CIS 668-102 Parallel Algorithms	**Homework 1** **Due: Week 7, Mon. March. 1**	Dr. David Nassimi Spring 2004

Note: All algorithms in this course must be presented in a complete manner:

1. First, provide an informal sketch/outline of the algorithm.
2. Give the detailed formal algorithm, with comments where appropriate.
3. Provide a short correctness proof, unless very obvious.
4. Analyze the time complexity in terms of the number of routing steps and computational steps. (Include the derivation, unless obvious or repetitious.)
5. Give a pictorial/numerical illustration where it is asked for.

---

1. Consider a *linearly-connected* parallel computer with N processors (PEs) without endaround connections. Assume N is a power of 2. Each PE$[i]$ has several registers $A[i], B[i]$, etc. The architecture has a long-distance routing instruction. A statement of the type

   $$B[i] \leftarrow A[i+k]$$

   or

   $$B[i] \leftarrow A[i-k]$$

   is considered a *(long-distance) "routing instruction"* executed as k *"unit-distance moves"*. (This same terminology is used when $k = 1$.) Assume that all data movements during a routing instruction must be **in the same direction and by the same distance**.

   (a) Consider the problem of performing a **vector reversal** permutation on $A[0 : N-1]$, which is the permutation

   $$A[N-1-i] \leftarrow A[i],\ 0 \le i \le N-1.$$

   Prove a lower bound of $2(N-1)$ for the number of unit-distance moves. That is, prove that any algorithm for this problem on this architecture (without endaround connections) must use at least $2(N-1)$ unit-distance moves. *Hint*: Consider the maximum distance any item needs to move.

   (b) Write a non-recursive parallel algorithm to perform the vector reversal permutation. Find the total number of routing instructions and unit-distance moves.
   Hint: Using divide-and-conquer technique, the algorithm must use $O(\log N)$ routing instructions while achieving the above lower bound on the number of unit-distance moves.
   Illustrate your algorithm for $N = 8$ and $A[0:7] = (0, 1, 2, 3, 4, 5, 6, 7)$.

2. Consider a *linearly-connected* parallel computer with N PEs. We want the sum of $A[0 : N-1]$ to endup in $S[i]$ **for all** i. (That is, every PE is to receive the sum.)

   (a) Assuming no endaround connections, prove a lower bound of $2(N-1)$ for the number of unit-distance moves. Design an algorithm achieving this lower bound, while using $O(\log N)$ routing instructions and additions. Analyze the algorithm. (Find the total number of addition steps, routing instructions, and unit-distance moves.) Illustrate the algorithm for $N = 8$ and $A[0 : 7] = (0, 1, 2, 3, 4, 5, 6, 7)$.

(b) Now assume endaround connections. That is, PE$[N-1]$ is connected to PE[0]. (PE[0] is the "right" neighbor of PE$[N-1]$.) Prove a lower bound of $N-1$ for the number of unit-distance moves. Write an algorithm which achieves this lower bound, while using $O(\log N)$ routing instructions and additions. Analyze the algorithm. Illustrate the algorithm for $N = 8$ and $A[0:7] = (0, 1, 2, 3, 4, 5, 6, 7)$.

3. Given a vector $A[0 : N-1]$, *all-prefix-sums* (also called *all-partial-sums*) of A is the vector $S[0 : N-1]$ where $S[i] = A[0]+A[1]+\cdots+A[i],\ 0 \le i \le N-1$. We want to compute all-prefix-sums on a *linearly-connected* parallel processor with N PEs.

   (a) Assuming no endaround connections, prove a lower bound of $N-1$ for the number of unit-distance moves. Write an algorithm which achieves this lower bound, while using $O(\log N)$ routing instructions and additions. Illustrate the algorithm for $N = 8$ and $A[0:7] = (0, 1, 2, 3, 4, 5, 6, 7)$.

   (b) Now assume endaround connections. Prove that the algorithm of part a can not be improved by showing that the lower bound of $N-1$ for the number of unit-distance moves is still valid. (The argument is similar to 2b.)

