August 1988 Report No. STAN-CS-88-1218

Square Meshes are not always Optimal

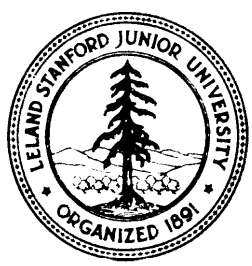
by

Amotz Bar-Noy and David Peleg

Department of Computer Science

Stanford University

Stanford, California 94305



Square Meshes are not always Optimal

Amotz Bar-Noy * David Peleg †

August 9, 1988

Abstract

In this paper we consider mesh connected computers with multiple buses, providing broadcast facilities along rows and columns. A tight bound of $\Theta(n^{\frac{1}{6}})$ is established for the number of rounds required for semigroup computations on n values distributed on a 2-dimensional rectangular mesh of size n with a bus on every row and column. The upper bound is obtained for a *skewed* rectangular mesh of dimensions $n^{3/8}$ x $n^{5/8}$. This result is to be contrasted with the tight bound of $\Theta(n^{\frac{1}{6}})$ for the same problem on the *square* ($n^{1/2}$ *x* $n^{1/2}$) mesh [PR]. This implies that in the presence of multiple buses, a skewed configuration may perform better than a square configuration for certain computational tasks. Our result can be extended to the d-dimensional mesh, giving a lower bound of $\Omega(n^{\frac{1}{d2^d}})$ and an upper bound of $O(d2^{d+1}n^{\frac{1}{d2^d}})$.

*Stanford University. Supported in part by a Weizmann fellowship and by contract ONR N00014-88-K-0166.

†Stanford University. Supported in part by contract ONR N00014-88-K-0166 and a grant of Stanford Center for Integrated Systems.

1 Introduction

The mesh organization is considered an attractive and practical architecture for parallel processing. The main desirable features of this organization are threefold: it has a simple, modular interconnection pattern, which makes it easy to construct and program; it naturally corresponds to the data format of many useful problems in matrix computations and image processing; and it is amenable to VLSI implementation [D, KLW, Kr, Re, TK, U]. A basic example of this architecture is an arrangement of the processors on integral points on the plane in a rectangular form where each processor is connected by a bidirectional communication link to its immediate neighbors on the vertical and horizontal axis. Information passes through these links in unit time. Typical tasks assigned to a computer based on the mesh architecture (a *mesh-connected* computer) involve an assignment of data items to each of the processors in the mesh and a global computational requirement involving all of the data stored at the processors. This computational requirement may entail the need to sort the elements, find certain order-statistics on them (such as their maximum etc.) or compute basic functions such as partial sums and products. Typical applications are presented in, e.g., [C, CDL, G, Ko]

The main drawback of the mesh architecture is its large diameter. Since information flow is one of the major factors affecting processing time on a parallel machine, a large diameter implies long delays even when relatively low traffic loads are required, since certain data items may need to be moved over long distances. For instance, in a square mesh of size n as described above, a data item may travel a distance of $O(\sqrt{n})$ in the worst case. This implies long processing time for various basic computational tasks.

A possible approach for overcoming the problem of long-distance data movements is to design a parallel machine based on the mesh configuration and extend it with a *broadcast* mechanism that will enable fast data transfers. Such a mechanism can be implemented using a *bus*, or a collection of buses. This approach was proposed in [B, G, JS, S1], which consider the addition of a single global bus to the mesh. It is assumed that the mesh operates synchronously using a central clock. At the beginning of each time step a processor may send a message along any or all of its links, and also send a broadcast message on the global bus. Processors receive all messages sent to them within the same time unit, and may perform some internal computation. We assume that at most one message can be broadcast on the bus at any given time. While the assumption of immediate broadcast is unrealistic since it assumes that the propagation time of messages on the bus is independent of the size

of the network, for practical situations the difference may be justifiably ignored.

While a global bus enables us to overcome sporadic instances in which a long-distance data movement is required, it does not solve all data flow problems. In particular, when "many,, data items need to be transferred over long distances, using the single bus will create a bottleneck and result in increasing the processing time. In view of this observation it was proposed in [PR, Ra, S2] to augment the mesh computer by adding multiple buses. In particular, it was suggested to include a bus for each row and column of the mesh. In a mesh with multiple buses, a processor may locally communicate with its four neighbors or broadcast a message on the bus connecting its row or column. Again we make the assumption that such a broadcast takes unit time and that at most one message may be broadcast at any given time.

We may consider the addition of multiple buses to d-dimensional meshes for any $d \geq 1$. In such a mesh each processor has $2d$ links connecting it to its $2d$ immediate neighbors. (A processor may have fewer than $2d$ neighbors if it is located on the "edges" of the mesh.) In addition, each processor belongs to d buses, one for each dimension.

Virtually all of the papers cited above assume a square configuration for the mesh. That is, a mesh of n processors is assumed to have dimensions $n^{1/2} \times n^{1/2}$. This assumption, (or rather, "design decision,") is fully justified for meshes without buses. This is because for such meshes the diameter is minimized by choosing the square design. However, when multiple buses are added to the mesh, this consideration becomes less important. At first glance, one may argue that since the architecture remains symmetric with respect to its two dimensions, a square configuration should still be preferable as far as time complexity goes. The results described in this paper indicate that this is not the case. In fact, it turns out that in the presence of multiple buses, a skewed rectangular configuration may perform better than a square configuration for certain computational tasks.

We concentrate on the problem of semigroup computations, which is an important representative for the types of problems suited for a mesh, and was considered in several of the papers mentioned above. Assume that each processor p has a value $a(p)$ taken from an infinite domain D. An associative binary operation "+" is defined on D .(for simplicity of terminology we refer to "+" as addition). The task is to compute the sum $A = \sum a(p)$, where the summation is over all the processors in the mesh. Examples of such functions are addition, multiplication and maximum.

