IN-61
158534
P.25

NASA Contractor Report 191430
ICASE Report No. 93-5

ICASE

20 Years of Excellence

ISOMORPHIC ROUTING ON A TOROIDAL MESH

(NASA-CR-191430) ISOMORPHIC ROUTING ON A TOROIDAL MESH Final Report (ICASE) 25 p

N93-23058

Unclas

G3/61 0158534

Weizhen Mao
David M. Nicol

NASA Contract Nos. NAS1-19480 and NAS1-18605
February 1993

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23681-0001

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

ISOMORPHIC ROUTING ON A TOROIDAL MESH

Weizhen Mao and David M. Nicol[1]

Department of Computer Science
The College of William and Mary
Williamsburg, VA 23185

ABSTRACT

We study a routing problem that arises on SIMD parallel architectures whose communication network forms a toroidal mesh. We assume there exists a set of k message descriptors $\{(x_i, y_i)\}$, where (x_i, y_i) indicates that the i^{th} message's recipient is offset from its sender by x_i hops in one mesh dimension, and y_i hops in the other. Every processor has k messages to send, and all processors use the same set of message routing descriptors. The SIMD constraint implies that at any routing step, every processor is actively routing messages with the same descriptors as any other processor. We call this *Isomorphic Routing.* Our objective is to find the isomorphic routing schedule with least makespan. We consider a number of variations on the problem, yielding complexity results from $O(k)$ to NP-complete. Most of our results follow after we transform the problem into a scheduling problem, where it is related to other well-known scheduling problems.

[1]Research was supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-18605 and NAS1-19480 while the second author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

1 Introduction

The issue of routing messages in a parallel computer network has attracted a considerable amount of attention. A host of problem variations exist. For example, some models presume that every processor i holds a number and that one wishes to implement some permutation (e.g., [6]). Another variation is to assume that each processor i has a list of messages each of which is destined for an arbitrary processor, this is known as "all-to-all personalized communication" [4]. Our problem is a constrained case of all-to-all personalized communication, on an $n \times m$ toroidal mesh. It is also a constrained case of the general "compiled communication" problem studied in [1], where the problem is to construct a communication schedule for an irregular computation.

To begin with, in our problem, we can always describe a message's destination in terms of the offset in both mesh dimensions X and Y of the source processor. Thus, a pair (x, y) describes a message's routing requirements. Observe however that a message needn't travel exactly x units in the X dimension and y in the Y—because of wrap-around, it may equally well choose to travel $m - x$ units in X and/or $n - y$ units in Y. Now imagine a parallel computation where every processor performs the same computation, but on different data. Further suppose that the pattern of messages every processor sends is the same, e.g., patterns associated with discretization stencils [7]. We may thus describe the communication requirements of the entire computation in terms of the offsets $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ of the k messages a single processor sends. We will say that the $n \times m$ different messages with a common offset pair are all *isomorphic.*

Every processor has four communication ports, referenced as North, East, West, and South (N, E, W, and S). We assume the communication links are full-duplex. We are interested in SIMD (Single Instruction Multiple Data) architectures, where processors execute the same instruction stream in lock-step. Unless the architecture provides special support for local indirect addressing (which is much slower even when provided), an implication of SIMD processing is that at every instant, the set of messages moving through all ports of a common type (e.g., N) are isomorphic. We desire a routing schedule that minimizes the time required to complete the communication, i.e., the makespan.

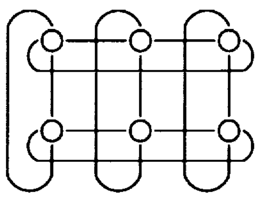
We will examine variations of the problem, finding they have a surprising range of complexities. The variations derive from assumptions concerning how many communication ports may be active at a time, and whether a message must be fully routed once it begins moving or if it can be temporarily buffered at an intermediate processor. The assumptions and associated complexities are given below.

- **One port active at a time:** $O(k)$;
- **All ports active, temporary buffering allowed:** $O(k \log k)$;

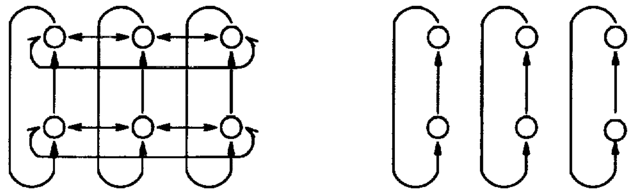
- **All ports active, no temporary buffering:** NP-complete.

Let us now state the problem more formally. In a toroidal mesh of $n \times m$ processors, a set of messages $M = \{m_1, m_2, \ldots, m_k\}$ is to be sent from each of the $n \times m$ processors (the sources) to some other processors (the destinations). Each message m_i is represented by a pair of integers (x_i, y_i) giving the relative offset of its destination from its source. Assume $0 \leq x_i \leq m-1$ and $0 \leq y_i \leq n-1$. We wish to design a schedule so that all messages at each processor are sent to (and received by) their destinations in the minimum amount of time. Depending on the problem variation, at any time a processor can send one message in one of four directions (N, E, W, or S), or at any time a processor can send up to four messages, one in each direction. We assume it always takes one time unit for a message to traverse one link. We notice that for any message $m_i = (x_i, y_i)$ there are four possible ways to send it, East and North (x_i, y_i), East and South $(x_i, -(n-y_i))$, West and North $(-(m-x_i), y_i)$, and finally, West and South $(-(m-x_i), -(n-y_i))$. Because the mesh is toroidal, they all reach the same destination. Depending on the problem variation, we either assume that a message must be routed to completion in a successive series of steps, or that a message's movement can be fragmented, e.g., one step N, two steps buffered, one step W, another step N, and so on.

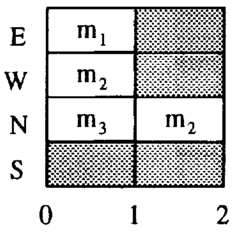
For example, in a 2×3 toroidal mesh shown in FIG. 1(a), 3 messages are to be sent, they are $m_1 = (1,0)$, $m_2 = (2,1)$, and $m_3 = (0,1)$. Assuming that all ports may be active simultaneously, we easily determine that the makespan of the optimal schedule, denoted by C^*_{max}, is 2. From time 0 to time 1, each processor sends m_1 East to its destination, m_2 West, and m_3 North to its destination. From time 1 to time 2, each processor sends m_2 North to its destination. The schedule is illustrated in FIG. 1(b). Under our assumption of isomorphic message passing, each processor does exactly the same thing at the same time. Any time a processor sends out a message on one port, (e.g., N), in the following time step a message isomorphic to it is received on the opposite port (e.g., S), save that one unit of routing service in one dimension (e.g., Y) has been given. This observation suggests that we can approach the scheduling problem in terms of a single processor giving routing service to each of its k messages. The schedule for one processor can be shown by the traditional Gantt chart as in FIG. 1(c).



(a)



(b)



(c)

FIG. 1. An example.

A message may travel either direction within a dimension. This allows the possibility of schedules that cause a message to "backtrack", e.g., move 3 units W, and later move 4 units E. In the case when temporary buffering is provided at each processor, such a schedule can always be improved (at least not degraded) by removing the backtracking loop, whence if C^*_{max} is the minimized makespan for an instance of the isomorphic routing problem, there exists a backtracking-free schedule with cost C^*_{max}. When there is no temporary buffering, backtracking may be needed just to keep a message moving until it reaches its destination. In the remainder we will confine our attention to backtracking-free schedules.

The problem defined above can be converted to an equivalent problem similar to the open shop scheduling. We are given four machines E, W, N, S, in which E, and W are identical in function but give different service times, as do N and S. There are k jobs, $J_1, J_2, \ldots, J_k$. Each job J_i consists of two tasks X_i and Y_i, where X_i can only be executed by E or W (but not both, because there is no backtracking), taking x_i or $m - x_i$ time units, respectively, and Y_i can only be executed by N or S, taking y_i or $n - y_i$ time units, respectively. The integers m, n, x_i's and y_i's are as defined in the original problem above.

Tasks X_i and Y_i cannot be executed simultaneously at any time, but may be broken into unit-time slices. However, in some problem variations a job J_i may be *suspended* once it has begun execution, a feature that corresponds with a message being buffered en route. All task execution periods occur on the same machine. Our goal is to find a schedule to execute all jobs so that the makespan or the maximum completion time C_{max} is minimized.

This problem definition suggests a new group of problems, which we call multi-operation/multi-machine scheduling problems. In the classical multi-operation model [5], each job requires execution on more than one machine. In an open shop the order in which a job passes through the machines is immaterial, whereas in a flow shop each job has the same machine ordering and in a job shop the jobs may have different machine orderings. In the multi-operation/multi-machine model, instead of having just one machine to perform a certain kind of task for a job, there is a back-up machine with the same function and a possibly different cost.

We can distinguish the situations in which a task requires identical service at either common function machine, or has different service requirements that depend on the machine. Our problem is a special case of the latter. In particular, we assume that for each pair of common function machines there exists an integer c ($c = m$ for N-S, $c = n$ for E-W) such that a task with demand x_i requires x_i units on one machine and $c - x_i$ units on the other. In this case we will say that the machines give *complementary* service.

