets cached. In this case, 8 FLOPS can be performed on 1 cache line (for the matrix). This corresponds to a peak computation rate of 80 MFLOPS (note that the matrix does not fit in the cache).

4 In this case, 8 FLOPS can be performed on 5 cache lines (one for matrix `a` and four for column-major access to matrix `b`). This corresponds to a speed of 16 MFLOPS.

5 For sample codes, see any SGEMM/DGEMM BLAS library source code.

6 Mean access time = $0.8 \times 1 + 0.1 \times 100 + 0.8 \times 400 \approx 50$ns. This corresponds to a computation rate of 20 MFLOPS (assuming 1 FLOP/word).
Mean access time for serial computation = $0.7 \times 1 + 0.3 \times 100 \approx 30$ns. This corresponds to a computation rate of 33 MFLOPS.
Fractional CPU rate = $20/33 \approx 0.60$.

7 Solution in text.

8 Scaling the switch while maintaining throughput is major challenge. The complexity of the switch is $O(p^2)$.

9 CRCW PRAM is the most powerful because it can emulate other models without any performance overhead. The reverse is not true.

10 We illustrate the equivalence of a butterfly and an omega network for an 8-input network by rearranging the switches of an omega network so that it looks like a butterfly network This is shown in Figure 2.1 [Lei92a].

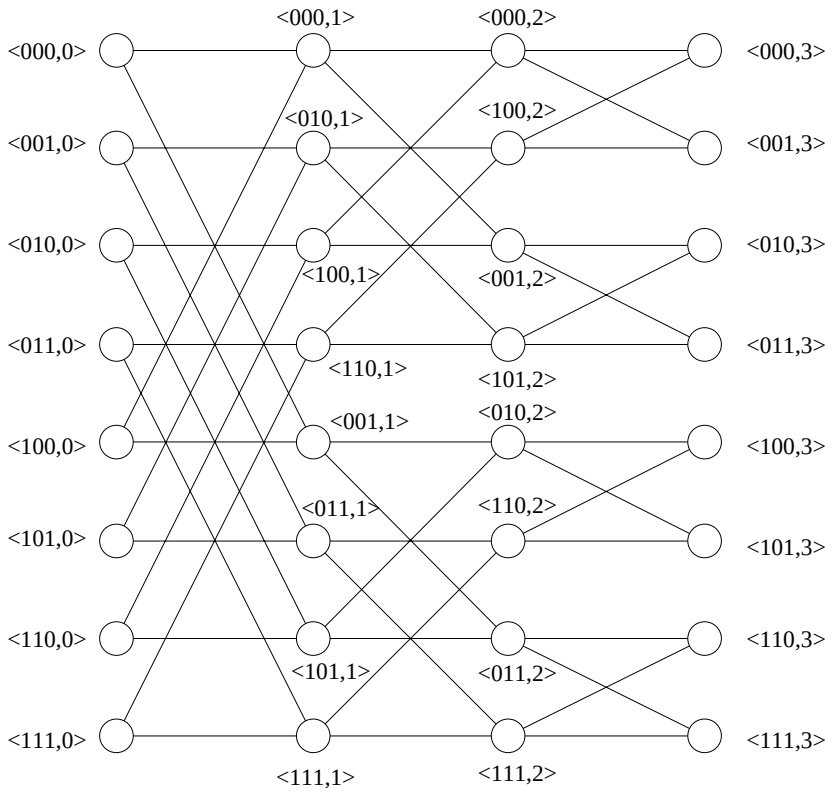

Figure 2.1 An 8-input omega network redrawn to look like a butterfly network. Node $\langle i, l \rangle$ (node i at level l) is identical to node $\langle j, l \rangle$ in the butterfly network, where j is obtained by right circular shifting the binary representation of i l times.

12 Consider a cycle $A_1, A_2, \ldots, A_k$ in a hypercube. As we travel from node A_i to A_{i+1}, the number of ones in the processor label (that is, the parity) must change. Since $A_1 = A_k$, the number of parity changes must be even. Therefore, there can be no cycles of odd length in a hypercube.
(Proof adapted from Saad and Shultz [SS88]).