4. (a) Write a parallel algorithm to compute *all-prefix-sums* of $A[0 : N-1]$ on an N-PE square *mesh* without endaround connections. (The mesh is $M \times M$, $N = M^2$, and $M = 2^P$.) Find the total number of long-distance-routing steps, unit-distance-moves, and addition steps. Give an illustration for $M = 4$.

   Hint: It can be done with $O(\log M)$ routing instructions and approximately $3M$ unit-distance moves.

   (b) Now assume *endaround connections* of the following type:

   PE$[i, M-1]$ is connected to PE$[i+1, 0]$, for $i = 0, 1, \ldots, M-2$.
   PE$[M-1, j]$ is connected to PE$[0, j]$, for $j = 0, 1, \ldots, M-1$.

   (The last PE in each row is connected to the first PE in the next row. And, the last PE in each column is connected back to the first PE in the same column.) Note that this scheme of row-endaround connections make the entire N PEs linearly-connected.

   Design an efficient algorithm for this case. Analyze the algorithm. (Be sure to show the derivation.) Illustrate the algorithm for $M = 4$ and A as in part a.

CIS 668-102 Parallel Algorithms	**Homework 2** **Due: Week 8, Mon. March 8**	Dr. David Nassimi Spring 2004

1. (a) Write an algorithm to compute *all-prefix-sums* (also called *all-partial-sums*) of $A[0:N-1]$ on an N-PE *cube*, $N = 2^P$. Denote the result as $S[0:N-1]$. (Recall $S[i] = A[0]+A[1]+\cdots+A[i]$.) Analyze the total number of routing steps and addition steps. Illustrate the cube algorithm for N=8 and A=(5,10,8,7,3,12,11,4).

   Hint: Use divide-and-conquer.

   (b) Write the algorithm for a *Perfect-Shuffle-Computer* (PSC) with N PEs. Analyze the number of steps. Illustrate the PSC algorithm for N=8 and A=(5,10,8,7,3,12,11,4).

2. Write an algorithm to compute *all-prefix-sums* on a *tree-connected* parallel processor with N PEs, $N = 2^P - 1$. The PEs are numbered $1, 2, ..., N$ in breadth-first order. That is, the root is number 1 and level 0. The nodes at the next level are number 2 and 3 left-to-right, and so on. The leaf nodes are number $(N+1)/2, ..., N$. For simplicity, assume that *only the leaf nodes contain one data item each*, and the internal nodes contain none. That is, the initial data vector is $A[(N+1)/2 : N]$ and the result will be $S[(N+1)/2 : N]$ where

$$S[i] = A[(N+1)/2] + \cdots + A[i], \quad (N+1)/2 \le i \le N.$$

   Analyze the total number of routing steps and addition steps. Illustrate the algorithm for $N = 15$ and $A[8:15] = (0, 1, 2, 3, 4, 5, 6, 7)$.

   Hint: The algorithm will consists of 2 phases: During phase 1, the information flows from the leaves to the root. During the second phase, the flow is from the root down to the leaves.

3. Some parallel computers built in the past used *bit-serial* processors (PEs) and bit-serial communication links to reduce the amount of hardware in the PEs and the interconnection network, while supposedly increasing the flexibility. (An example is the Massively-Parallel-Processor (MPP) which had 16,384 PEs arranged in a 128×128 mesh. Another example of such a system is the Connection Machine.)

   This problem explores simple **pipelining** on a bit-serial parallel computers with N PEs, $N = 2^P$, and with *linear-interconnection*. (PEs are numbered $0 : N-1$.) The links between adjacent PEs are only 1-bit wide. During a unit-distance-routing step, only one bit may be transmitted from a PE to its neighbor. $R[i]$ is a one-bit register in PE$[i]$ used for routing. There is still a *long-distance routing* instruction such as

$$R[i] \leftarrow R[i+d], \quad (mask)$$

   which is executed as a sequence of d *unit-distance-moves*.