In the remainder we will refer to problem variations by the following names.

P_1: Only one machine (out of all four) may be executing at a time.

P_2: All four machines may execute simultaneously, jobs may be suspended, common function machines give complementary service.

P_3: All four machines may execute simultaneously, jobs may not be suspended, common function machines give complementary service.

P_4: All four machines may execute simultaneously, jobs may be suspended, common function machines give uniform service.

P_5: All four machines may execute simultaneously, jobs may not be suspended, common function machines give uniform service.

P_1, P_2, and P_3 have meaning in the context of the isomorphic routing problem; P_4 and P_5 are natural variations of the multi-operation/multi-machine scheduling problem. We will establish complexity bounds on each of these problems.

We organize this paper as follows. In Section 2, we study the complexity of all the problems above, save P_2. P_1 is shown to be $O(k)$, while the other variations are shown to be NP-complete. Section 3 develops an algorithm for problem P_2, and Section 4 develops an $O(k \log k)$ implementation of the algorithm. Section 5 presents our conclusions. The Appendix proves some useful lemmas in detail.

2 Complexity results for P_1, P_3, P_4, and P_5

Problem P_1 allows only one machine to be executing at a time. The solution is trivial. Step through the jobs sequentially, giving exhaustive service to one task, and then the other, in each case selecting the machine which serves the task most quickly. $2k$ comparisons are performed in the course of selecting machines, giving the algorithm complexity $O(k)$. While not very interesting in the scheduling context, the situation follows from the isomorphic routing problem under the constraint that at any step, only one communication port can be active. This is a seemingly natural constraint, but is not always required. For example, the Thinking Machines CM-2 is able to communicate on all ports simultaneously [1]. Indeed, the problem studied in [1] is similar to ours, in that it seeks to schedule communication (albeit irregular, as opposed to our isomorphic assumption) on the CM-2's hypercube communication network.

Next we show that P_3, P_4 and P_5 are NP-complete. First consider P_4, where common function machines give uniform service. Assume that machines M_1, M_2 are identical, as are M_3, M_4. There are k jobs, $J_1, J_2, \ldots, J_k$. Each job J_i consists of two tasks X_i and Y_i, where X_i can only be executed by M_1 and M_2, taking x_i time units on either machine, and Y_i can only be executed by M_3 and M_4, taking y_i time units on either machine. A job may be suspended, but may never have both its tasks receiving service simultaneously. Our goal is to find a schedule with the minimum makespan C_{max}. We shall next prove that whether we allow preemption of tasks or not, the problem is always NP-complete. Note that the NP-completeness of this formulation (an open shop scheduling problem of identical back-up machines with or without preemption) implies the intractability of all general multi-operation/multi-machine scheduling problems.

THEOREM 1 *P_4 is NP-complete.*

PROOF. Consider the corresponding decision problem, in which given a bound B, we are asked whether there is a schedule with $C_{max} \leq B$. For any instance of the NP-complete problem PARTITION [2], given $A = \{a_1, a_2, \ldots, a_k\}$ (positive integers) we construct an instance of the decision problem, in which there are $k+2$ jobs, $x_i = a_i$ and $y_i = 0$ for $i = 1, 2, \ldots, k$, $x_{k+1} = x_{k+2} = \frac{1}{2}\sum_{i=1}^{k} a_i + 1$ and $y_{k+1} = y_{k+2} = 0$, and finally $B = \sum_{i=1}^{k} a_i + 1$. We claim that there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$ iff there is a schedule with $C_{max} \leq B$ for the instance defined.

If there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$ (for notational simplicity assume that $A' = \{a_1, a_2, \ldots, a_h\}$), then we can construct a schedule with $C_{max} = B$ as shown in

FIG. 2. Even though the schedule constructed does not preempt any task, it is also feasible for the instance that allows preemption since non-preemption is considered as a special case of preemption. As a matter of fact, the schedule in FIG. 2 is the best possible since for any feasible schedule $C_{max} \geq \lceil \frac{1}{2} \sum_{i=1}^{k+2} x_i \rceil = \sum_{i=1}^{k} a_i + 1 = B$.

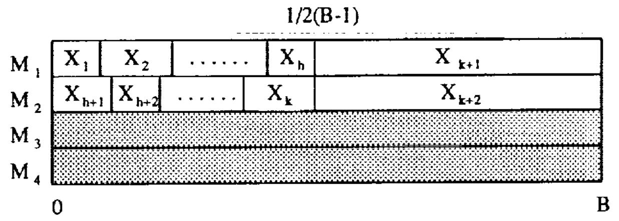


FIG. 2. A schedule with $C_{max} = B$ for the instance of the decision problem of P_4.

If there exists a schedule with $C_{max} = B$, then the two big tasks X_{k+1} and X_{k+2} cannot be scheduled on one machine since otherwise $C_{max} \geq x_{k+1} + x_{k+2} = \sum_{i=1}^{k} a_i + 2 > B$. Without loss of generality, assume X_{k+1} is scheduled on M_1 and X_{k+2} is scheduled on M_2. For the remaining k X-type tasks $X_1, \ldots, X_k$, because $\sum_{i=1}^{k} x_i = \sum_{i=1}^{k} a_i = 2B - (x_{k+1} + x_{k+2})$, M_1 and M_2 are not idle from time 0 to time B. Without loss of generality, assume tasks $X_1, \ldots, X_h$ are scheduled on M_1, and tasks $X_{h+1}, \ldots, X_k$ are scheduled on M_2. We have $\sum_{i=1}^{h} x_i = \sum_{i=h+1}^{k} x_i = \frac{1}{2} \sum_{i=1}^{k} a_i$. This is true regardless of whether preemption of tasks is allowed or not. So there exists $A' = \{a_1, \ldots, a_h\} \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2} \sum_{i=1}^{k} a_i$. ∎

Now let us consider P_5, in which a job's service must be continuous, and common function machines give uniform service. The requirement of continuity does not prohibit the tasks from being broken into slices which are independently scheduled, so long as a job's execution is not interrupted. It is easy to see that the proof of Theorem 1 can be used without any change to prove the NP-completeness of problem P_5 in both cases of preemption and non-preemption. Thus we have the additional result:

THEOREM **2** *P_5 is NP-complete.*

Now suppose that a job's service must be continuous, and that common function machines give complementary service. We assume that a task can be broken into unit-time slices. This formulation corresponds directly to an isomorphic routing problem where we require that once begun, a message continues to move at each step until it reaches its destination. It turns out that this variation is also intractable.

THEOREM **3** *P_3 is NP-complete.*

PROOF. Consider the corresponding decision problem, in which given a bound B, we are asked whether there is a schedule with $C_{max} \leq B$. For any instance of the NP-complete problem PARTITION, given $A = \{a_1, a_2, \ldots, a_k\}$ (positive integers) we construct an instance of the decision problem as follows. Let m and n be two integers much larger than $\sum_{i=1}^{k} a_i + 2$. Given four machines M_1, M_2, M_3, M_4, where M_1 and M_2 are identical in function but give complementary service (x and $m - x$ for workload x respectively), as do M_3 and M_4 (y and $n - y$ for workload y respectively). There are $k + 4$ jobs, $J_1, J_2, \ldots, J_{k+4}$, each of which consists of an X-type task and a Y-type task. Let $x_i = a_i$ and $y_i = 0$ for $i = 1, 2, \ldots, k$, $x_{k+1} = m - \frac{1}{2}\sum_{i=1}^{k} a_i$ and $y_{k+1} = \frac{1}{2}\sum_{i=1}^{k} a_i + 1$, $x_{k+2} = m - \frac{1}{2}\sum_{i=1}^{k} a_i$ and $y_{k+2} = n - \frac{1}{2}\sum_{i=1}^{k} a_i - 1$, $x_{k+3} = 1$ and $y_{k+3} = \frac{1}{2}\sum_{i=1}^{k} a_i$, $x_{k+4} = m - 1$ and $y_{k+4} = n - \frac{1}{2}\sum_{i=1}^{k} a_i$. Finally, let $B = \sum_{i=1}^{k} a_i + 1$. We claim that there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$ iff there is a schedule with $C_{max} \leq B$ for the instance defined.

If there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$ (for notational simplicity, assume $A' = \{a_1, a_2, \ldots, a_h\}$), then we can construct a schedule with $C_{max} = B$ as shown in FIG. 3. As a matter of fact, the schedule in FIG. 3 is the best possible since for any feasible schedule $C_{max} \geq \lceil \frac{1}{4}\sum_{i=1}^{k+4}(\min\{x_i, m - x_i\} + \min\{y_i, n - y_i\}) \rceil = \sum_{i=1}^{k} a_i + 1 = B$.