13 Consider a 2^d processor hypercube. By fixing k of the d bits in the processor label, we can change the remaining $d - k$ bits. There are 2^{d-k} distinct processors that have identical values at the remaining k bit positions. A p-processor hypercube has the property that every processor has $\log p$ communication links, one each to a processor whose label differs in one bit position. To prove that the 2^{d-k} processors are connected in a hypercube topology, we need to prove that each processor in a group has $d - k$ communication links going to other processors in the same group.
Since the selected d bits are fixed for each processor in the group, no communication link corresponding to these bit positions exists between processors within a group. Furthermore, since all possible combinations of the $d - k$ bits are allowed for any processor, all $d - k$ processors that differ along any of these bit positions are also in the same group. Since the processor will be connected to each of these processors, each processor within a group is connected to $d - k$ other processors. Therefore, the processors in the group are connected in a hypercube topology.

14 Refer to Saad and Shultz [SS88].

15 **NOTE**

The number of links across the two subcubes of a d-dimensional hypercube is 2^{d-1} and not $2^d - 1$.
The proposition can be proved by starting with a partition in which both halves form subcubes. By construction, there are $p/2 (= 2^{d-1})$ communication links across the partition. Now, by moving a single processor

from one partition to the other, we eliminate one communication link across the boundary. However, this processor is connected to $d-1$ processors in the original subcube. Therefore, an additional $d-1$ links are added. In the next step, one of these $d-1$ processors is moved to the second partition. However, this processor is connected to $d-2$ processors other than the original processors. In this way, moving processors across the boundary, we can see that the minima resulting from any perturbation is one in which the two partitions are subcubes. Therefore, the minimum number of communication links across any two halves of a d-dimensional hypercube is 2^{d-1}.

16 Partitioning the mesh into two equal parts of $p/2$ processors each would leave at least $\sqrt{p}$ communication links between the partitions. Therefore, the bisection width is $\sqrt{p}$. By configuring the mesh appropriately, the distance between any two processors can be made to be independent of the number of processors. Therefore, the diameter of the network is $O(1)$. (This can however be debated because reconfiguring the network in a particular manner might leave other processors that may be more than one communication link away from each other. However, for several communication operations, the network can be configured so that the communication time is independent of the number of processors.) Each processor has a reconfigurable set of switches associated with it. From Figure 2.35 (page 80), we see that each processor has six switches. Therefore, the total number of switching elements is $6p$. The number of communication links is identical to that of a regular two-dimensional mesh, and is given by $2(p-\sqrt{p})$.

The basic advantage of the reconfigurable mesh results from the fact that any pair of processors can communicate with each other in constant time (independent of the number of processors). Because of this, many communication operations can be performed much faster on a reconfigurable mesh (as compared to its regular counterpart). However, the number of switches in a reconfigurable mesh is larger.

17 Partitioning the mesh into two equal parts of $p/2$ processors each would leave at least $\sqrt{p}$ communication links between the partitions. Therefore, the bisection width of a mesh of trees is $\sqrt{p}$. The processors at the two extremities of the mesh of trees require the largest number of communication links to communicate. This is given by $2\log(\sqrt{p}) + 2\log(\sqrt{p})$, or $2\log p$. A complete binary tree is imposed on each row and each column of the mesh of trees. There are $2\sqrt{p}$ such rows and columns. Each such tree has $\sqrt{p}-1$ switches. Therefore, the total number of switches is given by $2\sqrt{p}(\sqrt{p}-1)$, or $2(p-\sqrt{p})$.

Leighton [Lei92a] discusses this architecture and its properties in detail.

18 In the d-dimensional mesh of trees, each dimension has $p^{1/d}$ processors. The processor labels can be expressed in the form of a d-tuple. The minimum number of communication links across a partition are obtained when the coordinate along one of the dimensions is fixed. This would result in $p^{(d-1)/d}$ communication links. Therefore, the bisection width is $p^{(d-1)/d}$.

Connecting $p^{1/d}$ processors into a complete binary tree requires $p^{1/d}-1$ switching elements. There are $p^{(d-1)/d}$ distinct ways of fixing any one dimension and there are d dimensions. Therefore, the total number of switching elements is given by $dp^{(d-1)/d}(p^{1/d}-1)$, or $d(p-p^{(d-1)/d})$.