Semigroup computations were analyzed for meshes with a single global bus and multiple buses. Bokhari [B] gives an $O(n^{1/3} \log n)$ time algorithm for computing max-

imum on a 2-dimensional mesh with a single global bus. This result was extended to higher dimensions and shown to be optimal by Aggarwal and Stout [A, S1]. They established that for the d-dimensional mesh with a single global bus, semigroup computations require $\Theta(n^{\frac{1}{d+1}})$ time.

As for square 2-dimensional meshes with multiple buses, Prasanna Kumar and Raghavendra [PR] give a tight bound of $\Theta(n^{\frac{1}{6}})$ for the problem.

Our main result is that for semigroup computations the square design is not optimal. We give a tight bound of $\Theta(n^{\frac{1}{8}})$ on the number of rounds needed to compute an n-valued semigroup function on a 2-dimensional rectangular mesh with row- and column-buses. The upper bound is obtained for a *skewed* mesh of dimensions $n^{3/8}$ x $n^{5/8}$. We also generalize our result to meshes of any number of dimensions $d \geq 1$. For d-dimensional meshes (with buses along each dimension) we present a lower bound of $\Omega(n^{\frac{1}{d2^d}})$ and an upper bound of $O(d2^{d+1}n^{\frac{1}{d2^d}})$ on the time complexity of semigroup computations. These bounds are tight for fixed d with n tending to infinity. The dimensions $n = r_1$ x . . . x r_d for which the upper bound is obtained are defined as follows. Let $r = n^{\frac{1}{d2^d}}$ (for simplicity assume that r is an integer). For every i $(1 \leq i \leq d)$ let $s_i = 2^{i-1}d + 1$ and define $r_i = r^{s_i}$.

The results for $d > 3$ are merely of theoretical interest, since from a practical point of view only 2 and 3-dimensional meshes will conceivably become feasible in future technologies. Nonetheless, we feel that the observation conveyed by our bounds is of general interest in its own right.

The rest of the paper is organized as follows. Section 2 presents some notation and definitions needed for our algorithms. The algorithm for the 2-dimensional mesh and d-dimensional mesh are presented in Section 3 and 4, respectively. In Section 5 we present the lower bound for the d-dimensional mesh for every $d \geq 2$. Throughout the rest of this paper we refer to the architecture of mesh with multiple buses simply as a *mesh,* and say *basic mesh* when referring to *a* mesh *without* buses.

2 Preliminaries

The 2-dimensional mesh is a rectangular array of processors of dimensions x x y, where $n = xy$ is the number of processors on the mesh. Denote the processors by p_{ij} for all $0 \leq i \leq y - 1$ and $0 \leq j \leq$ x $-$ 1, and denote their values by a_{ij}. The rows and the columns of the mesh are denoted by $R_0, \ldots, R_{y-1}$ and $C_0, \ldots, C_{x-1}$ respectively.

For every i and j where $0 < i < y -$ 1 and **0** < $j <$ x $-$ 1, the processor p_{ij}

is connected by communication links to its four neighbors $p_{(i-1)j}$, $p_{(i+1)j}$, $p_{i(j-1)}$ and $p_{i(j+1)}$. These links enable direct message transmissions between neighbors. Processors p_{00}, $p_{0(y-1)}$, $p_{(x-1)0}$, and $p_{(x-1)(y-1)}$ have two neighbors and the other-processors on the buses C_0, C_{x-1}, R_0 and R_{y-1} have three neighbors. (All of our results hold for meshes with wrap-around, i.e., in which the processors in column C_0 and row R_0 are connected to their corresponding processors in column C_{y-1} and row R_{x-1} respectively.) Where no confusion arises, we use R_i and C_j to denote either the set of processors they contain or the names of the appropriate row-buses and column-buses that pass through them.

For the d-dimensional mesh we need more definitions. Let $n = r_1 \times r_2 \times \cdots \times r_d$ be the size of the d-dimensional mesh, where $1 \leq r_1 \leq r_2 \leq \ldots \leq r_d$. For simplicity we select all the r_i's to be of the form r^{s_i} for some parameter r, and therefore n is also a power of r.

For every nonnegative integer x define $Z_x = \{0, \ldots, x-1\}$.

A processor in the d-dimensional mesh is represented by a d-vector $\langle c_1, c_2, \ldots, \text{cd}\rangle$, where $c_i \in Z_{r_i}$ for $1 \leq \text{i} \leq d$. Its input value is denoted by a($\langle c_1, c_2, \ldots, \text{cd}\rangle$). The basic mesh connections are as follows. For every i ($1 \leq \text{i} \leq d$) if c; $< r_i - 1$ (respectively, $0 < c_i$) then processor $\langle c_1, \ldots, \text{c;}, \ldots, \text{cd}\rangle$ is connected by a link to processor $\langle c_1, \ldots, c_i+1, \ldots, c_d\rangle$ (respectively, $\langle c_1, \ldots, c_i-1, \ldots, \text{cd}\rangle$).

Given subsets $A_i \subseteq Z_{r_i}$ for every $1 \leq \text{i} \leq d$, denote by *(Al,. . . , Ad)* the set of processors $\{ \langle x_1, \ldots, x_d\rangle \mid x_i \in A;, \ 1 \leq \text{i} \leq \text{d}\}$. When *A;* is a singleton {a} we sometimes replace it by its member, a, for clarity.

A bus is a l-dimensional submesh of the mesh. Every bus is defined by a dimension i, $1 \leq i \leq d$, and $d-1$ constants $c_j \in Z_{r_j}$ for $1 \leq j \leq d$, $j \neq$ i. Such a bus connects the processors of the set $\langle c_1, \ldots, c_{i-1}, Z_{r_i}, c_{i+1}, \ldots_d\rangle$. The set $\mathcal{B}_i$ is the set of all buses defined by the i'th dimension.