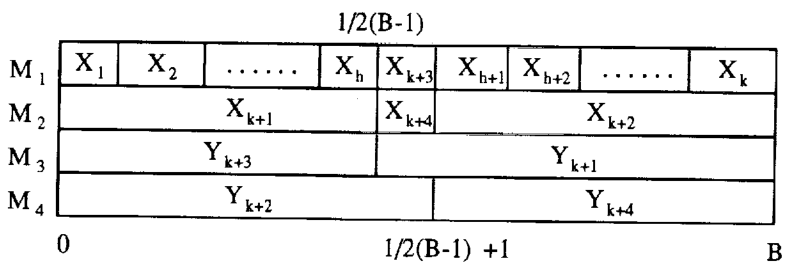


FIG. 3. A schedule with $C_{max} = B$ for the instance of the decision problem of P_3.

If there exists a schedule with $C_{max} = B$, then $X_1, X_2, \ldots, X_k$ and X_{k+3} must be scheduled on M_1, $X_{k+1}, X_{k+2}, X_{k+4}$ on M_2, Y_{k+1}, Y_{k+3} on M_3, and Y_{k+2}, Y_{k+4} on M_4. Since X_{k+1} and Y_{k+1} can not be executed simultaneously, M_2 executes X_{k+1} at the same time M_3 executes Y_{k+3}. So we say that the executions of X_{k+1} and Y_{k+3} are completely parallel. Since the executions of X_{k+3} and Y_{k+3} are continuous, so are the executions of X_{k+3} and X_{k+1}. Similarly, we can show that the executions of X_{k+4} and X_{k+2} are also continuous. How can the schedule have X_{k+3} on M_1 and X_{k+1}, X_{k+2}, and X_{k+4} on M_2 such that the continuity of X_{k+3} and X_{k+1} and the continuity of X_{k+4} and X_{k+2} are both respected? It is not hard to see that X_{k+3} must be scheduled from time $\frac{1}{2}\sum_{i=1}^{k} a_i$ to time $\frac{1}{2}\sum_{i=1}^{k} a_i + 1$. Therefore set $\{X_1, X_2, \ldots, X_k\}$ is divided into two sets of equal sums. So there exists $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \frac{1}{2}\sum_{i=1}^{k} a_i$. ∎

We are left now with the problem of analyzing P_2. This will require most of the remainder of the paper. Our approach will be to recognize that P_2 is a variation on a scheduling problem, denoted by P_2', where the decision of which machine to use for any given task is a given input parameter; the sign of x_i or y_i determines which machine to use. In the isomorphic routing problem this is equivalent to specifying the specific directions the message must travel. This might arise, for instance, if the N and E ports could be used only for sending messages, whereas the S and W ports could be used only for receiving them.

Assuming that machine usage is pre-specified, the resulting problem P_2' is related to a paper by Gonzalez and Sahni [3]. The paper studies the general open shop scheduling with preemption, and proves that $C^*_{max} = \alpha \equiv \max_{i,j}\{T_i, L_j\}$, where T_i is the sum of execution time of all tasks scheduled on machine M_i and L_j is the sum of execution time of all tasks of job J_j. To construct the optimal schedule for any instance with m machines, n jobs, and r nonzero tasks, an $O(r(\min\{r, m^2\} + m \log n))$ algorithm is presented.

It is easy to see that P_2' is in fact a special case of this open shop scheduling problem, in which parameters are integers and preemptions are only allowed at the integral points. Furthermore, we also notice that the minimum makespan for any instance of P_2', C^*_{max}, is at least $\alpha = \max_{i,j}\{T_i, L_j\} = \max\{\sum_{\forall x_i>0} x_i, \sum_{\forall x_i<0}(-x_i), \sum_{\forall y_i>0} y_i, \sum_{\forall y_i<0}(-y_i), \max_j\{|x_i| + |y_j|\}\}$. When we apply Gonzalez and Sahni's algorithm to P_2', we have an optimal preemptive schedule with $C^*_{max} = \alpha$. Since all preemptions occur at the integral points, this is actually the optimal solution to P_2'. The time complexity of Gonzalez and Sahni's algorithm when applied to P_2' is $O(k \log k)$.

In view of this result, our approach will be to take a problem instance of P_2, and determine the machine assignments that minimize the α. Gonzalez and Sahni's algorithm may then be applied to construct the actual schedule.

3 An algorithm for P_2

As pointed out in last section, solving P_2 can be reduced to the problem of finding the task-to-machine assignment that minimizes the makespan. The actual schedule can then be determined in $O(k \log k)$ time using the algorithm of Gonzalez and Sahni. In this section we develop an algorithm that makes the needed assignment.

We abstract our problem as follows. We are given two sets of items, $X = \{X_1, X_2, \ldots, X_k\}$, and $Y = \{Y_1, Y_2, \ldots, Y_k\}$, and nonnegative integers $x_1, x_2, \ldots, x_k$, $y_1, y_2, \ldots, y_k$, m, and n, where $x_i \leq m-1$ and $y_i \leq n-1$ for all i's. We must define a function $F : X \cup Y \rightarrow \mathbb{N}$ with $F(X_i) = x_i$ or $m - x_i$, and $F(Y_i) = y_i$ or $n - y_i$ such that $\alpha = \max_{i,j}\{T_i, L_j\} = \max\{\alpha_1, \alpha_2, \alpha_3\}$ is minimized, where

$$\begin{aligned}
\alpha_1 &= \max\{\sum_{\forall i(F(X_i)=x_i)} x_i, \sum_{\forall i(F(X_i)=m-x_i)} (m - x_i)\} \\
\alpha_2 &= \max\{\sum_{\forall i(F(Y_i)=y_i)} y_i, \sum_{\forall i(F(Y_i)=n-y_i)} (n - y_i)\}
\end{aligned}$$

$$\alpha_3 = \max_{\forall i}\{F(X_i) + F(Y_i)\}$$

We first describe an algorithm **A** that defines a function $f : X \cup Y \to \mathbb{N}$ with $f(X_i) = x_i$ or $m - x_i$, and $f(Y_i) = y_i$ or $n - y_i$ such that the resulting α_1 and α_2 are both minimized. Following this, we look at how F may deviate from f, and show how to modify f so as to create F.

Algorithm A

1. Sort $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_k$ in nondecreasing order, separately.
2. Do the following to each sorted list. The pseudo-code below defines $f_X : X \to \mathbb{N}$ with $f(X_i) = x_i$ or $m - x_i$ such that the resulting α_1 is minimized. To define $f_Y : Y \to \mathbb{N}$ for the minimum α_2, we simply replace the notations for the X-list by the corresponding notations for the Y-list.

```
For notational simplicity, assume $x_1, x_2, \ldots, x_k$ are in nondecreasing order;
$\alpha_1^+ \leftarrow 0$; $\alpha_1^- \leftarrow 0$; $i \leftarrow 1$; $j \leftarrow k$;
while $i \leq j$ do
  if $\alpha_1^+ + x_i \leq \alpha_1^- + (m - x_j)$
  then { $f_X(X_i) \leftarrow x_i$; $\alpha_1^+ \leftarrow \alpha_1^+ + x_i$; $i++$ }
  else { $f_X(X_j) \leftarrow m - x_j$; $\alpha_1^- \leftarrow \alpha_1^- + (m - x_j)$; $j++$ };
$\alpha_1 \leftarrow \max\{\alpha_1^+, \alpha_1^-\}$;
```

3. $f : X \cup Y \to \mathbb{N}$ is the combination of f_X and f_Y.

We recognize α_1^+ as accumulating the first term in α_1, and α_1^- as accumulating the second. Given the sorted ordering of the x_i's, the algorithm finds a *turning point* t, where $f(X_i) = x_i$ for $i \leq t$, and $f(X_i) = m - x_i$ for $i > t$; furthermore, among all such turning points the one chosen minimizes $\max\{\alpha_1^+, \alpha_1^-\}$. That this algorithm defines α_1^* follows from the fact that the optimal schedule must have this structure, for suppose not. Assume there are p and q with $1 \leq p < q \leq k$ such that $f(X_p) = m - x_p$, and $f(X_q) = x_q$. Since $x_p \leq x_q$ and $m - x_p \geq m - x_q$, it follows that $\max\{m - x_p, x_q\} \geq \max\{x_p, m - x_q\}$, so that changing the assignment for X_p and X_q does not increase α_1. We may apply this argument repeatedly until the resulting assignment exhibits a turning point, as claimed.