Similarly, the number of communication links required to connect processors along any one dimension is given by $2(p^{1/d}-1)$. Using a procedure similar to the one above, we can show that the total number of communication links is given by $dp^{(d-1)/d}2(p^{1/d}-1)$, or $2d(p-p^{(d-1)/d})$.

The diameter of the network can be derived by traversing along each dimension. There are d dimensions and traversing each dimension requires $2\log(p^{1/d})$ links. Therefore, the diameter is $d2\log(p^{1/d})$, or $2\log p$.

The advantages of a mesh of trees is that it has a smaller diameter compared to a mesh. However, this comes at the cost of increased hardware in the form of switches. Furthermore, it is difficult to derive a clean planar structure, as is the case with 2-dimensional meshes.

19 Leighton [Lei92a] discusses this solution in detail.

20 Figure 2.2 illustrates a 4×4 wraparound mesh with equal wire lengths.

21 Consider a $p \times q \times r$ mesh being embedded into a 2^d processor hypercube. Assume that $p = 2^x$, $q = 2^y$, and $r = 2^z$. Furthermore, since $p \times q \times r = 2^d$, $x + y + z = d$.

The embedding of the mesh can be performed as follows: Map processor (i, j, k) in the mesh to processor

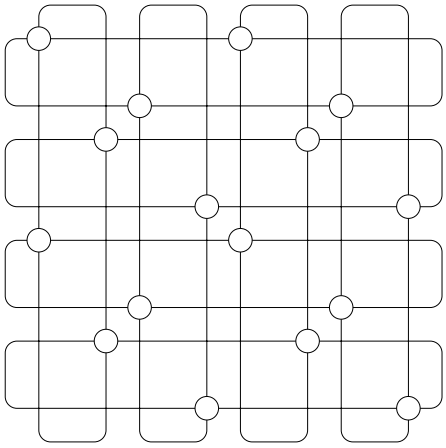

Figure 2.2 A 4×4 wraparound mesh with equal wire lengths.

$G(i, x)G(j, y)G(k, z)$ (concatenation of the Gray codes) in the hypercube using the Gray code function G described in Section 2.7.1 (page 67).

To understand how this mapping works, consider the partitioning of the d bits in the processor labels into three groups consisting of x, y, and z bits. Fixing bits corresponding to any two groups yields a subcube corresponding to the other group. For instance, fixing $y+z$ bits yields a subcube of 2^x processors. A processor (i, j, k) in the mesh has direct communication links to processors $(i + 1, j, k)$, $(i - 1, j, k)$, $(i, j + 1, k)$, $(i, j-1, k)$, $(i, j, k+1)$, and $(i, j, k-1)$. Let us verify that processors $(i+1, j, k)$ and $(i-1, j, k)$ are indeed neighbors of processor (i, j, k). Since j and k are identical, $G(j, y)$ and $G(k, z)$ are fixed. This means that the two processors lie in a subcube of 2^x processors corresponding to the first x bits. Using the embedding of a linear array into a hypercube, we can verify that processors $(i+1, j, k)$ and $(i-1, j, k)$ are directly connected in the hypercube. It can be verified similarly that the other processors in the mesh which are directly connected to processor (i, j, k) also have a direct communication link in the hypercube.

23 Ranka and Sahni [RS90] present a discussion of the embedding of a complete binary tree into a hypercube.

24 The mapping of a mesh into a hypercube follows directly from an inverse mapping of the mesh into a hypercube. Consider the congestion of the inverse mapping. A single subcube of $\sqrt{p}$ processors is mapped onto each row of the mesh (assuming a $\sqrt{p} \times \sqrt{p}$ mesh). To compute the congestion of this mapping, consider the number of links on the mesh link connecting one half of this row to the other. The hypercube has $\sqrt{p}/2$ links going across and a single row of the mesh (with wraparound) has two links going across. Therefore, the congestion of this mapping is $\sqrt{p}/4$.

It can be shown that this mapping yields the best congestion for the mapping of a hypercube into a mesh. Therefore, if the mesh links are faster by a factor of $\sqrt{p}/4$ or more, the mesh computer is superior. For the example cited, $p = 1024$. Hence, for the mesh to be better, its links must be faster by a factor of $\sqrt{1024}/4 = 8$. Since the mesh links operate at 25 million bytes per second and those of the hypercube operate at 2 million bytes per second, the mesh architecture is indeed strictly superior to the hypercube.