3 The algorithm for the 2-dimensional mesh

3.1 Outline

In this section we present our algorithm for the 2-dimensional mesh. We set a global parameter $r = n^{\frac{1}{8}}$ (for simplicity we assume that r is an integer) and select the dimensions of the mesh to be x $= r^5$ and $y = r^3$. During the execution of the algorithm the values get grouped and summed together into some specially designated

processors, called the *active processors,* and the values they hold are called *active values.* The algorithm is defined in such a way that in any given stage, each input value "occurs" in exactly one currently active value, so the sum of all the active values gives the correct result. At the beginning all the processors are active and at the end only processor p_{00} is active.

The algorithm is composed of eight stages, some of which are split into two substages. Each stage reduces the number of active processors by a factor of r. This is done by partitioning the active values into disjoint sets of cardinality r, and summing each into one active value. Each substage takes at most r rounds, and is performed in its entirety using either the links or the buses, but not both. In case the summation is done by the links, the r active values of each set must be at distance at most r from the processor to which they need to be summed. In case the summation is done by the buses, the r values of each set must be located on the same bus and must be the only active values on this bus. To obtain these requirements for links or buses the algorithm uses distribution operations on the active values, which take at most r rounds. Again, if the distribution is done by links then every active value cannot be sent to distances greater then r, while if the distribution is done by buses then each bus used for this operation contains no more then r active values.

3.2 The basic procedures

We now describe four basic procedures on meshes with buses, performing the four operations discussed above. All four procedures use the global parameter r, which equals $n^{\frac{1}{8}}$ in the 2-dimensional case.

Procedure SUMLINK(B)

Input: The parameter B is a bus containing the processors $q_0, \ldots, q_{k-1}$, $k = \ell r$. It is assumed that all of the processors are active, and they hold the active values $a_0, \ldots, a_{k-1}$ respectively.

We think of the bus as partitioned into consecutive segments of length r, with the j'th segment consisting of $q_{jr}, \ldots, q_{jr+r-1}$. For every $j \in Z_\ell$, the procedure sums the values of the j'th segment and stores the result, $\sum_{i=0}^{r-1} a_{jr+i}$, into q_{jr}. This operation is performed using the links only, by sequentially accumulating the values along the segment, starting from q_{jr+r-1} and going towards q_{jr}, and requires $r - 1$ rounds. (See Figure la. Boldface dots represent active processors.)

Output: There are ℓ active values on B, stored at the active processors q_{jr}, $j \in Z_\ell$.

Procedure SUMBUS(B)

Input: The parameter B is a bus containing the processors $q_0, \ldots, q_{k-1}$, of which exactly r processors $q_{i_0}, \ldots, q_{i_{r-1}}$ are active, and hold the active values $a_{i_0}, \ldots, a_{i_{r-1}}$ respectively.

This procedure sums all r active values in r rounds using only the bus. In the j'th round, $1 \leq j \leq r$, processor q_{i_j} broadcasts the value a_{i_j} on the bus B.

Output: Processor q_0 is designated as the only active processor on B, setting its active value to be $\sum_{j=0}^{r-1} a_{i_j}$. (See Figure lb.) (Note that in fact, all processors on B know this active value.)

Procedure DISTBUS(B)

Input:

(1) The parameter B is a bus containing the processors $q_0, \ldots, q_{k-1}$. On B there are exactly $m = \ell r$ active processors $q_{i_0}, \ldots, q_{i_{m-1}}$ that hold the active values $a_{i_0}, \ldots, a_{i_{m-1}}$ respectively.

(2) If $B = R_i$ (respectively, $B = C_j$) then define $B_0, \ldots, B_{\ell-1}$ to be the ℓ buses $R_i, \ldots, R_{i+\ell-1}$ (respectively, $C_j, \ldots, C_{j+\ell-1}$). The processors on the bus B_i are denoted by $q_0^i, \ldots, q_{k-1}^i$. The bus B is the only one among $B_0, \ldots, B_{\ell-1}$ that has active values.

This procedure distributes the m active values among the buses $B_0, \ldots, B_{\ell-1}$ such that each bus will contain exactly r active values. In case $B = R_i$ (respectively, $B = C_j$) then the distribution is made by the buses $C_{i_0}, \ldots, C_{i_{m-1}}$ (respectively, $R_{i_0}, \ldots, R_{i_{m-1}}$).

Output: The value a_{i_j} is held by processor $q_{i_j}^{[\frac{j}{r}]}$ which belongs to the bus $B_{[\frac{j}{r}]}$. *(See* Figure lc.)

Since this procedure is never used concurrently for parallel buses it follows that each bus distributes at most one value. Hence this procedure requires only one round.

Procedure DISTLINK(*B)*

This procedure is essentially the same as **DISTBUS(B).** The only difference is that the distribution is carried out using the links rather than the buses. Since

it takes $\ell - 1$ rounds for active values to reach $B_{\ell-1}$, it follows that this procedure requires $\ell - 1$ rounds. (See Figure 1d.)

Note that one can reduce the number of rounds required by procedures **SUM-LINK** and **DISTLINK** by a factor of roughly *2,* i.e., it is possible to sum r values (respectively, distribute ℓ values) in about $\frac{r}{2}$ (resp., $\frac{\ell}{2}$) rounds. However, for clearer description of the algorithm we prefer the above formulation.

3.3 The algorithm

Before describing the algorithm for the 2-dimensional mesh we demonstrate the usage of these procedures for the l-dimensional mesh. This mesh is equipped with a single bus denoted *B,* and the procedures are defined setting $r = n^{\frac{1}{2}}$.

Algorithm 1-DIM

1. SUMLINK(*B);*

2. SUMBUS(*B);*

It is easy to verify that algorithm **1-DIM** is correct and requires $O(n^{\frac{1}{2}})$ rounds which is optimal by [S1].