   The arithmetic inside each PE is also bit-serial. For example, if $X[i], Y[i], Z[i], C[i], S[i]$ are 1-bit registers in PE$[i]$, an addition step may be of the form

$$(C[i], S[i]) := \text{ADD } (X[i], Y[i], Z[i])$$

   which adds the 3 bits $X[i], Y[i], Z[i]$ using a FULL-ADDER unit, and puts the resulting sum-bit into $S[i]$ and carry-bit into $C[i]$.

   Assume that each PE$[i]$ has several 1-bit registers ($A[i], B[i]$, etc.), and in addition *one* K-bit wide register $W[i]$, where K is a constant independent of N. An arithmetic step may use either the least-significant-bit (LSB) or the most-significant-bit (MSB) of W. Let $W[i]_b$ denote bit b of $W[i]$, where bit 0 is the LSB. An example of an arithmetic step is

$$(C[i], W[i]_0) := \text{ADD } (C[i], W[i]_0, R[i]).$$

Assume a right-rotate instruction of the type

$$\text{RIGHT-ROTATE } (W[i], R[i]), \quad (mask)$$

which right-rotates the $K+1$ bits. That is, $W[i]$ is rightshifted while $R[i]$ goes into the MSB of $W[i]$, and the LSB of $W[i]$ goes into $R[i]$. Similarly, the instruction

$$\text{LEFT-ROTATE } (W[i], R[i]), \quad (mask)$$

left-rotates the $K+1$ bits, with $R[i]$ going into LSB of $W[i]$ and the MSB of W[i] going into $R[i]$.

(a) Using the above primitive instructions, write a parallel algorithm to accomplish the following:

$$W[i] \leftarrow W[i] + W[i+D], \text{ (if } i \text{ is a multiple of } 2D).$$

For example if $D = 2$, PE[0] receives data from PE[2], 4 from 6, etc. Note that the above mask insures that the (destination, source) pairs are non-overlapping. This enables efficient use of **pipelining**. (Don't be concerned about preserving the original data in $W[i+D]$.) Be sure your algorithm handles properly both cases of $D < K$ and $D \geq K$. Derive the total number of unit-distance moves for each case.

(b) Now, consider the summation problem

$$W[0] := W[0] + W[1] + \cdots + W[N-1].$$

(Each PE initially has one data item. The sum is to endup in PE 0.) The following algorithm is a high-level description for doing this task.

1. for $b := 0$ to $\log N - 1$ do
2. $\quad W[i] \leftarrow W[i] + W[i + 2^b], \ (i_{b:0} = 0)$

(Note that the above mask means $i_b i_{b-1} \ldots i_0 = 0$, which means i is a multiple of 2^{b+1}.) Assuming line 2 is implemented using the program of part a, derive the total number of unit-distance moves used by this algorithm.

4. Consider a *bit-serial* parallel computer with *tree* interconnection. (There are $N = 2^P - 1$ PEs numbered $1 : N$ in breadth-first order.) Each *leaf* PE[i] has a k-bit integer $W[i]$. (k is a constant independent of N). Write an efficient **pipelined** algorithm to compute the sum and put the result in $W[1]$, the root. (Don't be concerned about preserving the original data in the leaf nodes.) Find the total number of unit-distance routing steps and addition steps. (In one routing step, a parent can receive data from *both* of its children.) Illustrate the algorithm for $N = 7$, $k = 4$, and $W[4:7] = (1, 5, 2, 6)$. (Show the original numbers in binary. Show the result in each PE after each iteration of your algorithm. Draw the structure as *tree*.)

5. (a) Consider an $M \times M$ *mesh* holding an $M \times M$ matrix $A[i, j], 0 \leq i, j \leq M-1$, where $A[i, j]$ is in PE[i, j]. Let $N = M^2$ and $M = 2^P$. Assume no endaround connections. Write a non-recursive parallel algorithm to *transpose* the matrix. Use index-bit-masking scheme. Analyze the number of steps.