25 The diameter of a k-ary d-cube can be derived by traversing the farthest distance along each dimension. The farthest distance along each dimension is $k/2$ and since there are d such dimensions, the diameter is $dk/2$.

Each processor in a k-ary d-cube has $2d$ communication links. Therefore, the total number of communication links is pd.

The bisection width of a k-ary d-cube can be derived by fixing one of the dimensions and counting the number of links crossing this hyperplane. Any such hyperplane is intersected by $2k^{(d-1)}$ (for $k > 2$) communication links. The factor of 2 results because of the wraparound connections. (Note that the bisection width can also be written as $2k^{d-1}$).

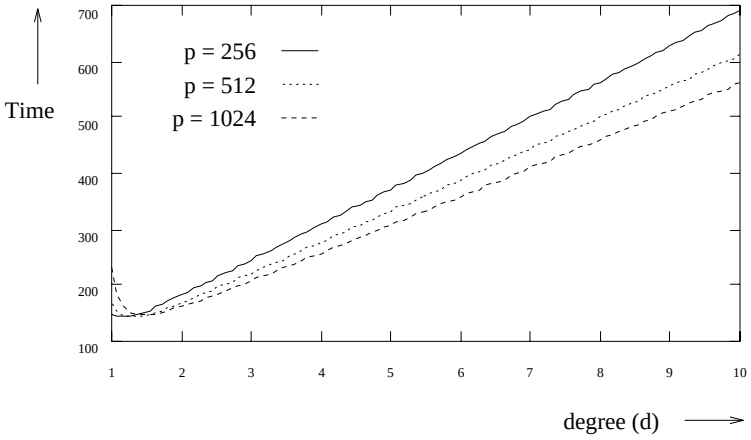

Figure 2.3 Communication time plotted against the degree of a cut-through network routing using number of communication links as a cost metric.

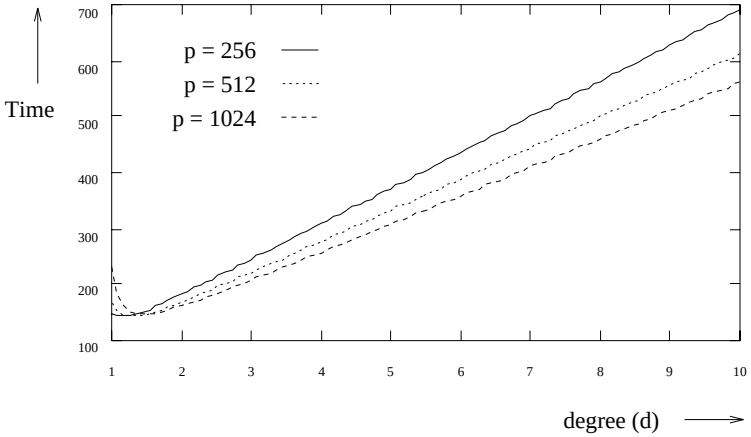

Figure 2.4 Communication time plotted against the degree of a cut-through routing network using bisection width as a cost metric.

The average communication distance along each dimension is $k/4$. Therefore, in d dimensions, the average distance l_{av} is $kd/4$.

27 (1) The cost of a k-ary d-cube of p processors in terms of number of communication links is given by dp (for $k > 2$). The corresponding cost for a binary hypercube is given by $p \log p/2$. Therefore, if the width of each channel in the k-ary d-cube is r, then the total cost is given by dpr. If this cost is identical to the cost of a binary hypercube, then $dpr = p \log p/2$, or $r = \log p/(2d)$.
(2) The bisection width of a k-ary d-cube of p processors is given by $2k^{d-1}$ and that of a binary hypercube is given by $p/2$. If the channel width of the k-ary d-cube is r, then equating the costs yields $r \times 2k^{d-1} = p/2$. Therefore, the channel width is $r = p/(4 \times k^{d-1})$. Since $k^d = p$, $r = k/4$.
(The cost and bisection width of these networks is given in Table 2.1 (page 44))