We now present the algorithm for the 2-dimensional mesh. Recall that for two dimensions we have $r = n^{\frac{1}{8}}$, x = r^5 and $y = r^3$. The algorithm is composed of a sequence of twelve substages, each involving the parallel execution of one of the above procedures on several buses. During the execution of the algorithm the set *ACTIVE* is the set of all active processors (i, j) (recall that this pair represents the processor p_{ij}). In order to clarify the flow of the algorithm we specify, for each of the stages, the set of active processors after executing that stage and its cardinality *#A.* In particular, at the beginning of the run *ACTIVE* contains all the possible pairs and *#A* = n, and at the end of the algorithm *ACTIVE* contains only the pair $\langle 0, 0 \rangle$ and *#A* = 1. Figure 2 depicts the flow of the algorithm for a 32 *x* 8 mesh (r = *2).*

Algorithm 2-DIM

Stage		ACTIVE	#A
0.		$\langle Z_{r^5}, Z_{r^3} \rangle$	r^8
1.	**for** $i \in Z_{r^3}$ **do SUMLINK**(R_i);	$\langle Z^r_{r^5}, Z_{r^3} \rangle$	r^7
2.	**for** $j \in Z^r_{r^5}$ **do SUMLINK**(C_j);	$\langle Z^r_{r^5}, Z^r_{r^3} \rangle$	r^6
3.1.	**for j** $\in Z^r_{r^5}$ **do DISTLINK**(C_j);	$\{\langle j, (j \bmod r)r^2 + ir \rangle \mid i \in Z_r, j \in Z_{r^5}\}$	r^6
3.2.	**for j** $\in Z_{r^5}$ **do SUMBUS**(C_j);	$\langle Z_{r^5}, 0 \rangle$	r^5
4.	**SUMLINK**(R_0);	$\langle Z^r_{r^5}, 0 \rangle$	r^4
5.1.	**DISTBUS**(R_0);	$\{\langle ir^2 + jr, i \rangle \mid i \in Z_{r^3}, j \in Z_r\}$	r^4
5.2.	**for** $i \in Z_{r^3}$ **do SUMBUS**(R_i);	$\langle 0, Z_{r^3} \rangle$	r^3
6.1.	**DISTBUS(** C_0);	$\{\langle j, jr + i \rangle \mid i \in Z_r, j \in Z_{r^2}\}$	r^3
6.2.	**for j** $\in Z_{r^2}$ **do SUMBUS**(C_j);	$\langle Z_{r^2}, 0 \rangle$	r^2
7.1.	**DISTBUS(** R_0);	$\{\langle ir + j, i \rangle \mid i, j \in Z_r\}$	r^2
7.2.	**for** $i \in Z_r$ **do SUMBUS**(R_i);	$\langle 0, Z_r \rangle$	r
8.	**SUMBUS**(C_0);	$\langle 0, 0 \rangle$	1

Observe that we can omit stages **6.1** and **7.1,** since after summing on a bus all the processors on the bus know the result, including, in particular, the processor designated as active after these stages. Straightforward counting reveals that the number of rounds required by Algorithm **2-DIM** is $9r - 1$ (or $9r - 3$ if stages 6.1 and **7.1** are omitted), which is $O(n^{\frac{1}{8}})$. In Section 5 we give a matching lower bound.

It remains to prove correctness. Specifically, we need to show that at the end of the run the only active processor is p_{00} and its value, a_{00}, is indeed the desired value $\sum_{i,j} a_{ij}$. This requires us to prove the following properties for each of the stages:

1. The distribution of active values on the mesh at the beginning of the stage is compatible with the requirements of the procedure applied in this stage.

2. Whenever a procedure is activated in parallel on several buses, these activations do not interfere with each other (i.e., each processor participates in at most one activation of the procedure).

3. The set of active processors in the end of each stage is as specified in the above table.

All of these properties follow in a straightforward way from the definitions of the procedures and are left for the reader to verify.

. 4 The algorithm for the d-dimensional mesh

4.1 Outline

In this section we present Algorithm **d-DIM** for the d-dimensional mesh for arbitrary $d \geq 2$. This algorithm is a generalization of Algorithm **2-DIM** of the previous section.

First let us define the dimensions $r_1, \ldots, r_d$ of the mesh. Define $\mathbf{r} = n^{\frac{1}{d2^d}}$ (again for simplicity assume that r is an integer). For every i $(1 \leq i \leq d)$ let $s_i = 2^{i-1}d + 1$ and define $r_i = r^{s_i}$. Note that $\sum_{i=1}^{d} s_i = d2^d$, so the mesh is of size $\mathbf{n}$.

As in Algorithm **2-DIM** some of the processors are active in the sense that only their values need to be summed. In each stage of the algorithm the number of active processors is reduced by a factor of r^s for some integer s. Each such stage requires at most $\mathbf{dsr}$ rounds, and makes use of one of three operators $\mathbf{SUM}_i$, $i = 1, 2, 3$, defined in the next section.

In order to describe our later constructions it is convenient to define some special submeshes. For every i $(1 \leq i \leq d)$ and for every j $(1 \leq \mathrm{j} \leq i)$ define the following sets of processors:

1. $V_{j,i} = \langle Z_{r_1}, \ldots, Z_{r_{j-1}}, 0, \ldots, 0, Z_{r_{i+1}} \ldots, Z_{r_d} \rangle$.

   Thus $V_{j,i}$ is the submesh obtained by restricting the dimensions j through i

(j $\leq i$) to the point 0 and taking all points on all other dimensions. In particular, $V_{1,d} = \{\langle 0, \ldots, 0\rangle\}$

2. $W_i = V_{1,i-1} = \langle 0, \ldots, 0, Z_{r_i}, \ldots, Z_{r_d}\rangle$.