FIG. 4 shows an example of using algorithm **A** to compute the optimal value of α_1. The numbers in the circles are the values of function f_X of the corresponding tasks. We also illustrate α_1^+ and α_1^- as functions of index, even though the algorithm will not generate all such values we display. From now on, we shall use the diagrams similar to FIG. 4 but without the α_1^+, α_1^- values to represent the definition of f, which we will also call *assignment diagrams.*

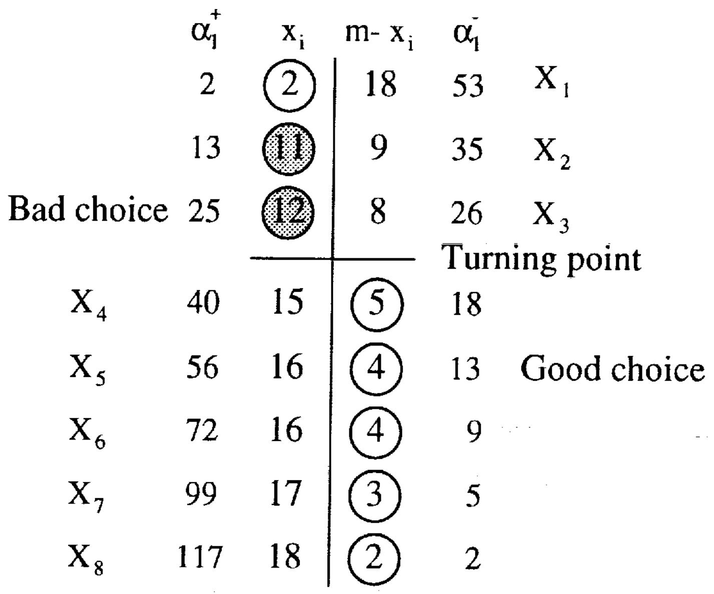


FIG. 4. An example of using algorithm **A** *to compute* α_1^*.

We see from the above discussion that α_1^* and α_2^*, the optimal values of α_1 and α_2, can be obtained by algorithm **A** in time $O(k \log k)$, while α_3^*, the optimal value of α_3, can be obtained by choosing $\min\{x_i, m - x_i\}$ for task X_i and $\min\{y_i, n - y_i\}$ for task Y_i. However, the difficulty we face is that these optimalities may not be achieved at the same time, i.e., the assignment minimizing α_1^* and α_2^* may not be consistent with the assignment minimizing α_3^*. To highlight the differences we will say that $f(X_i)$ (alt., $f(Y_i)$) is a *bad choice* if $f(X_i) \neq \min\{x_i, m - x_i\}$ (alt., $f(Y_i) \neq \min\{y_i, n - y_i\}$), and that $f(X_i)$ (alt., $f(Y_i)$) is a *disastrous choice* if $f(X_i) \neq \min\{x_i, m - x_i\}$ (alt, $f(Y_i) \neq \min\{y_i, n - y_i\}$) and $f(X_i) + f(Y_i) > \alpha^*$, where $\alpha^* = \max\{\alpha_1^*, \alpha_2^*, \alpha_3^*\}$. In the example in FIG. 4, the shaded circles represent the bad choices. It is easy to see that bad choices always form a contiguous block which includes the turning point. Without loss of generality, assume that the block of bad choices is in the left column of the assignment diagram and ends at the turning point. We observe that if f contains no disastrous choices, then $\alpha^* = \max\{\alpha_1^*, \alpha_2^*, \alpha_3^*\} \geq \max\{\alpha_1^*, \alpha_2^*\} \geq \alpha_3^*$, and $F = f$. Should f contain disastrous choices, we need to consider modifying it in order to find the function F with the minimum α.

Let us assume then that we have computed an assignment f by applying Algorithm **A** to the X list (and so find the X assignment function f_X), and to the Y list (and so find the Y assignment function f_Y), and have identified at least one disastrous choice. f may or may not be the optimal assignment F. We have developed a number of results that help us to identify jobs J_i for which it may be possible that $f(X_i) \neq F(X_i)$ or $f(Y_i) \neq F(Y_i)$. Most importantly, these results severely constrain the number of tasks whose assignment in f can differ from their assignment in F. Given f, we will identify a set of possible assignment switches to consider; the least-cost assignment among these will be the optimal assignment. We show that for the given f, only $O(k \log k)$ alternative assignments must be considered, whence the optimal assignment is found in $O(k \log k)$ time.

We proceed now by making some definitions, and stating certain results founded upon them (proofs are relegated to the Appendix). Without loss of generality, we assume $m \geq n$ in the remainder.

Given f, let B_X and B_Y be the sets of bad choices in f_X and f_Y, respectively, and D_X and D_Y be the sets of disastrous choices in f_X and f_Y, respectively. Now, in the assignment diagrams of f_X and f_Y, let X_L and Y_L be the sets of choices in the left columns of f_X and f_Y, respectively, and X_R and Y_R be the sets of choices in the right columns of f_X and f_Y, respectively. We denote the sets of tasks whose assignment differs under f and F as $U_X \subseteq X_L$, $V_X \subseteq X_R$, $U_Y \subseteq Y_L$, $V_Y \subseteq Y_R$. We use $\alpha_1(U_X, V_X)$ (alt. $\alpha_2(U_Y, V_Y)$) to denote the corresponding α_1 (alt., α_2) resulting from the switches in U_X, V_X (alt. U_Y, V_Y). Finally, we will say that assignment $f(X_i)$ (alt., $f(Y_i)$) is a *potential switch* if either $f(X_i)$ (alt., $f(Y_i)$) is a disastrous choice, or $f(X_i)$ (alt., $f(Y_i)$) is a bad choice while $f(Y_i)$ (alt., $f(X_i)$) is in V_Y (alt., V_X), and $f(X_i)+n-f(Y_i) > \alpha_2(U_Y, V_Y)$ (alt., $m-f(X_i)+f(Y_i) > \alpha_1(U_X, V_X)$).

The next three results serve to constrain the number of switches we must consider.

LEMMA 1 *If* $|B_X| \geq 3$, *then* $F = f$.

LEMMA 2 $|D_Y| \leq 2$.

LEMMA 3 $|U_X| \geq |V_X|$ *and* $|U_Y| \geq |V_Y|$, .

LEMMA 4 *All members of* U_X *and* U_Y *are potential switches.*

Now consider the implications of these results. By Lemma 1 we only have to worry about situations when $|B_X| \leq 2$. By Lemma 4 we know that U_X contains only potential switches, which are recognizable bad choices. There are at most 4 different combinations of changing or not changing the assignments of bad choices in the left column of f_X. By Lemma 3 we know that at most two assignments in the right column of f_X may change. For each fixed combination of changes to f_X's left column we need consider no more than $O(\binom{k}{2})$ pairs of possible changes to assignments in f_X's right column. We also need to consider possible changes to f_Y. Lemma 2 tells us $|D_Y| \leq 2$; Lemma 3 tells us $|V_Y| \leq |U_Y|$; Lemma 4 tells us that U_Y may contain only potential switches, which again are either disastrous choices in f_Y, or bad choices $f(Y_i)$ with $f(X_i) \in V_X$. It follows that $|V_Y| \leq |U_Y| \leq |D_Y| + |V_X| \leq 4$. This means that for every fixed combination of switched/non-switched assignments of potential switches in the left column of f_Y, we need consider no more than all switched/non-switched combinations of four good choices from the right column of f_Y. There are $O(\binom{k}{4})$ of these.

Considering all combinations of possible changes to f_X and possible changes to f_Y requires time $O(\binom{k}{2} \cdot \binom{k}{4}) = O(k^6)$.

We describe this algorithm for problem P_2 formally as follows.

1. If $m < n$, rotate the mesh by 90 degrees, and redefine the parameters in the new coordinate system.
2. Use algorithm **A** twice to define f which minimizes $\max\{\alpha_1, \alpha_2\}$
3. If the block of bad choices in an assignment diagram is not in the left column, rotate the diagram by 180 degrees, and exchange the roles of the two machines involved in the assignment.
4. If there are no disastrous choices in both f_X and f_Y, let F be f and go to step 6. Otherwise continue in step 5.
5. List all possible definitions of U_X, V_X and U_Y, V_Y. For each possible combination of U_X, V_X, U_Y, V_Y, compute its α. Let F be the function determined by the U_X, V_X, U_Y, V_Y which together result in the smallest α.
6. Use Gonzalez and Sahni's algorithm to construct the schedule with $C_{max} = \alpha$.

In this algorithm, steps 1, 3, and 4 each take $O(k)$ time, while steps 2 and 6 each take $O(k \log k)$ time. We also know that for step 5, even if we use the brute-force method of checking all possible combinations of U_X, V_X, U_Y, V_Y, the time needed is still polynomial, $O(k^6)$. In the next section, we shall show that step 5 can in fact be implemented in time $O(k \log k)$, thus yielding an $O(k \log k)$ algorithm for P_2.

4 An $O(k \log k)$ implementation of the algorithm

The previous section demonstrated that the routing problem has polynomial complexity. We *can* drive the asymptotic complexity to $O(k \log k)$, but at the price of tremendous complication in the algorithm. Our results may be primarily of theoretical interest; our algorithm can be implemented, but suffers from a lack of elegance. One hopes that additional work on the problem may yield a more intuitive solution.

Let us now consider the following three cases: $|B_X| = 0$, $|B_X| = 1$, and $|B_X| = 2$. We shall prove that in each case the function F, which minimizes α, can be obtained in $O(k \log k)$ by switching some assignments in the function f. We will use the next three lemmas to help reduce the number of possible combinations we must consider. Their proofs can be found in the Appendix.

LEMMA 5 *If* $|B_X| = 0$, *then* $|U_X| = |V_X| = 0$ *and* $|D_Y| \leq 2$.