(b) Write the matrix transpose algorithm for a *cube* with N PEs, $N = M^2 = 2^{2P}$. The matrix is stored in row-major-order in $A[0], A[1], A[2], \ldots, A[N-1]$. Note that the PE-index bits are numered $2P - 1 : 0$. Analyze the number of steps.

(c) Now consider a PSC with N PEs. Write a matrix transpose algorithm which is obtained as a "simulation" of the cube algorithm. Find the total number of routing steps.

(d) A much more efficient algorithm is possible for transpose on PSC which is obtained not from the cube algorithm, but from a direct clever design for PSC. Write the algorithm and analyze the number of steps.

Additional Exercises (Not to be handed-in)

6. This problem is to do *vector reversal* of $A[0 : N-1]$ on a linearly-connected parallel processor with N PEs, as in problem 1 of homework 1, but *with endaround connections*. Assume $N \geq 4$.

   (a) Prove a lower bound of $N-2$ for the number of unit-distance moves. (Recall that all data movements must be in the same direction at a time.)

   (b) Suppose we allow $O(N)$ routing instructions and arbitrary masking scheme. A simple algorithm which uses $N-1$ unit-distance moves is to circulate the data in $N-1$ unit-distance moves (always in the same direction, say right). After each move, the items that reach their final destinations are dropped off. (Note that this algorithm is general enough to handle *any arbitrary permutation* in $N-1$ unit-distance moves.) Write the algorithm for vector-reversal using this approach.

   (c) Although the algorithm of part b is nearly optimal in unit-distance moves, the algorithm is inefficient because it uses $O(N)$ routing instructions. Additionally, there is a large overhead associated with computing the mask after each routing step to determine which items to drop off.

   A much more efficient algorithm is possible which is optimal in the number of unit-distance moves, and uses $O(\log N)$ routing instructions and the efficient index-bit-masking scheme. (The architecture may implement this masking scheme using only $2 \log N$ bits, rather than N bits as needed by a general masking scheme.) Obtain the algorithm. Provide an informal correctness proof. Analyze the number of steps. Illustrate the algorithm for $N = 8$ and $A = (0, 1, 2, 3, 4, 5, 6, 7)$.

   (d) Another algorithm, yet more efficient than c, is possible which uses only right moves. This algorithm will use $N-1$ unit-distance moves (one more than c) but with about half as many routing instructions as c. As in c, this algorithm uses the efficient index-bit-masking scheme. Obtain the algorithm. Provide an informal correctness proof. Analyze the number of steps. Illustrate the algorithm for $N = 8$ and $A = (0, 1, 2, 3, 4, 5, 6, 7)$.

CIS 668-102 Parallel Algorithms	**Homework 3** **Due: Week 12, Mon. April 19**	Dr. David Nassimi Spring 2004

1. This problem is to multiply two $M \times M$ matrices on an $M \times M$ *mesh* with *orthogonal endaround* connections. (The last PE in each row is connected to the first PE in the same row, and the last PE in each column is connected to the first PE in the same column.) Let the product matrix be $C = AB$. Let $a_{i,j}, b_{i,j}, c_{i,j}$ be the elements of the matrices, and let $A[i,j], B[i,j], C[i,j]$ be registers in PE$[i,j]$. Initially, $A[i,j] = a_{i,j}$ and $B[i,j] = b_{i,j}$. At the end, $C[i,j] = c_{i,j}$.

   (a) Assuming that each PE has a local memory of $2M$ words (in addition to a few registers), a simple algorithm is to first bring into PE$[i,j]$ row i of A and column j of B, and then have PE$[i,j]$ compute $c_{i,j}$. Let MEM$[i,j,k]$ be word k within the memory of PE$[i,j]$. Assume each PE has an index register, $X[i,j]$, which can be used to reference the PE's memory. You may write such references as MEM$[i,j,X[i,j]]$. For example, you may have the following sequence of instructions:
   $X[i,j] := i,\ (\forall i,j)$ {Put row number into PE's index register.}
   $T[i,j] := MEM[i,j,X[i,j]],\ (\forall i,j)$ {Load register T from memory.}
   Write the algorithm. Analyze the time complexity.