3. $U_i = \langle 0, \ldots, 0, Z^r_{r_i}, \ldots, Z^r_{r_d}\rangle$.
Thus U_i is a "sparse" submesh of W_i containing every r'th point in dimensions i through d. There is an implicit correspondence between each point in U_i and the $r \times \ldots x\ r$ "subcube" it belongs to, and we refer to this point as "representing" its subcube.

Note that the set W_1 is the set of all processors. Also observe that all the buses in $\mathcal{B}_i$ intersect the set $V_{i,i}$ in exactly one processor, and the set $\mathcal{B}_i$ is exactly the set of all buses that are not contained in the set $V_{i,i}$.

4.2 The SUM$_i$ operations

The algorithm uses three operators of the form X = **SUM**$_i$(Y), for i = 1, 2, 3. The sets X and Y are the sets of all active processors before and after the operator is applied, respectively, and are, generally, submeshes in one of the forms W_i, U_i or $V_{j,i}$. Let us now describe how the operators **SUM**$_i$ work.

1) U_i = **SUM**$_1$(W_i)

This operator sums the values in every r x . . . $x\ r$ subcube (on the dimensions i through *d)* of the submesh W_i into the point representing it in the sparse submesh U_i. More formally, the processor (0,. . . , 0, x;, . . . , x_d⟩ where $x_j \in Z^r_j$ for $i \leq j \leq d$, receives as its new active value the sum

$$\sum_{0 \leq y_i, \ldots, y_d \leq r-1} a(\langle 0, \ldots, 0, x_i + y_i, \ldots, x_d + y_d\rangle).$$

The summation is performed using only the links, by d − i + 1 applications of the procedure **SUMLINK**, starting with the i'th dimension and ending with the d'th dimension. More precisely, the following code is executed.

for j = **i to** d **do**
  for every bus B in $\mathcal{B}_j$ containing active values **do**
    SUMLINK(B)

The operator requires $(d - i + 1)(r - 1)$ rounds and the number of active values is reduced by a factor of r^{d+1-i}.

2) $V_{i,i} = \mathbf{SUM}_2(U_i)$

The summation is done in two phases. In the first phase the active values are distributed in a way that on each bus in the set $\mathcal{B}_i$ there are exactly r active values. The second phase involves applying procedure **SUMBUS** on the buses of $\mathcal{B}_i$.

For the distribution phase we need a generalized version of the procedures **DISTBUS** and **DISTLINK.** In the 2-dimensional case all the buses perpendicular to the **given** bus B can distribute its active values. In the d-dimensional case the procedures must get an additional parameter j indicating the dimension of the distribution. Thus the distribution is done by applying the generalized procedure **DISTBUS**(B, **j)** in dimensions j = 1, . . . , i-l and then applying the generalized procedure **DISTLINK**(B, j) in dimensions j = $i + 1, \ldots, d - 1$. All the distributions are done on the buses of $\mathcal{B}_d$ that have active values. We omit the exact description of the generalized procedures, which is straightforward, but present the description of the operator $\mathbf{SUM}_2$.

for $j = 1$ **to** $i - 1$ **do**
  for every bus $B, B \in \mathcal{B}_d$ and B has active values **do**
    DISTBUS(*B, j)*
for $j = i + 1$ **to** $d - 1$ **do**
  for every bus *B*, $B \in \mathcal{B}_d$ and *B* has active values **do**
    DISTLINK(*B, j)*
for every bus *B*, $B \in \mathcal{B}_i$ **do**
  SUMBUS(*B)*

The distribution on the buses takes $i - 1$ rounds, the distribution on the links takes $(d - i - 2)(r - 1)$ rounds and the summation on the buses of $\mathcal{B}_i$ takes r rounds. Altogether, the operator requires $(d - i - 1)r$ + (2i $-$ d + 1) rounds. The number of active values is reduced by a factor of r.

3) $V_{j,i} = \mathbf{SUM}_3(V_{j+1,i})$

The operator consists of s_j phases, each reducing the number of active values by a factor of *r*. After an odd phase, ℓ, the active processors are

$$\langle Z_{r_1}, \ldots, Z_{r_{j-1}}, 0, Z_{r^{s_j-\ell}}, 0, \ldots, 0, Z_{r_{i+1}} \ldots, Z_{r_d} \rangle .$$

After an even phase, ℓ, the active processors are

$$\langle Z_{r_1}, \ldots, Z_{r_{j-1}}, Z_{r^{s_j-\ell}}, \mathbf{0}, \ldots, \mathbf{0}, Z_{r_{i+1}} \ldots, Z_{r_d} \rangle .$$

Each odd (respectively, even) phase is performed by first applying the generalized procedure **DISTBUS(** B, j) on the buses of $\mathcal{B}_j$ in dimension j + 1 (respectively, on the buses of $\mathcal{B}_{j+1}$ in dimension j) and then applying procedure **SUMBUS** on the buses of $\mathcal{B}_j$ (respectively, $\mathcal{B}_{j+1}$). The exact description is as follows.

for ℓ = 1 **to** s_j **do**
if ℓ is odd **then**
  for every bus B in $\mathcal{B}_j$ containing active values **do**
    DISTBUS(B, j + 1)
  for every bus B in $\mathcal{B}_j$ containing active values **do**
    SUMBUS(*B*)
if ℓ is even **then**
  for every bus *B* in $\mathcal{B}_{j+1}$ containing active values **do**
    DISTBUS(*B, j)*
  for every bus *B* in $\mathcal{B}_{j+1}$ containing active values **do**
    SUMBUS(*B)*

As noted after the description of algorithm **2-DIM,** the distribution part is not needed. Therefore the operator **SUM3** requires $s_j r$ rounds. The number of active values is reduced by a factor of r^{s_j}.

4.3 The algorithm

In order to illustrate the usage of the operators $\mathbf{SUM}_i$ let us first provide a different, equivalent formulation of Algorithms **1-DIM** and **2-DIM,** which makes use of these operators. Recall that W_1 is always the set of all processors, and in the l-dimensional (respectively, 2-dimensional) case W_2 (resp., W_3) contains only the processor $\langle 0,0 \rangle$.