Lemma 6 *If $|B_X| = 1$, then $|V_X| \leq |U_X| \leq 1$ and $|D_Y| \leq 2$. Furthermore, if $D_Y = \{f(Y_1), f(Y_2)\}$, then $\alpha^* < f(X_1) + f(X_2)$ and one of $f(X_1)$ and $f(X_2)$ is the largest bad choice in f_X.*

Lemma 7 *If $|B_X| = 2$, then $|D_X| \leq 1$ and $|D_Y| \leq 1$. Furthermore, if $D_X = \{f(X_1)\}$, then $D_Y = \{f(Y_1)\}$; if $D_Y = \{f(Y_1)\}$ and $f(X_1) \notin B_X$, then $f(X_1)$ must be in the right column in the assignment diagram of f_X.*

We first consider **Case 1:** $|B_X| = 0$.

By Lemma 5, $U_X = V_X = \phi$. Since $V_X = \phi$, only disastrous choices in f_Y can be potential switches for U_Y. We consider two subcases: (a) $|D_Y| = 1$; and (b) $|D_Y| = 2$.

(a) If $D_Y = \{f(Y_1)\}$, then $f(Y_1)$ is the only potential switch in f_Y. Consider the following possible combinations of U_X, V_X and U_Y, V_Y, each of which determines a feasible definition of F, and choose the one with the smallest α to be F. The entire process takes $O(k)$ time.

#	U_X	V_X	U_Y	V_Y	Time
1	ϕ	ϕ	ϕ	ϕ	$O(1)$
2	ϕ	ϕ	$\{f(Y_1)\}$	ϕ	$O(1)$
3	ϕ	ϕ	$\{f(Y_1)\}$	$\{f(Y_i)\}, \forall f(Y_i) \in Y_R$	$O(k)$

(b) If $D_Y = \{f(Y_1), f(Y_2)\}$, then $f(Y_1)$ and $f(Y_2)$ are the two potential switches in f_Y. Without loss of generality, assume $f(X_1) + f(Y_1) \geq f(X_2) + f(Y_2)$. This means that if $|U_Y| = 1$, it must contain $f(Y_1)$, not $f(Y_2)$. Consider the following feasible definitions of F.

#	U_X	V_X	U_Y	V_Y	Time
1	ϕ	ϕ	ϕ	ϕ	$O(1)$
2	ϕ	ϕ	$\{f(Y_1)\}$	ϕ	$O(1)$
3	ϕ	ϕ	$\{f(Y_1)\}$	$\{f(Y_i)\}, \forall f(Y_i) \in Y_R$	$O(k)$
4	ϕ	ϕ	$\{f(Y_1), f(Y_2)\}$	ϕ	$O(1)$
5	ϕ	ϕ	$\{f(Y_1), f(Y_2)\}$	$\{f(Y_i)\}, \forall f(Y_i) \in Y_R$	$O(k)$
6	ϕ	ϕ	$\{f(Y_1), f(Y_2)\}$	$\{f(Y_i), f(Y_j)\}, \forall f(Y_i), f(Y_j) \in Y_R$	$O(k \log k)$

In the sixth situation, if we check all combinations of $f(Y_i), f(Y_j) \in Y_R$ for V_Y, there will be $O(k^2)$ possibilities. However, not all combinations need to be examined. Our goal is to choose $f(Y_j) \in Y_R$ for each fixed $f(Y_i) \in Y_R$ so as to minimize $\max\{\alpha_2(U_Y, V_Y), f(X_j) + n - f(Y_j)\}$, where $\alpha_2(U_Y, V_Y) = \alpha_2^* + 2n - f(Y_1) - f(Y_2) - f(Y_i) - f(Y_j)$. First, sort in time $O(k \log k)$ all $f(Y_j) \in Y_R$ according to the value $f(X_j) + n - f(Y_j)$ nondecreasingly. Then in the sorted list discard those choices no greater than their left neighbors, yielding a list of $f(Y_j)$'s sorted by nondecreasing $f(X_j) + n - f(Y_j)$ and nonincreasing $\alpha_2^* + 2n -$

$f(Y_1) - f(Y_2) - f(Y_i) - f(Y_j)$ for any fixed $f(Y_i)$. This step takes $O(k)$ time. Finally, for each fixed $f(Y_i)$, perform a binary search in the list to locate the $f(Y_j)$ with the minimum $\max\{\alpha_2^* + 2n - f(Y_1) - f(Y_2) - f(Y_i) - f(Y_j), f(X_j) + n - f(Y_j)\}$, taking $O(k \log k)$ for all $f(Y_i)$'s. We can then check the $|Y_R|$ feasible definitions of F with $V_Y = \{f(Y_i), f(Y_j)\}$ as defined above.

Case 2: $|B_X| = 1$.

By Lemma 6, when $D_Y = \{f(Y_1), f(Y_2)\}$, one of $f(X_1)$ and $f(X_2)$ must be a bad choice in f_X, which implies that it is a disastrous choice. Therefore, if $|D_Y| = 2$, then $|D_X| = 1$. We consider four subcases: (a) $|D_X| = 0$ and $|D_Y| = 1$; (b) $|D_X| = 1$ and $|D_Y| = 0$; (c) $|D_X| = 1$ and $|D_Y| = 1$; and (d) $|D_X| = 1$ and $|D_Y| = 2$. We notice that in any situation with $V_X = \phi$, only disastrous choices in f_Y can be potential switches for U_Y, and that whether $|D_Y| = 0$ or 1 or 2, we can use the same method as in Case 1 to determine F in $O(k \log k)$ time. Let us now assume $|V_X| = 1$, i.e., $V_X = \{f(X_i)\}, \forall f(X_i) \in X_R$, which also implies $U_X = B_X$.

(a) If $D_X = \phi$, and $D_Y = \{f(Y_1)\}$, then $f(X_1)$ can not be a bad choice. Assuming $B_X = \{f(X_2)\}$, we have $U_X = \{f(X_2)\}$. Because $f(X_2)$ is a potential switch that is not a disastrous choice, we have $f(Y_2) \in V_Y$ and $f(Y_1) \in U_Y$. Note that $f(Y_i)$ may also be in U_Y if $f(Y_i)$ is a bad choice. Consider the following feasible definitions of F.

#	U_Y	V_Y	Time
1	$\{f(Y_1)\}$	$\{f(Y_2)\}$	$O(k)$
2	$\{f(Y_1), f(Y_i)\}, i \neq 1$	$\{f(Y_2)\}$	$O(k)$
3	$\{f(Y_1), f(Y_i)\}, i \neq 1$	$\{f(Y_2), f(Y_j)\}, \forall f(Y_j) \in Y_R, j \neq 2$	$O(k \log k)$

In the second and third situations, we only need to check those feasible definitions of F with $V_X = \{f(X_i)\}$, for which $f(Y_i) \in B_Y$ and $m - f(X_i) + f(Y_i) > \alpha_1(U_X, V_X) = \alpha_1^* + m - f(X_2) - f(X_i)$. In the third situation, we can avoid checking all $O(k^2)$ combinations of $f(X_i) \in X_R$ with $i \neq 1$ and $f(Y_j) \in Y_R$ with $j \neq 2$ by using the same method developed in the sixth situation of subcase (b) in Case 1.

(b) If $D_X = \{f(X_1)\}$, and $D_Y = \phi$, then $U_X = \{f(X_1)\}$, and $V_X = \{f(X_i)\}, \forall f(X_i) \in X_R$. Consider the following feasible definitions of F.

#	U_Y	V_Y	Time
1	ϕ	ϕ	$O(k)$
2	$\{f(Y_i)\}$	ϕ	$O(k)$
3	$\{f(Y_i)\}$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R$	$O(k \log k)$

In the second and third situations, we only need to check those feasible definitions of F with $V_X = \{f(X_i)\}$, for which $f(Y_i) \in B_Y$ and $m - f(X_i) + f(Y_i) > \alpha_1(U_X, V_X) =$

$\alpha_1^* + m - f(X_1) - f(X_i)$. In the third situation, we use a method similar to that in the sixth situation of subcase (b) in Case 1 to avoid checking all $O(k^2)$ combinations of $f(X_i) \in X_R$ and $f(Y_j) \in Y_R$. The only differences are that $\alpha_2(U_Y, V_Y) = \alpha_2^* + n - f(Y_i) - f(Y_j)$ and that if $f(Y_1) \in Y_R$ we use $m - f(X_1) + n - f(Y_1)$ instead of $f(X_1) + n - f(Y_1)$ for choice $f(Y_1)$ in the sorting part.

(c) If $D_X = \{f(X_1)\}$, and $D_Y = \{f(Y_h)\}$, where h is a fixed index equal to or not equal to 1, then $U_X = \{f(X_1)\}$, $V_X = \{f(X_i)\}, \forall f(X_i) \in X_R$. Note that $f(Y_i) \in U_Y$ only when $f(Y_i) \in B_Y$ and $m - f(X_i) + f(Y_i) > \alpha_1(U_X, V_X) = \alpha_1^* + m - f(X_1) - f(X_i)$. Consider the following feasible definitions of F.