   (b) A more efficient $O(M)$ algorithm is possible which assumes only a few registers in each PE (no local memory). (The algorithm was outlined in class.) Write the algorithm. Illustrate the algorithm for $M = 4$ and

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

   (c) Now suppose the mesh is without the endaround connections. Briefly explain how the algorithm of part b may be "simulated" to still achieve O(M) time complexity.

2. Write a parallel algorithm to multiply two $M \times M$ matrices on a *cube* with N PEs, for each of the following cases of N. Each PE has only a few registers A, B, C, etc. (No local memory.) Let $M = 2^P$. Let the product matrix be $C = AB$. Let $a_{i,j}, b_{i,j}, c_{i,j},\ 0 \le i,j \le M-1$, denote the elements of the matrices. Provide comments inside {} to improve readability. Analyze the time complexity in each case.

   (a) $N = M^2$.
   For notational convenience, denote the PEs as PE$[i,j]$, where $0 \le i,j \le M-1$. Note that $iM + j$ gives the PE's single-index in the range $[0 : N-1]$. Let $A[i,j], B[i,j], C[i,j]$ be registers in PE$[i,j]$. Initially, $A[i,j] = a_{i,j}$ and $B[i,j] = b_{i,j}$. At the end, $C[i,j] = c_{i,j}$. Illustrate the algorithm for $M = 4$, $a_{i,j} = i + j$ and $b_{i,j} = i$, $0 \le i,j \le 3$.
   Hint: Let $G[0 : M-1]$ be a gray-code sequence. (The sequence is a permutation of $[0, 1, \ldots, M-1]$, the first element is $G[0] = 0$, and every consecutive pair in the sequence differ in only one bit.) Assume the control unit has the precomputed sequence $T[1 : M-1]$ such that

$$G[i] = G[i-1] \oplus 2^{T[i]},\ 1 \le i \le M-1.$$

   That is, $T[i]$ is the bit position in which the consecutive pair $(G[i-1], G[i])$ differ. For example for $M = 8$, a possible gray-code sequence is $(0, 1, 3, 2, 6, 7, 5, 4)$ resulting in $T[1:7] = (0, 1, 0, 2, 0, 1, 0)$. The algorithm is outlined as follows:

1. Initially, re-align the matrices so that $A[i,j] = a_{i,i\oplus j}$ and $B[i,j] = b_{i\oplus j,j}$.
2. $C[i,j] := A[i,j] * B[i,j],\ (\forall i,j)$
3. For $k := 1$ to $M-1$ do
   3.1 Route A, B to obtain $A[i,j] = a_{i,i\oplus j\oplus G[k]},\ B[i,j] = b_{i\oplus j\oplus G[k],j}$.
   3.2 $C[i,j] := C[i,j] + A[i,j] * B[i,j],\ (\forall i,j)$

(b) $N = M^3$.
Denote the PEs as PE$[r,i,j]$, where $0 \le r,i,j \le M-1$. This corresponds to a single index $s = rM^2 + iM + j$ in the range $[0 : N-1]$. Initially, $A[0,i,j] = a_{i,j}$ and $B[0,i,j] = b_{i,j}$. At the end, $C[0,i,j] = c_{i,j}$.

(c) $N = HM^2$, where $1 \le H \le M$ and $H = 2^h$.
(Note that this problem reduces to part a if $H = 1$, and to part b if $H = M$.) Denote the PEs as PE$[r,i,j]$, where $0 \le r \le H-1$ and $0 \le i,j \le M-1$. (The PEs have a single-index $s = rM^2 + iM + j$ in the range $0 : N-1$.) Initially, $A[0,i,j] = a_{i,j}$ and $B[0,i,j] = b_{i,j}$. At the end, $C[0,i,j] = c_{i,j}$. Give an illustration for $M = 4$ and $H = 2$.
Hint: Partition each $M \times M$ matrix into H^2 submatrices, each of size $M/H \times M/H$. And, partition the set of PEs into H^3 regions, each of size $M/H \times M/H$.