Algorithm l-DIM

1. $U_1 = \mathbf{SUM}_1(W_1)$;
2. $W_2 = V_{1,1} = \mathbf{SUM}_2(U_1)$;

Algorithm 2-DIM

1. $U_1 = \mathbf{SUM}_1(W_1)$;
2. $W_2 = V_{1,1} = \mathbf{SUM}_2(U_1)$;
3. $U_2 = \mathbf{SUM}_1(W_2)$;
4. $V_{2,2} = \mathbf{SUM}_2(U_2)$;
5. $W_3 = \mathbf{SUM}_3(V_{2,2})$;

The **d-DIM** algorithm is a generalization of the above presentation.

Algorithm d-DIM

1. $U_1 = \mathbf{SUM}_1(W_1)$;
2. $W_2 = V_{1,1} = \mathbf{SUM}_2(U_1)$;
3. **for** $i = 2$ **to** d **do**
   (a) $U_i = \mathbf{SUM}_1(W_i)$;
   (b) $V_{i,i} = \mathbf{SUM}_2(U_i)$;
   (c) for j = i − **1 down to** 1 **do**
   $V_{j,i} = \mathbf{SUM}_3(V_{j+1,i})$;
   (d) $W_{i+1} = \mathbf{SUM}_3(V_{1,i})$;

Let us calculate the number of rounds required by the algorithm. Except for Stage 3a, whenever the number of active processors is reduced by r^s for some s, the reduction takes $sr + O(d)$ rounds. Moreover, for every i ($1 \leq i \leq d$), Stage 3a requires $(d - i - 2)r$ additional rounds. Altogether, algorithm **d-DIM** requires fewer than

$$d2^{d+1}r = d2^{d+1}n^{\frac{1}{d2^d}}$$

rounds.

In order to prove the correctness of the algorithm, one needs to check that all three operators **SUM;** are correct according to their specifications; it follows immediately that the whole algorithm works properly. Correctness of the $\mathbf{SUM}_i$ operators follows from the special way we selected the sizes of the mesh in each dimension. Formal verification is tedious but straightforward, and is omitted from the paper.

5 The lower bounds

The main result of this section is a proof that every algorithm for semigroup computation on a rectangular mesh with buses takes at least $T = \Omega(n^{\frac{1}{8}})$ steps, where n is the number of processors in the mesh. The proof technique is a generalization of similar lower bounds for the l-dimensional mesh and the square 2-dimensional mesh [Sl]. At the end of this section we extend this result to d-dimensional meshes for $d \geq 2$.

The proof is based on bounding from above the maximum number of distinct input values that an active value may "cover" in each step of the algorithm. Since our semigroup functions are "globally sensitive," in the sense that any single input may be changed so as to affect the final result, we sometimes say that a processor p "knows" some subset of the inputs, meaning that it has their sum.

The basic idea is best demonstrated by reviewing the proof in [S1] for the 1-dimensional case. In this case all n processors are on the same bus. By the end of round t, for $0 \leq t \leq T$, every processor has received at most $1 + 2t$ distinct values through the links. Only one processor can use the bus at each round t, and by doing so it can tell all other processors about at most $1 + 2(t-1) = 2t - 1$ new values (unknown to them up until now). Thus at time T a processor may have received at most $2T + 1$ values through the links and $\sum_{t=1}^{T}(2t - 1)$ values through the bus. Altogether it knows at most

$$(2T + 1) + \sum_{t=1}^{T}(2t - 1) = (T + 1)^2$$

input values. This number must exceed n, hence $T = \Omega(n^{\frac{1}{2}})$.

For the 2-dimensional case assume that the mesh size is x x y where $n = sy$. Without loss of generality let $x \leq y$.

Straightforward counting reveals that by the end of round t, for $0 \leq t \leq T$, a processor has received at most $4\binom{t+1}{2} + 1$ distinct input values through the links (including its own input value). For the derivation of our first inequality we make the over-permissive assumption that every value sent on a bus arrives at *all* n processors (for "free"). Therefore, in round t a processor may receive, through the $x + y$ buses, at most $(x + y)\left(1 + 4\binom{t}{2}\right)$ distinct new values. Consequently, at the end of round T a processor may know at most

$$\xi = \left(1 + 4\binom{T+1}{2}\right) + \sum_{t=1}^{T}\left(1 + 4\binom{t}{2}\right)$$

input values, where the first term accounts for values received through the links and the second for those received through the buses. This sum, which is $O(T^3(x + y)) = O(T^3 y) = O(T^3 n/x)$, must exceed n, hence

$$T^3 = \Omega(x). \tag{1}$$

If the mesh is square, i.e., x = y = $n^{1/2}$, the last equation implies that $T = \Omega(n^{\frac{1}{6}})$. However, one can choose a small value for x and then the bound on T is not enough.

Therefore we need to derive a second inequality. For that purpose we may again make a permissive assumption, asserting that a value known to a processor is also known to all other processors on the same row (for free). This implies that we do not need the row-buses. Moreover, assume that the goal function is to sum only the input values of one column, say, C_0, so there are only y input values. Similar arguments as for the l-dimensional case show that after round t, for $0 \leq t \leq T$, at most $2t - 1$ new values can be sent on each column-bus. There are x such buses, so necessarily

$$(1 + 2T) + x\sum_{t=1}^{T}(2t - 1) \geq y,$$

which implies that

$$T^2 = \Omega\left(\frac{y}{x}\right). \tag{2}$$

Combining Equations (1) and (2) we get

$$(T^3)^2 \cdot T^2 = \Omega(x^2 \cdot \frac{y}{x}) = \Omega(n)$$

or

$$T = \Omega\left(n^{\frac{1}{8}}\right).$$

Before reading the derivation for the general case, the reader may find it instrumental to consider the 3-dimensional case. Assume that n = xyz and that $x \leq y \leq z$. By arguments similar to the 2 dimensional case we derive *three* inequalities. The first is