#	U_Y	V_Y	Time
1	ϕ	ϕ	$O(k)$
2	$\{f(Y_i)\}$	ϕ	$O(k)$
3	$\{f(Y_i)\}$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R$	$O(k \log k)$
4	$\{f(Y_h)\}$	ϕ	$O(k)$
5	$\{f(Y_h)\}$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R, j \neq i$	$O(k \log k)$
6	$\{f(Y_h), f(Y_i)\}, i \neq h$	ϕ	$O(k)$
7	$\{f(Y_h), f(Y_i)\}, i \neq h$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R$	$O(k \log k)$
8	$\{f(Y_h), f(Y_i)\}, i \neq h$	$\{f(Y_j), f(Y_l)\}, \forall f(Y_j), f(Y_l) \in Y_R$	$O(k \log k)$

The reason why $j \neq i$ in the fifth situation is that if $f(Y_i) \in Y_R$ and we let $V_Y = \{f(Y_i)\}$, then $m - f(X_i) + n - f(Y_i) \geq f(X_1) + f(Y_h) \geq \max\{f(X_1) + f(Y_1), f(X_h) + f(Y_h)\}$, which indicates the resulting assignment is even worse than the original assignment without any switches. The method used in the sixth situation of subcase (b) in Case 1 can be applied to the third, fifth and seventh situations in this subcase to achieve the $O(k \log k)$ bound. In the eighth situation, if we check all combinations of $f(X_i) \in X_R$ and $f(Y_j), f(Y_l) \in Y_R$, there will be $O(k^3)$ possibilities. We will show that not all combinations need to be examined. Our goal is to choose $f(Y_j), f(Y_l) \in Y_R$ for each fixed $f(X_i) \in X_R$ so as to minimize $\max\{\alpha_2(U_Y, V_Y), f(X_j) + n - f(Y_j), f(X_l) + n - f(Y_l)\}$, where $\alpha_2(U_Y, V_Y) = \alpha_2^* + 2n - f(Y_1) - f(Y_i) - f(Y_j) - f(Y_l)$. Without loss of generality, assume $f(X_j) + n - f(Y_j) \geq f(X_l) + n - f(Y_l)$. First, sort in $O(k \log k)$ time $f(Y_j) \in Y_R$ according to the value $f(X_j) + n - f(Y_j)$ nondecreasingly. Second, in the sorted list, for each $f(Y_j)$, except the first one, let $f(Y_l)$ be the largest choice among those on the left side of $f(Y_j)$. This can easily be done in $O(k)$ time. Now, we have a list of $|Y_R| - 1$ choice pairs $f(Y_j), f(Y_l)$ ordered according to the value $f(X_j) + n - f(Y_j)$ nondecreasingly. Third, in $O(k)$ time discard those pairs with their sum $f(Y_j) + f(Y_l)$ no greater than that of their left neighbors in the list. Finally, for each $f(X_i) \in X_R$, use binary search to find the pair $f(Y_j), f(Y_l)$ with the minimum $\max\{\alpha_2^* + 2n - f(Y_1) - f(Y_i) - f(Y_j) - f(Y_l), f(X_j) + n - f(Y_j), f(X_l) + n - f(Y_l)\}$ among the remaining pairs in the list, which altogether takes $O(k \log k)$ time. In the above process, if $f(Y_1) \in Y_R$, use $m - f(X_1) + n - f(Y_1)$ instead of $f(X_1) + n - f(Y_1)$ for choice $f(Y_1)$ in the sorting part.

(d) If $D_X = \{f(X_1)\}$, and $D_Y = \{f(Y_1), f(Y_2)\}$, then by Lemma 6 $f(X_1)+f(X_2) > \alpha^*$, and $f(X_1)$ and $f(X_2)$ are in the different columns of f_X. Since $f(X_1)$ is the bad choice, then $f(X_2) \in X_R$. By assumption $U_X = \{f(X_1)\}$, and $V_X = \{f(X_i)\}, \forall f(X_i) \in X_R$. We notice the following properties of the feasible definitions of F.

First, for the situation in which $V_X = \{f(X_2)\}$, the number of feasible definitions we need to check is bounded by $O(k \log k)$ time. In the following discussion, we assume $V_X = \{f(X_i)\}$, where $i \neq 2$.

Second, we do not need to consider those situations where $V_X = \{f(X_i)\}$, for which $f(Y_i) \in B_Y$. Assume $V_X = \{f(X_i)\}$, for which $i \neq 2$ and $f(Y_i) \in B_Y$. We have $f(X_2) + f(Y_2) > \alpha^* \geq \alpha_2^* \geq f(Y_1) + f(Y_2) + f(Y_i)$, therefore $f(X_2) > f(Y_1) + f(Y_i)$. We also have $f(X_2) + f(Y_2) > \alpha^* \geq \alpha_1^* \geq f(X_2) + f(X_i)$, therefore $f(Y_2) > f(X_i)$. We can show that $m - f(X_i) + n - f(Y_i) > f(X_1) + f(Y_1)$, because $f(X_1) \leq m - f(X_2) < m - f(Y_1) - f(Y_i) < m - f(Y_1) - f(Y_i) + n - f(X_i)$. We can then show that $m - f(X_i) + n - f(Y_i) > f(X_2) + f(Y_2)$, because $f(X_2) + f(X_i) \leq \alpha^* < f(X_1) + f(Y_1) < f(X_1) + f(X_2) - f(Y_i) \leq m - f(Y_i) \leq m - \min\{f(Y_2), f(Y_i)\} < m - \min\{f(Y_2), f(Y_i) + n - \max\{f(Y_2), f(Y_i)\} = m + n - f(Y_2) - f(Y_i)$. This means that $m - f(X_i) + f(Y_i) > m - f(X_i) + n - f(Y_i) > \max\{f(X_1) + f(Y_1), f(X_2) + f(Y_2)\}$, which indicates that whether we switch $f(Y_i)$ or not the resulting assignment is always worse than the original assignment without any switches.

Taking the above facts into account, we only need to consider the following feasible definitions of F. Without loss of generality, assume $h = 1$ or 2, where $f(X_h) + f(Y_h) = \max\{f(X_1) + f(Y_1), f(X_2) + f(Y_2)\}$. This means that if $|U_Y| = 1$ then $U_Y = \{f(Y_h)\}$.

#	U_Y	V_Y	Time
1	ϕ	ϕ	$O(k)$
2	$\{f(Y_h)\}$	ϕ	$O(k)$
3	$\{f(Y_h)\}$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R, j \neq i$	$O(k \log k)$
4	$\{f(Y_1), f(Y_2)\}$	ϕ	$O(k)$
5	$\{f(Y_1), f(Y_2)\}$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R, j \neq i$	$O(k \log k)$
6	$\{f(Y_1), f(Y_2)\}$	$\{f(Y_j), f(Y_l)\}, \forall f(Y_j), f(Y_l) \in Y_R, j, l \neq i$	$O(k \log k)$

Similar to the previous subcases, the number of situations we need to check in this subcase is also bounded by $O(k \log k)$.

Case 3: $|B_X| = 2$.

By Lemma 7, $|D_X| \leq 1$ and $|D_Y| \leq 1$, and if there is a disastrous choice in f_X, there is also a disastrous choice in f_Y. We consider two subcases: (a) $|D_X| = 0$ and $|D_Y| = 1$; and (b) $|D_X| = 1$ and $|D_Y| = 1$.

(a) If $D_X = \phi$, and $D_Y = \{f(Y_1)\}$, then by Lemma 7 $f(X_1) \in X_R$, and V_Y, if nonempty, only contains $f(Y_i)$ with $f(X_i) \notin B_X$. Otherwise, $f(X_i)+n-f(Y_i) > m-f(X_i)+n-f(Y_i) \geq f(X_1) + f(Y_1)$, which implies that the resulting α is even larger than that for f. So there is no potential switches in f_X. Consider the following feasible definitions of F.

#	U_X	V_X	U_Y	V_Y	Time
1	ϕ	ϕ	ϕ	ϕ	$O(1)$
2	ϕ	ϕ	$\{f(Y_1)\}$	ϕ	$O(1)$
3	ϕ	ϕ	$\{f(Y_1)\}$	$\{f(Y_i)\}, \forall f(Y_i) \in Y_R$ with $f(X_i) \notin B_X$	$O(k)$

(b) If $D_X = \{f(X_1)\}$, and $D_Y = \{f(Y_1)\}$, then we notice the following properties of the feasible definitions of F.

First, we do not have to consider the situation in which both $f(X_1)$ and $f(Y_1)$ are switched. Because assuming $f(X_2)$ is the other bad choice in f_X, $m - f(X_1) + n - f(Y_1) < f(X_1) + n - f(Y_1) < f(X_1) + f(X_2) \leq \alpha^*$, which suggests that switching just $f(Y_1)$ is already good enough, why bother to switch both $f(X_1)$ and $f(Y_1)$?