28 The average distance between two processors in a hypercube is given by $\log p/2$. The cost of communicating a message of size m between two processors in this network with cut-through routing is

$$T_{comm} = t_s + t_h \frac{\log p}{2} + t_w m.$$

The average distance between two processors in a k-ary d-cube is given by $kd/4$. The cost of communicating

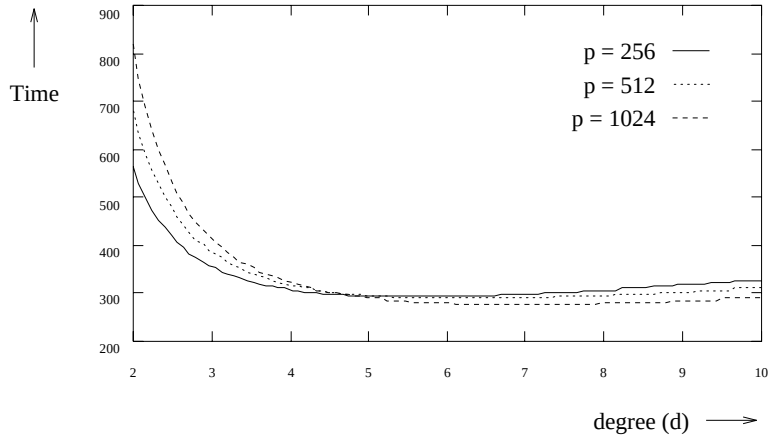

Figure 2.5 Communication time plotted against the degree of a store-and-forward network routing using number of communication links as a cost metric.

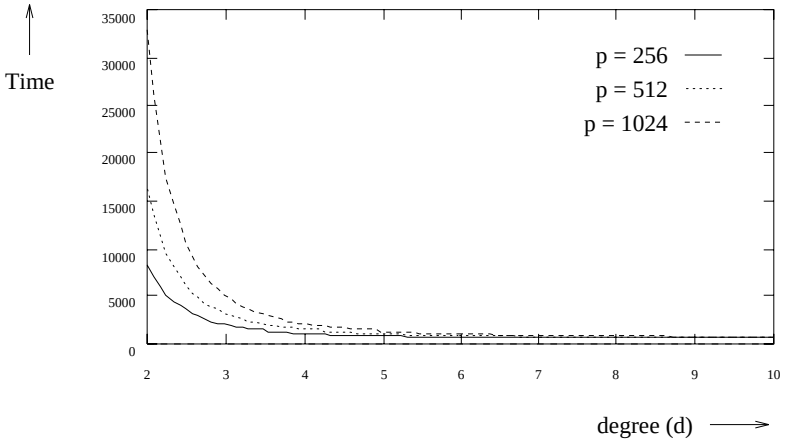

Figure 2.6 Communication time plotted against the degree of a store-and-forward network routing using bisection width as a cost metric.

a message of size m between two processors in this network with cut-through routing is

$$T_{comm} = t_s + t_h \frac{kd}{4} + \frac{t_w}{r} m,$$

where r is the scaling factor for the channel bandwidth.

From Solution 2.20, if number of channels is used as a cost metric, then we have $r = s = \log p/(2d)$. Therefore,

$$T_{comm} = t_s + t_h \frac{kd}{4} + \frac{2t_w d}{\log p} m.$$

Similarly, using the bisection width as a cost metric, we have $r = s = k/4$. Therefore,

$$T_{comm} = t_s + t_h \frac{kd}{2} + \frac{kt_w}{4} m.$$

The communication times are plotted against the dimension of the k-ary d-cube for both of these cost metrics in Figures 2.3 and 2.4.

29 The cost of communicating a message of size m between two processors in a hypercube with store-and-forward

routing is

$$T_{comm} = t_s + t_w m \frac{\log p}{2}.$$

Using the number of links as a cost metric, for a k-ary d-cube the corresponding communication time is given by

$$T_{comm} = t_s + t_w \frac{2d}{\log p} \frac{kd}{2} m.$$

This communication time is plotted against the degree of the network in Figure2.5.
Using the bisection width as a cost metric, for a k-ary d-cube the corresponding communication time is given by

$$T_{comm} = t_s + \frac{kt_w}{4} \frac{kd}{2} m.$$

This communication time is plotted against the degree of the network in Figure2.6.