$$\left(1 + 6\binom{T}{3}\right) + (xy + yz + xz)\sum_{t=1}^{T}\left(1 + \binom{t+1}{3}\right) \geq xyz,$$

which implies that

$$T^4 = \Omega(x). \tag{3}$$

The second inequality is

$$\left(1 + 4\binom{T}{2}\right) + (xz + xy)\sum_{t=1}^{T}\left(+\binom{t}{2}\right) \geq yz,$$

which implies that $T^3 = \Omega\left(\frac{y}{x}\right)$. Multiplying this by equation (3) we get

$$T^7 = \Omega(y). \tag{4}$$

The third inequality is

$$\left(1 + 2\binom{T}{1}\right) + xy\sum_{t=1}^{T}\left(1 + 2\binom{t-1}{1}\right) \geq z,$$

which implies that $T^2 = \Omega\left(\frac{z}{xy}\right)$. Multiplying this by equations (3) and (4) we get

$$T^{13} = \Omega(z). \tag{5}$$

Multiplying equations (3), (4) and (5) we get that $T^{24} = O(n)$ and thus

$$T = \Omega(n^{\frac{1}{24}}).$$

We conclude this section by presenting the inequalities for any dimension $d \geq 2$. Assuming that n = $r_1 r_2 \cdot \cdot \cdot r_d$ and that $r_1 \leq r_2 \leq \cdot \cdot \cdot \leq r_d$ the following inequalities can be derived.

$$\sum_{i=1}^{d} \frac{n}{r_i} \sum_{t=1}^{T} \left(1 + 2d\binom{t+d-2}{d}\right) \geq \prod_{i=1}^{d} r_i;$$

$$\sum_{i=2}^{d} \frac{n}{r_i} \sum_{t=1}^{T} \left(1 + 2(d-1)\binom{t+d-3}{d-1}\right) \geq \prod_{i=2}^{d} r_i;$$

$$\sum_{i=j+1}^{d} \frac{n}{r_i} \sum_{t=1}^{T} \left(1 + 2(d-j)\binom{t+d-j-2}{d-j}\right) \geq \prod_{i=j+1}^{d} r_i;$$

$$(r_1 \cdots r_{d-1}) \sum_{t=1}^{T} \left(1 + 2\binom{t-1}{1}\right) \geq r_d.$$

From these inequalities we get that for every j in the range $1 \leq j \leq d$, the following holds:

$$T^{d+2-j} = \Omega\left(\frac{r_j}{r_1 \cdots r_{j-1}}\right). \tag{6}$$

When j = 1 the denominator is 1. By appropriate multiplications of equations from (6) we get that for every j in the range $1 \leq j \leq d$

$$T^{2^{j-1}d+1} = \Omega(r_j). \tag{7}$$

Multiplying all the equations in (7) we conclude that $T^{d2^d} = \Omega(n)$ and thus

$$T = \Omega(n^{\frac{1}{d2^d}}).$$

References

[A] A. Aggarwal, Optimal Bounds for Finding Maximum on Array of Processors with *k* Global Buses, *IEEE* Transactions on Computers, **c-35**, (1986), 62–64.

[B] S. H. Bokhari, Finding Maximum on an Array Processor with a Global Bus, *IEEE* Transactions on Computers, c-33, (1984), 133-139.

[C] S.N. Cole, Real-Time Computation by n-dimensional Iterative Arrays of Finite-State Machines, *IEEE* Transactions on Computers, **c-18**, (1969), 349-365.

[CDL] L.P. Cordella, M.J.B. Duff and S. Levialdi, An Analysis of Computational Cost in Image Processing: A Case Study, *IEEE* Transactions on Computers, **c-27**, (1978), 904-910.

[D] M.J.B. Duff, CLIP4: A Large Scale Integrated Circuit Array Parallel Processor, in *Proc.* 3rd *Int. Joint Conf.* Pattern Recognition, 1976, 728-732.

[G] W.M. Gentleman, Some Complexity Results for Matrix Computations on Parallel Processors, *J.* ACM, 25, (1978), 112-115.

[JS] H.F. Jordan and P.L. Sawyer, A Multimicroprocessor System for Finite Element Structural Analysis, *Comput. Struc.*, **10**, (1979), 21-29.

[KLW] W.H. Kautz, K.N. Levitt and A. Waksman, Cellular Interconnection Arrays, *IEEE* Transactions on Computers, c-17, (1968), 443–451.

[Ko] S.R. Kosaraju, Fast Parallel Processing Array Algorithms for some Graph Problems, *proc. 27th ACM Symp. on Theory* of Computing, pp. 231-236, 1979.

[Kr] B. Kruse, A Parallel Processing Machine, *IEEE* Transactions OR Computers, **c-23**, (1973), 1057–1087.

[PR] V. K. Prasanna Kumar and C. S. Raghavendra, Array Processor with Multiple Broadcasting, Journal of *Parallel* and Distributed Computing, **4**, (1987), 173-190.

[Ra] C. S. Raghavendra, Hmesh: a VLSI Architecture for Parallel Processing, *CONPAR* 86, LNCS 237, Springer-Verlag, pp. 76-83, 1986.

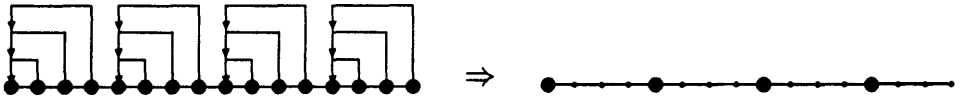
[Re] A.P. Reeves, A Systematically Designed Binary Array Processor, *IEEE Transactions on Computers*, c-29, (1980), 278-287.

[S1] Q. F. Stout, Mesh-Connected Computers with Broadcasting, IEEE Trans. *on Computers*, **c-32**, (1983), 826-830.

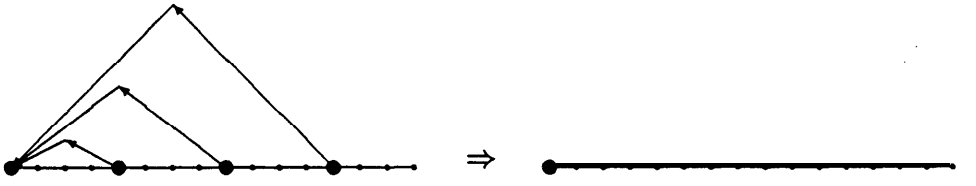
[S2] Q. F. Stout, Meshes with Multiple Buses, *proc. 27th IEEE Symp. on Foundations of Computer Science*, pp. 264-273, 1986.

[TK] C.D. Thompson and H.T. Kung, Sorting on a Mesh Connected.Processor Array, Comm. ACM, (1972), 263-271.

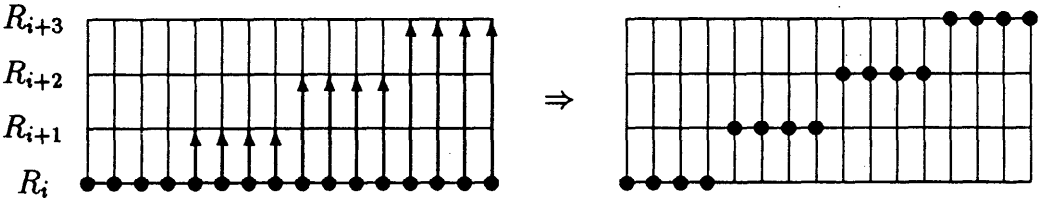
[U] S.H. Unger, A Computer Oriented Toward Spatial Problems, *proc. IRE*, pp. 1744-1750, 1958.



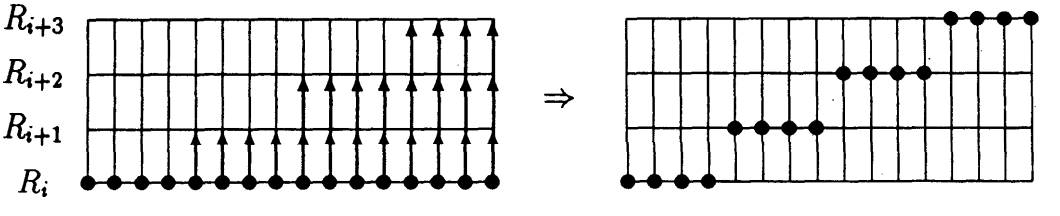
a) SUMLINK(B)



b) SUMBUS(B)

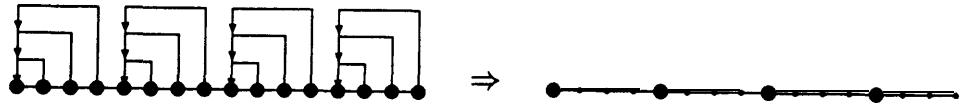


c) DISTBUS(B), $(B = R_i)$

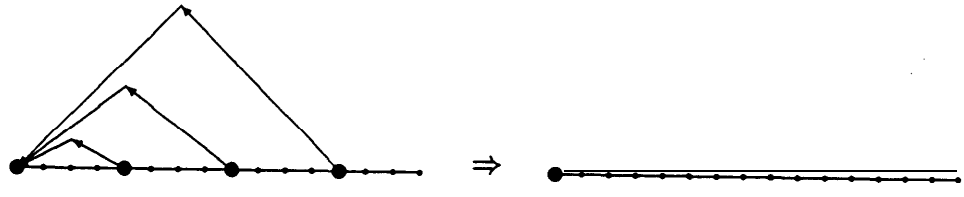


d) DISTLINK(B), $(B = R_i)$

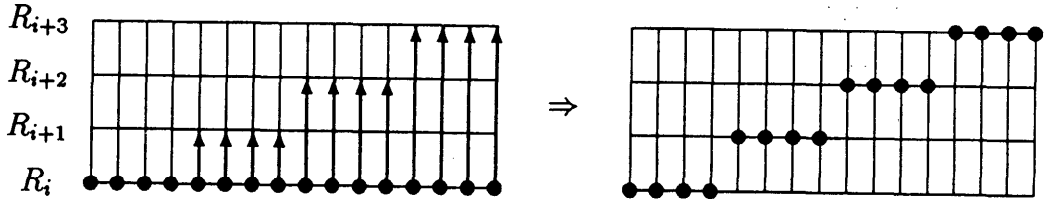
Figure 1: The four procedures, $(r = 4)$



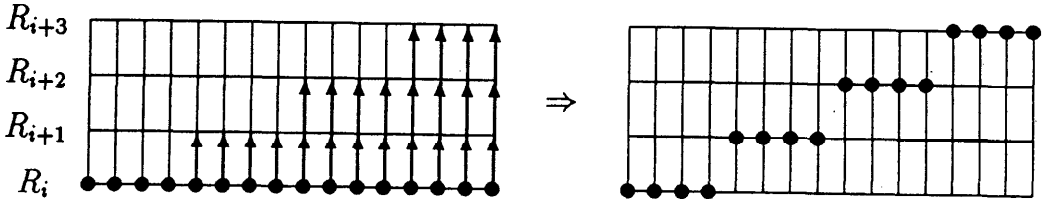
a) SUMLINK(B)



b) SUMBUS(B)



c) DISTBUS(B), $(B = R_i)$



d) DISTLINK(B), $(B = R;)$

Figure 1: The four procedures, $(r = 4)$