Second, $|U_X| \leq 1$. Assume $U_X = \{f(X_1), f(X_2)\}$. This case happens only when $f(Y_2) \in V_Y$, $f(X_3) \in V_X$ for some $f(Y_3) \in U_Y$, and $f(X_2) + n - f(Y_2) > \alpha_2(U_Y, V_Y) = \alpha_2^* + n - f(Y_3) - f(Y_2)$. Then $f(Y_1) + f(Y_3) \leq \alpha_2^* < f(X_2) + f(Y_3)$. So $f(Y_1) < f(X_2)$. On the other hand, $f(X_1) + f(Y_1) > \alpha^* \geq f(X_1) + f(X_2)$. So $f(Y_1) > f(X_2)$. A contradiction!

Taking the above facts into account, we only need to consider the following feasible definitions of F.

#	U_X	V_X	U_Y	V_Y	Time
1	ϕ	ϕ	ϕ	ϕ	$O(1)$
2	ϕ	ϕ	$\{f(Y_1)\}$	ϕ	$O(1)$
3	ϕ	ϕ	$\{f(Y_1)\}$	$\{f(Y_i)\}, \forall f(Y_i) \in Y_R$	$O(k)$
4	$\{f(X_2)\}$	ϕ	$\{f(Y_1)\}$	$\{f(Y_2)\}$	$O(1)$
5	$\{f(X_2)\}$	$\{f(X_i)\}, \forall f(X_i) \in X_R$	$\{f(Y_1)\}$	$\{f(Y_2)\}$	$O(k)$
6	$\{f(X_1)\}$	ϕ	ϕ	ϕ	$O(1)$
7	$\{f(X_1)\}$	$\{f(X_i)\}, \forall f(X_i) \in X_R$	ϕ	ϕ	$O(k)$
8	$\{f(X_1)\}$	$\{f(X_i)\}, \forall f(X_i) \in X_R$	$\{f(Y_i)\}$	ϕ	$O(k)$
9	$\{f(X_1)\}$	$\{f(X_i)\}, \forall f(X_i) \in X_R$	$\{f(Y_i)\}$	$\{f(Y_j)\}, \forall f(Y_j) \in Y_R$	$O(k)$

We check the fourth and fifth situations only when $f(Y_2) \in Y_R$, and $f(X_2) + n - f(Y_2) > \alpha_1(U_Y, V_Y) = \alpha_2^* + n - f(Y_1) - f(Y_2)$. In the eighth and ninth situations, we only need to check those combinations with $V_X = \{f(X_i)\}$, for which $f(Y_i) \in B_Y$ and $m - f(X_i) + f(Y_i) > \alpha_1(U_X, V_X) = \alpha_1^* + m - f(X_1) - f(X_i)$. We shall prove that there is at most one such $f(X_i)$. Assume there are two, say, $f(X_i)$ and $f(X_j)$. Then $m - f(X_i) + f(Y_i) > \alpha_1^* + m - f(X_1) -$

$f(X_i)$. So $f(X_1) + f(X_2) \leq \alpha_1^* < f(X_1) + f(Y_i)$. Therefore $f(X_2) < f(Y_i)$. Similarly, we have $f(X_2) < f(Y_j)$. However, $f(X_1) + f(Y_1) > \alpha^* \geq \alpha_2^* \geq f(Y_1) + f(Y_i) + f(Y_j)$. We have $m > f(X_1) > f(Y_i) + f(Y_j) > 2f(X_2)$. So $f(X_2) < \frac{m}{2}$, which is a contradiction to that $f(X_2)$ is a bad choice. In the eighth situation, we spend $O(k)$ to find the $f(X_i) \in X_R$, if it exists, and $O(1)$ to check the corresponding situation. In the ninth situation, we spend $O(k)$ to find the $f(X_i) \in X_R$, if it exists, and spend $O(k)$ to check the feasible definitions with $V_Y = \{f(Y_j)\}, \forall f(Y_j) \in Y_R$.

5 Conclusion

This paper studies a problem of routing messages on an SIMD parallel architecture whose processing elements (PE) are connected as a toroidal mesh. In our problem the sets of messages processors send are isomorphic, meaning that if some processor i has a message to send which must traverse x_i PEs in the East-West dimension and y_i PEs in the North-South, then all PEs have a message to send with identical routing offsets. We examine variants of the problem having differing assumptions concerning simultaneous use of communication channels, and the ability to buffer a message temporarily en-route. Our solution approach is to view the problem as a scheduling problem, related to a previously studied open shop scheduling problem. Our results provide new results not only on the motivating routing problem, but on a new class of scheduling problems as well.

A spectrum of complexities are obtained, from linear in the number of messages (k) per processor to NP-complete. The variation where all ports may be used simultaneously and messages may be buffered en-route is of particular interest; we first show quickly why the problem has a polynomial solution, and then do an extensive case analysis to show that the complexity is $O(k \log k)$. The case analysis lacks elegance; our hope is that future work may provide a more direct solution to the problem.

Appendix

LEMMA 1 *If* $|B_X| \geq 3$, *then* $F = f$.

PROOF. We shall prove that $D_X = \phi$ and $D_Y = \phi$. Suppose that $f(X_1)$, $f(X_2)$, $f(X_3)$,... are the bad choices in f_X, and that $f(X_1)$ is the largest among all. Assume that there is at least one disastrous choice in f_X (alt., f_Y), say, $f(X_i)$ (alt., $f(Y_i)$). Then $f(X_i) + f(Y_i) > \alpha^* \geq \alpha_1^* \geq f(X_1) + f(X_2) + f(X_3)$. So $f(Y_i) > (f(X_1) - f(X_i)) + f(X_2) + f(X_3) \geq f(X_2) + f(X_3) > 2 \times \frac{m}{2} = m$, which is impossible. ∎

LEMMA 2 $|D_Y| \leq 2$.

PROOF. Assume $D_Y = \{f(Y_1), f(Y_2), f(Y_3), \ldots\}$. Then at least two of $f(X_1)$, $f(X_2)$, $f(X_3)$ are in the same column in the assignment diagram for f_X, say, $f(X_1)$ and $f(X_2)$. Since both $f(Y_1)$ and $f(Y_2)$ are disastrous choices, we have $f(X_1) + f(Y_1) > \alpha^*$, and $f(X_2) + f(Y_2) > \alpha^*$, therefore,

$$f(X_1) + f(Y_1) + f(X_2) + f(Y_2) > 2\alpha^*.$$

However, we know $f(X_1) + f(X_2) < \alpha_1^* \leq \alpha^*$, and $f(Y_1) + f(Y_2) \leq \alpha_2^* \leq \alpha^*$, therefore,

$$f(X_1) + f(X_2) + f(Y_1) + f(Y_2) < 2\alpha^*.$$

This is a contradiction. ∎

LEMMA 3 *$|U_X| \geq |V_X|$ and $|U_Y| \geq |V_Y|$.*

PROOF. We only prove $|U_X| \geq |V_X|$, since the proof of $|U_Y| \leq |V_Y|$ is totally symmetric and hence can be omitted. For notational simplicity, we ignore the subscript X in the discussion below.

Assume $|U| < |V|$. Define any $V' \subset V$ with $|V'| = |U|$. Let $\alpha_1(U,V)$ be the corresponding α_1 resulting from switches in U and V, and $\alpha_1(U,V')$ be the corresponding α_1 resulting from switches in U and V'. Let $\alpha_1^* = \max\{\alpha_1^+, \alpha_1^-\}$, where $\alpha_1^+ = \sum_{f(X_i)=x_i} x_i$, and $\alpha_1^- = \sum_{f(X_i)=m-x_i}(m - x_i)$.

$$\begin{aligned}
\alpha_1(U,V) &= \max\{\alpha_1^+ - \textstyle\sum_U f(X_i) + \sum_V (m - f(X_i)), \alpha_1^- - \sum_V f(X_i) + \sum_U (m - f(X_i))\} \\
&= \alpha_1^+ - \textstyle\sum_U f(X_i) + \sum_V (m - f(X_i)) \\
&\quad (\text{Since } \alpha_1^+ - \alpha_1^- > m(|U| - |V|).) \\
\alpha_1(U,V') &= \max\{\alpha_1^+ - \textstyle\sum_U f(X_i) + \sum_{V'} (m - f(X_i)), \alpha_1^- - \sum_{V'} f(X_i) + \sum_U (m - f(X_i))\} \\
&= \alpha_1^* - \textstyle\sum_U f(X_i) + \sum_{V'} (m - f(X_i)) \\
&\leq \alpha_1^+ - \textstyle\sum_U f(X_i) + \sum_V (m - f(X_i)) \\
&\quad (\text{Since } \alpha_1^- - \alpha_1^+ \leq \textstyle\sum_{V-V'} (m - f(X_i)) \text{ if } \alpha_1^- > \alpha_1^+.) \\
&= \alpha_1(U,V).
\end{aligned}$$

Therefore, $\alpha_1(U,V') \leq \alpha_1(U,V)$, and it has fewer bad choices. Why not try $\alpha_1(U,V')$? In other words, the choices in $V - V'$ do not have to be switched to the opposite column since this does not lower α_1, and instead creates some new bad choices. ∎

LEMMA 4 *All members of U_X and U_Y are potential switches.*

PROOF. As declared earlier, we only prove the lemma for U_X, and omit the subscript X. If $|U| = |V|$, and U contains some choices that are not potential switches, let U' be the set of potential switches in U, and V' be any subset of V with $|V'| = |U'|$.

$$\begin{aligned}
\alpha_1(U,V) &= \max\{\alpha_1^+ - \textstyle\sum_U f(X_i) + \sum_V(m - f(X_i)), \alpha_1^- - \sum_V f(X_i) + \sum_U(m - f(X_i))\} \\
&= \alpha_1^* - \textstyle\sum_U f(X_i) + \sum_V(m - f(X_i)) \\
\alpha_1(U',V') &= \max\{\alpha_1^+ - \textstyle\sum_{U'} f(X_i) + \sum_{V'}(m - f(X_i)), \alpha_1^- - \sum_{V'} f(X_i) + \sum_{U'}(m - f(X_i))\} \\
&= \alpha_1^* - \textstyle\sum_{U'} f(X_i) + \sum_{V'}(m - f(X_i)) \\
&\le \alpha_1^* - \textstyle\sum_U f(X_i) + \sum_V(m - f(X_i)) \\
&\quad (\text{Since } \textstyle\sum_{U-U'} f(X_i) \le \sum_{V-V'}(m - f(X_i)).) \\
&= \alpha_1(U,V).
\end{aligned}$$

Therefore, $\alpha_1(U',V') \le \alpha_1(U,V)$, and U' does not contain any unnecessary switches.

If $|U| > |V|$, and U contains some choices that are not potential switches, let U' be the set of potential switches in U, and V' be any subset of V with $|V'| = \max\{0, |V| - |U - U'|\}$.

$$\begin{aligned}
\alpha_1(U,V) &= \max\{\alpha_1^+ - \textstyle\sum_U f(X_i) + \sum_V(m - f(X_i), \alpha_1^- - \sum_V f(X_i) + \sum_U(m - f(X_i))\} \\
&= \alpha_1^- - \textstyle\sum_V f(X_i) + \sum_U(m - f(X_i)) \\
&\quad (\text{Since } \alpha_1^+ - \alpha_1^- < m(|U| - |V|).) \\
\alpha_1(U',V') &= \max\{\alpha_1^+ - \textstyle\sum_{U'} f(X_i) + \sum_{V'}(m - f(X_i)), \alpha_1^- - \sum_{V'} f(X_i) + \sum_{U'}(m - f(X_i))\} \\
&= \alpha_1^- - \textstyle\sum_{V'} f(X_i) + \sum_{U'}(m - f(X_i)) \\
&\quad (\text{Since } \alpha_1^+ - \alpha_1^- < m(|U'| - |V'|).) \\
&\le \alpha_1^- - \textstyle\sum_V f(X_i) + \sum_U(m - f(X_i)) \\
&\quad (\text{Since } \textstyle\sum_{V-V'} f(X_i) \le \sum_{U-U'}(m - f(X_i)).) \\
&= \alpha_1(U,V).
\end{aligned}$$

Therefore, $\alpha_1(U',V') \le \alpha_1(U,V)$, and U' does not contain any unnecessary switches. ■

LEMMA 5 *If $|B_X| = 0$, then $|U_X| = |V_X| = 0$, and $|D_Y| \le 2$.*

PROOF. By Lemma 3 and Lemma 4, $|V_X| \le |U_X| = 0$. By Lemma 2, $|D_Y| \le 2$. ■

LEMMA 6 *If $|B_X| = 1$, then $|V_X| \le |U_X| \le 1$ and $|D_Y| \le 2$. Furthermore, if $D_Y = \{f(Y_1), f(Y_2)\}$, then $\alpha^* < f(X_1) + f(X_2)$ and one of $f(X_1)$ and $f(X_2)$ is the largest bad choice in f_X.*

PROOF. Assume $\alpha^* \ge f(X_1) + f(X_2)$, then $f(X_1) + f(Y_1) > \alpha^* \ge f(X_1) + f(X_2)$. So $f(Y_1) > f(X_2)$. On the other hand, $f(X_2) + f(Y_2) > \alpha^* \ge \alpha_2^* \ge f(Y_1) + f(Y_2)$. So $f(X_2) > f(Y_1)$. A contradiction!

We notice that when $\alpha^* < f(X_1) + f(X_2)$, $f(X_1)$ and $f(X_2)$ are in the different columns, and one of them, say, $f(X_1)$, has to be the largest bad choice in f_X. ■

LEMMA 7 *If $|B_X| = 2$, then $|D_X| \le 1$ and $|D_Y| \le 1$. Furthermore, if $D_X = \{f(X_1)\}$, then $D_Y = \{f(Y_1)\}$; if $D_Y = \{f(Y_1)\}$ and $f(X_1) \notin B_X$, then $f(X_1)$ is in the right column of the assignment diagram of f_X.*

PROOF. Suppose that $f(X_i)$ and $f(X_j)$ are two bad choices in f_X. First, we notice that if $f(X_l)$ ($l = i$ or j) is a disastrous choice, $f(Y_l)$ is also a disastrous choice, because $f(X_l) + f(Y_l) > \alpha^* \geq \alpha_1^* \geq f(X_i) + f(X_j)$, and therefore $f(Y_l) > \frac{m}{2} \geq \frac{n}{2}$.

Assume that there are at least two disastrous choices in f_X. They must be $f(X_i)$ and $f(X_j)$. Since both $f(Y_i)$ and $f(Y_j)$ are disastrous choices, they are in the same column in assignment diagram of f_Y. Then $f(X_i) + f(Y_i) > \alpha^* \geq \alpha_2^* \geq f(Y_i) + f(Y_j)$. So $f(X_i) > f(Y_j)$. On the other hand, $f(X_j) + f(Y_j) > \alpha^* \geq \alpha_1^* \geq f(X_i) + f(X_j)$. So $f(Y_j) > f(X_i)$. A contradiction!

Assume that there are at least two disastrous choices in f_Y, say, $f(Y_1)$ and $f(Y_2)$. Then $f(X_1) + f(Y_1) > \alpha^* \geq \alpha_2^* \geq f(Y_1) + f(Y_2)$. So $f(X_1) > f(Y_2)$. On the other hand, $f(X_2) + f(Y_2) > \alpha^* \geq \alpha_1^* \geq f(X_i) + f(X_j) \geq f(X_1) + f(X_2)$. So $f(Y_2) > f(X_1)$. A contradiction!

If $D_X = \{f(X_1)\}$, then $D_Y = \{f(Y_1)\}$. If $D_Y = \{f(Y_1)\}$ and $f(X_1) \notin B_X$, then $f(X_1)$ must be in the right column. Otherwise, $f(X_1) + f(Y_1) > f(X_i) + f(X_j) + f(X_1)$. So $f(Y_1) > f(X_i) + f(X_j) > m$, which is impossible. ∎

References

[1] E. D. Dahl. Mapping and Compiled Communication on the Connection Machine System, *Proceedings of the* 5^{th} *Distributed Memory Computing Conference*, Charleston, SC, 1990.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[3] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time, *J. Assoc. Comput. Mach.*, vol. 23, 1976, pp. 665-679.

[4] L. Johnson and C. Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes, *Yale Computer Science Technical Report TR-500*, 1986.

[5] E. L. Lawler, J. K. Lenstra. A. H. G. Rinnooy Kan and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity, in *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan and P. Zipkin, ed., North-Holland, 1990.

[6] G. F. Lev, N. Pippenger, and L. G. Valiant. A Fast Parallel Algorithm for Routing in Permutation Networks, *IEEE Transactions On Computers*, C-30(2), February 1981.

[7] D. Reed, L. Adams, and M. Patrick. Stencils and problem partitionings: their influence on the performance of multiple processor systems, *IEEE Trans. on Computers*, C-36(7):845-858, July 1987.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE February 1993	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE ISOMORPHIC ROUTING ON A TOROIDAL MESH	5. FUNDING NUMBERS C NAS1-18605 C NAS1-19480 WU 505-90-52-01
6. AUTHOR(S) Weizhen Mao David M. Nicol	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23681-0001	8. PERFORMING ORGANIZATION REPORT NUMBER ICASE Report No. 93-5
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23681-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-191430 ICASE Report No. 93-5

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Michael F. Card
Final Report

Submitted to ORSA Journal on Computing

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 61	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We study a routing problem that arises on SIMD parallel architectures whose communication network forms a toroidal mesh. We assume there exists a set of k message descriptors $\{(x_i, y_i)\}$, where (x_i, y_i) indicates that the i^{th} message's recipient is offset from its sender by x_i hops in one mesh dimension, and y_i hops in the other. Every processor has k messages to send, and all processors use the same set of message routing descriptors. The SIMD constraint implies that at any routing step, every processor is actively routing messages with the same descriptors as any other processor. We call this Isomorphic Routing. Our objective is to find the isomorphic routing schedule with least makespan. We consider a number of variations on the problem, yielding complexity results from O(k) to NP-complete. Most of our results follow after we transform the problem into a scheduling problem, where it is related to other well-known scheduling problems.

14. SUBJECT TERMS message routing; network; scheduling; complexity; algorithms	15. NUMBER OF PAGES 24
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT