Upper Bounds to Processor-Time Tradeoffs under Bounded-Speed Message Propagation

Gianfranco Bilardi*
Dipartimento di Elettronica ed Informatica
Università di Padova
35131 Padova, Italy
bilardi@artemide.dei.unipd.it

Franco P. Preparata†
Department of Computer Science
Brown University
Providence, RI 02912
franco@cs.brown.edu

Abstract

Upper bounds are derived for the processor-time tradeoffs of machines such as linear arrays and two-dimensional meshes, which are compatible with the physical limitation expressed by bounded-speed propagation of messages (due to the finiteness of the speed of light). It is shown that parallelism and locality combined may yield speedups superlinear in the number of processors. The speedups are *inherent*, due to the optimality of the obtained tradeoffs as established in a companion paper.

Simulations are developed of multiprocessor machines by analogous machines with fewer processors. A crucial role is played by the hierarchical nature of the memory system. A divide-and-conquer technique for hierarchical memories is developed, based on the graph-theoretic notion of *topological separator*. For multiprocessors, this technique also requires a careful balance of memory access and interprocessor communication costs, which leads to non-intuitive orchestrations of the simulation process.

1 Introduction

Since the advent of the digital computer, its supporting technology has been characterized by steady and impressive growth. Although most parameters are still being improved, there is an emerging consensus that physical limitations to signal propagation speed and device size are becoming increasingly significant.

We have recently undertaken an analysis [BP92] of a hypothetical environment, called the "limiting technology", where – provocatively – the limits of physics are assumed to have been attained, and no further improvements are feasible. The limiting technology forces a breakdown of several traditional *instantaneous models* where signal propagation was assumed to be instantaneous, as motivated by technologies where delays were substantially dominated by device switching time.

In the framework of the limiting technology, the processor-time tradeoff can be different from the classical tradeoff embodied by Brent's Principle [B74, J92], whereby a computation running for T steps on n processors can be emulated in at most $\lceil n/p \rceil T$ steps on $p < n$ processors of the same type. A corollary of Brent's Principle is that the best parallel algorithm on p processors cannot be more than p time faster than the best sequential algorithm (the Fundamental Principle of Parallel Computation [S86]).

Informally, when communication delays are proportional to physical distances, the deployment of p processors can lead to speed-ups in two ways. A p-fold parallelism in the computation translates into a an $O(p)$ speed-up due to simultaneous execution of operations (and the corresponding data access), as in Brent's Principle. A more subtle effect is that, for a given total amount of memory, the average distance of any memory region from the closest processor decreases with the number of deployed processors. Then, data locality (operands and result of the same operation happen to be stored in physically close locations) can contribute a further speed-up.

Consider the following classical example. Two $\sqrt{n} \times \sqrt{n}$ matrices can be multiplied in $\Theta(\sqrt{n})$ steps by a $\sqrt{n} \times \sqrt{n}$ mesh of processors, each equipped with conventional arithmetic capabilities. The same computation can be carried out in $\Theta(n^{3/2})$ steps by a single processor equipped with a random access memory of size $\Theta(n)$. In the instantaneous model, a mesh step and a uniprocessor step take the same time, hence the speedup attained by the mesh is $\Theta(n)$, proportional to the number of processors. However, assuming that our

*Supported in part by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM), by the Italian Ministry of University and Research, and by Brown University.

†Supported in part by NSF Grant CCR94-00232 and ONR Contract N 00014-91-J-4052, ARPA order 2225.

SPAA'95 Santa Barbara CA USA© 1995 ACM 0-89791-717-0/95/07.$3.50

machines are laid out in a two-dimensional region of diameter $\Theta(\sqrt{n})$ and that delays for data accesses are proportional to distances, the situation changes as follows. The performance of the mesh is unaltered, since its near-neighbor connections have length independent of n. Instead, the performance of the uniprocessor deteriorates by a factor $\Theta(\sqrt{n})$, the average distance of a memory cell from the CPU. In this scenario, the speedup of the n-processor mesh is $\Theta(n^{3/2})$. The preceding estimate refers to a straightforward implementation of matrix multiplication; however, as noted in [AACS87], a careful exploitation of locality would enable to contain the access overhead in the uniprocessor to within a factor $\Theta(\log n)$. This simple example illustrates the potential for superlinear speedups under the limiting technology. The apparent paradox is easily resolved by observing that the speedup is now being referred to a uniprocessor whose basic step becomes slower with increased memory size. Nevertheless, the increased advantages of parallel over serial computation within the limiting technology represent an essential feature and deserve closer investigation.

The objective of this paper is to undertake a systematic analysis of processor-time tradeoffs in the limiting technology, identifying the separate contributions of *degree of parallelism* (number of deployed processors) and of *data locality*. With reference to a computing system where an access cost is specified for each memory location, we say that an algorithm possesses data locality if its running time depends upon the addresses at which both input and intermediate values of the computation are stored. The running time of algorithms with data locality can be minimized by a careful address management (indeed, such is the strategy that reduces the overhead from $\Theta(\sqrt{n})$ to $\Theta(\log n)$ in the matrix multiplication algorithm). Clearly, data locality plays no role in models with uniform access cost, such as the RAM and the P-RAM.

The physical computing system we shall adopt (to be formally defined in Section 2) is what appears to be the only scalable machine in the limiting technology [BP92]: the mesh, i.e., a uniform lattice (in $d \leq 3$ dimensions) of processors. In our study, a mesh node is a (CPU, hierarchical memory module) pair for which worst-case private-memory access time is of the same order as the data-exchange time with a near-neighbor unit.

Let $M_d(n, p, m)$, $d = 1, 2$, denote a d-dimensional mesh of p nodes, each of which is a CPU equipped with a hierarchical memory module of mn/p locations (see Section 2 for more details). Clearly M_1 and M_2 are the conventional linear array and two-dimensional mesh, respectively. For M_1 and M_2 we establish upper bounds to the speed-up by exhibiting suitable simulations between machines with different number of processors, but with identical total amounts of memory. The following theorem summarizes two companion results, respectively shown in Sections 4 and 5 for M_1 and M_2.

Theorem 1 *For $T_n \geq n^{1/d}$ and $d = 1, 2$, a T_n-step computation of an $M_d(n, n, m)$ can be simulated by an $M_d(n, p, m)$, $p \leq n$, with slowdown*

$$T_p/T_n = O((n/p)\Lambda(n, m, p))$$

where the term Λ is defined as follows:

- *for $m \leq (n/p)^{1/2d}$,*
  $\Lambda(n, m, p) = (m/p^{1/d}) \log m + m \log(2n^{1/d}/p^{1/d}m^2)$,
- *for $(n/p)^{1/2d} \leq m \leq (np)^{1/2d}$,*
  $\Lambda(n, m, p) = (m/p) \log(n/p)^{1/2d} + 2(n/p)^{1/2d}$,
- *for $(np)^{1/2d} \leq m \leq n^{1/d}$,*
  $\Lambda(n, m, p) = (m/p^{1/d}) \log(2n^{1/d}/m) + n^{1/d}/m$,
- *for $n^{1/d} \leq m$, $\Lambda(n, m, p) = (n/p)^{1/d}$.*

In the above theorem, the factor n/p can be viewed as the *parallelism slowdown* (corresponding to Brent's Principle) while the factor Λ can be viewed as the *locality slowdown*. The four different ranges of m where Λ takes a different expressions correspond to different mechanisms becoming dominant, as will be illustrated in detail in Section 4. We observe here that when m is large (for $m \geq n^{1/d}$) the locality slowdown is $\Lambda = O((n/p)^{1/d})$. Such slowdown is easily achieved by a step-by-step simulation of the larger machine by the smaller machine. For smaller values of m ($m < n^{1/d}$), a lower slowdown is achieved by more sophisticated simulation schemes which convert the spatial locality of the computation executed by the larger machine into temporal locality for the smaller machine. In other words, the above theorem states that some - but never all - of the locality of the larger machine computation can be recovered by the smaller machine.

The value of Theorem 1, however, goes well beyond the elucidation of the performance of the simulation scheme discussed in this paper. Its significance is eminently computational as we shall now illustrate. Certainly, if the larger machine computation has low locality, the proposed simulation scheme is inappropriate (since the smaller machine would unnecessarily simulate remote accesses occurring in the larger machine computation). Conversely, if the larger machine computation has inherently higher locality than explicitly exploited (since such exploitation is not afforded by the structure of private memory modules), it is conceivable that a subtler simulation scheme could exhibit lower slowdowns than those obtained by Theorem 1. The significance of Theorem 1 is in relation to applications for which the larger machine computation exactly reflects

the locality. Such computation exist, as we have shown in a companion paper [BP95b] yielding matching lower bounds (for most values of n, p, and m). Therefore, no improvement is possible beyond the results of Theorem 1, thereby showing that locality slowdown is an *inherent* feature of limiting technology simulations.

This paper is organized as follows: In Section 2 we develop the models of machines and computations, in Section 3 we introduce a general technique, called "topological separator," capturing the dependencies within computation dags, which is then used to analyze the simulations between linear arrays (Section 4) and between meshes (Section 5).

The topological separator approach is of independent interest, since it provides a framework for efficient memory management in dag computations.

2 Model

Machines. We shall consider parallel machines built as interconnections of (processing-element, memory-module) pairs. Such a pair is modeled as a Hierarchical Random Access Machine, or H-RAM, a generalization of the RAM [CR73] introduced by [AACS87] (under the name of Hierarchical Memory Model) to capture the higher cost of remote memory access. (See also [S95] and its bibliography.)

Definition 1 *An $f(x)$-H-RAM is a random access machine where an access to address x takes time $f(x)$.*

According to this definition, a (processing-element, memory-module) pair is an $f(x)$-H-RAM. We take as a unit of time the execution time of a RAM instruction involving only the lowest address ($x = 0$), and as a unit of length the distance within which memory cells can be accessed in unit time. In $d \leq 3$ dimensions, we let m be the number of memory cells that fit in a d-dimensional cube of unit side. Then, the H-RAM will be taken to have access function $f(x) = (x/m)^{1/d}$.

Definition 2 *For $d \geq 1$, we denote by $M_d(n, p, m)$, a d-dimensional near-neighbor interconnection of p $(x/m)^{1/d}$-H-RAMs where each* H-RAM *has memory size mn/p and the geometric distance between near neighbors is $(n/p)^{1/d}$.*

Thus, n represents the d-dimensional volume of the machine, and nm its total memory size. A processor is assumed to be laid out in at most one unit of volume. Under this hypothesis, the processors' contribution to the system volume never exceeds that of the memory, and therefore will not be explicitly accounted for in what follows.

In one dimension, $M_1(n, p, m)$ is a *linear array* network whose interconnection is specified by the graph $H = (N, E)$ with nodes $N = \{0, \ldots, p-1\}$ and bidirectional links $E = \{(i, i+1) : 0 \leq i < p-1\}$.

In two dimensions, $M_2(n, p, m)$ is a *two-dimensional square mesh* whose interconnection is specified by the graph $H = (N, E)$, with nodes $N = \{(i, j) : 0 \leq i, j < \sqrt{p}\}$ and bidirectional links

$$E = \{((i, j), (i + a, j + b)) :$$

$$(a, b) \in \{(-1, 0), (0, 1), (1, 0), (0, -1)\}, (i+a, j+b) \in N\}.$$

Computations. Some of the computations studied in this paper are straight-line and can be modeled by a directed acyclic graph (dag) $G = (V, A)$ where:

- A vertex v with no incoming arc represents an input operation and has an input value associated with it;
- A vertex v with k incoming arcs is labeled with a k-argument operator. The value associated with v is the result of applying the operator to the k values associated with the origins of the incoming arcs (taken in some specific order).
- The values associated with the vertices in a designated subset are regarded as the outputs of the computation.

An interesting case of dag arises when modeling an arbitrary T-step computations of network $H = (N, E)$.

Definition 3 *Given an undirected graph $H = (N, E)$ and an integer $T \geq 1$, we let $G_T(H) = (V, A)$ denote the directed acyclic graph where:*

$$V = \{(v, t) : v \in N, 0 \leq t \leq T\},$$

$$A = \{(u, t-1), (v, t)) : u = v \text{ or } (u, v) \in E, 1 \leq t \leq T\}.$$

Informally, vertex (v, t) represents the operation performed by node v of network H at step t. The arc set A reflects the fact that the operands for vertex (v, t) are the value of a (unique) memory cell of v and the values supplied by the neighbors of v at step $t-1$. Vertex $(v, 0)$ denotes the initial value of the memory cell in $v \in V$ and is an input vertex. A simulation of H can be viewed as the execution of graph $G_T(H)$.

3 The topological separator approach

In this section we discuss general simulation schemes within homogeneous classes of parallel machines, as introduced in the preceding section, i.e., linear arrays and square meshes. Specifically, we shall consider the simulation of $M_d(n, n, m)$, the *guest*, by means of an $M_d(n, p, m)$, with $1 \leq p \leq n$, the *host*. We shall study first the case in which the host has $p = 1$, and then generalize the results to arbitrary p.

3.1 Naive simulation on H-RAMs

Let the guest be described by a graph $H = (N, E)$ where each node has m memory cells. The most straightforward simulation scheme is one where the uniprocessor host, an H-RAM with nm memory cells, mimics individual steps of the guest, that is, for each such step, it carries out successively the activity of each individual processor of the guest in that step. In such approach, each step involves n remote accesses to the private memories of the simulated node, and therefore carries a substantial latency overhead. We call such approach the *naive simulation.*

Proposition 1 *The* naive simulation *of T steps of network $H = (N, E)$ by an $f(x)$-*H-RAM *takes time $O(Tnf(nm))$. In particular, $M_d(n, 1, m)$ can simulate $M_d(n, n, m)$ with* slowdown $O(n^{(1+1/d)})$.

3.2 Divide-and-conquer on H-RAMs

Intuitively, a more efficient approach is one where blocks of data are relocated to a region of memory closer to the CPU (thereby involving a smaller access overhead), such that a substantial amount of computation - spanning several steps of the guest – can be executed on the basis of the transferred block alone. Below, we explore this approach. Our objective is to identify decompositions of the dag so that execution of the decomposition blocks will involve moderate data relocation overhead.

Assuming $m = 1$, for a computation of T steps, $G_T(H) = (V, A)$ denotes the associated dag, as defined in Section 2.

A vertex (v, t) of $G_T(H)$ can be executed only if the set $\mathrm{Pred}(v, t)$ of its immediate predecessors has been executed. Generalizing this observation, a set $U \subseteq V$ can be executed only after the execution of its *preboundary* $\Gamma_{in}(U)$, defined as

$$\Gamma_{in}(U) \triangleq \cup_{(v,t)\in U} \mathrm{Pred}(v, t) - U.$$

The following definition captures the conditions under which the execution of set U can be decomposed into the successive executions of subsets $U_1, U_2, ..., U_q$.

Definition 4 *An ordered partition $(U_1, U_2, ..., U_q)$ of $U \subseteq V$ is said to be a* topological partition *of U if, for $r = 1, 2, ..., q$,*

$$\Gamma_{in}(U_r) \subseteq \Gamma_{in}(U) \cup (\cup_{i=1}^{r-1} U_i).$$

It can be seen that a topological partition of U can be refined into a topological sorting of U, and that sets U_i's are convex, in the following sense.

Definition 5 *A set of $U \subseteq V$ is said to be* convex *if, whenever u and v are in U, so is any vertex on any path from u to v.*

As we shall see in Sections 4 and 5, topological partitions of dags are not trivial. For example, if the dag under consideration is a cubic lattice, a partition of such dag into cubes is not a topological partition.

We now formulate and analyze a simple strategy to execute the vertices of a set $U \subseteq V$ on an $f(x)$-H-RAM, based on a topological partition of U.

Proposition 2 *Let $T(W)$ and $S(W)$ respectively denote time and space required by the execution of convex set $W \subseteq V$ on an $f(x)$-*H-RAM*, with $\Gamma_{in}(W)$ initially stored in memory locations from $S(W) - |\Gamma_{in}(W)|$ to $S(W) - 1$.*

If $(U_1, U_2, ..., U_q)$ is a topological partition of U, then

$$S(U) \leq max_{i=1}^{q} S(U_i) + \varphi(U),$$

and

$$T(U) \leq \sum_{i=1}^{q} T(U_i) + 4f(S(U))\varphi(U),$$

where $\varphi(U) = \sum_{i=1}^{q} |\Gamma_{in}(U_i)|$.

Proof Assume that $\Gamma_{in}(U)$ is initially stored in memory locations from $S(U) - |\Gamma_{in}(U)|$ to $S(U) - 1$. To execute U, for $i = 1, 2, ..., q$ do:

1. Copy preboundary $\Gamma_{in}(U_i)$ into memory locations from $S(U_i) - |\Gamma_{in}(U_i)|$ to $S(U_i) - 1$. (This step takes time at most $2f(S(U))|\Gamma_{in}(U_i)|$, since each preboundary value is read from and written to a location with address lower than $S(U)$.)
2. Execute U_i, using memory locations from 0 to $S(U_i) - 1$ as working space, (in time $T(U_i)$).
3. Copy the value of the vertices in $U_i \cap \Gamma_{in}(U_{i+1} \cup ... \cup U_q)$ into suitable memory locations in the range from $S(U) - \varphi(U)$ to $S(U) - 1$. (Over all q iterations, at most $\sum_{i=1}^{q} |\Gamma_{in}(U_i)|$ values are moved (read and written) between locations with address lower than $S(U)$, taking time at most

$$2f(S(U)) \sum_{i=1}^{q} |\Gamma_{in}(U_i)|.$$

The bounds stated for $S(U)$ and $T(U)$ follow from those to the individual steps of the above procedure. □

Our aim is to identify topological partitions of a convex set U whose components, although not necessarily "geometrically similar" to U, *inherit* its decomposability properties. This feature is captured by the following definition of the key notion of topological separator.

Definition 6 *Let $0 < \delta < 1$, let $q > 1$ be an integer, and let $g(x)$ be a real function. A convex set $U \subseteq V$ has a $(g(x), \delta)$-*topological separator *if either $|U| = 1$ or*

- $|\Gamma_{in}(U)| \leq g(|U|)$;
- *U has a topological partition $(U_1, U_2, ..., U_q)$ where, for each $i = 1, 2, ..., q$, $|U_i| \leq \delta|U|$;*
- *for each $i = 1, 2, ..., q$, U_i has a $(g(x), \delta)$-topological separator.*

We now consider the worst case space $\sigma(k)$ and the worst case time $\tau(k)$ to execute a set U of size $|U| = k$ when U has a topological separator. The bounds given by Proposition 2 can be rewritten as

$$\sigma(|U|) \leq \sigma(\delta|U|) + q\ g(\delta|U|), \tag{1}$$

and

$$\tau(|U|) \leq \sum_{i=1}^{q} \tau(|U_i|) + 4q\ f(\sigma(U))\ g(\delta|U|), \tag{2}$$

where, in bounding φ, we have assumed that $g(x)$ is monotone non-decreasing. Of interest for later developments are dags with topological separators of size $O(x^{d/(d+1)})$ to be executed by an $O(x^{1/d})$-H-RAM. In such cases, the following proposition applies, with $\alpha = 1/d$ and $\gamma = d/(d+1)$ (proof omitted):

Proposition 3 *Let U have a (cx^γ, δ)-topological separator, for some constants $c > 0$, $1/2 \leq \gamma < 1$, and $0 < \delta < 1$. Then, time and space to execute U on an (ax^α)-H-RAM, with $a > 0$ and $0 < \alpha \triangleq (1-\gamma)/\gamma \leq 1$, satisfy*

$$\sigma(|U|) \leq \sigma_0 |U|^\gamma, \tag{3}$$

$$\tau(|U|) \leq \tau_0 |U| \log |U|, \tag{4}$$

where $\sigma_0 \triangleq qc\delta^\gamma/(1-\delta^\gamma)$ and $\tau_0 \triangleq 4qa\sigma_0^\alpha \delta^\gamma / \log(1/\delta)$.

In the following subsections, we shall apply the preceding techniques by exhibiting appropriate topological separators for the computation graphs of the linear array and of the mesh. This simulation approach can be extended to arbitrary values of m, although it becomes increasingly onerous so that, for large values of m, the naive simulation remains as the only alternative.

The vertex sets of dags corresponding to linear arrays or to square meshes are two- or three-dimensional lattices, respectively. We shall find it convenient to specify a convex subset X of the vertex set V by means of a semi-closed convex geometric domain D in the same space as the lattice. Specifically, D does not contain those points of its frontier corresponding to minimum values of t (for any fixed values of the other coordinates). Set X consists of all lattice points in D. Since such domain D is a very close approximation to the domain D' obtained as the union of unit cubes centered at the points of X, for the sake of simplicity we shall approximate $|X|$ with the measure (area or volume, as appropriate) of D.

We first describe the instance of a one-dimensional parallel machine (a linear array), which, while simple, contains all the essential features of the more complex two-dimensional case.

4 Simulation between linear arrays

4.1 Simulation by uniprocessors

We begin by illustrating the simpler case of $m = 1$, which encompasses the cases where the guest system is either a systolic network or a cellular automaton. The guest is an $M_1(n, n, 1)$, a T_n-step computation of which is modeled by dag $G_{T_n}(M_1(n, n, 1))$ (see Definition 3). By the topological-separator technique, we can establish the following result.

Theorem 2 *For $T_n \geq n$, a T_n-step computation of an $M_1(n, n, 1)$ can be simulated by an $M_1(n, 1, 1)$ with slowdown $T_1/T_n = O(n \log n)$.*

Proof We describe the simulation scheme for $T_n = n$. For larger values of T_n, it is sufficient to repeat the n-step simulation $\lceil T_n/n \rceil$ times. Thus, the computation to be simulated is modeled by dag $L_{n-1} \triangleq G_{n-1}(M_1(n, n, 1)) = (V, A)$ where V is specified by the rectangle $[0, n-1] \times [0, n-1]$. We shall make use of a diamond-shaped domain $D(r)$ defined as the intersection of the following four half-planes of the (x, y)-plane: $y \pm x \geq -r/2$ and $y \pm x \leq r/2$. Note that $|D(r)| = r^2/2$ and that $\Gamma_{\text{in}}(D(r)) \leq 2r = 2\sqrt{2|D(r)|}$ and that $D(r)$ has an ordered partition into four domains of type $D(r/2)$. We conclude that convex set $D(r)$ has a $(2\sqrt{2x}, 1/4)$-topological separator.

Referring to Figure 1, we also recognize that V has an ordered partition $(U_1, U_2, U_3, U_4, U_5)$ into five domains, of which U_3 is of type $D(n)$ and the others are truncated versions of $D(n)$. It is therefore sufficient to consider $D(n)$, which – as noted above – has a $(2\sqrt{2x}, 1/4)$-topological separator. We wish to execute $D(n)$ on an (x)-H-RAM. Therefore, with the terminology of Proposition 3, we have $\gamma = 1/2, c = 2\sqrt{2}, \delta = 1/4, q = 4, \alpha = 1$, whence (with $\sigma_0 = 2\sqrt{2}$ and $\tau_0 = 8\sqrt{2}$):

$$\sigma(|D(n)|) = \sigma_0 |D(n)|^{1/2} = O(n),$$

$$\tau(|D(n)|) = \tau_0 \frac{n^2}{2} \log \frac{n^2}{2} = O(n^2 \log n).$$

Since the simulation is to be repeated $\lceil T_n/n \rceil$ times, the overall simulation time is $O(T_n n \log n)$. □

In the $O(n \log n)$ slowdown, the factor n accounts for

the reduction of parallelism and the factor $\log n$ is due to the loss of locality.

We now consider $m > 1$, so that $M_1(n, n, m)$ and $M_1(n, 1, m)$ are respectively the guest and the host.

As seen in Subsection 3.1, the "naive" simulation scheme takes time $T_1 = O(T_n n^2)$. Alternative to this approach is an adaptation of the divide-and-conquer strategy of Subsection 3.2, where the access to a single variable is replaced by the access to the entire private memory of an individual processor (i.e., by the access to a block of m data items). With this approach, we obtain $T_1 = O(T_n nm \log n)$, which is smaller than the time of the naive simulation for $m < n/\log n$.

The two approaches can be combined by applying the topological-separator technique to diamonds $D(r)$ with $r > m$ and executing diamonds with $r \leq m$ by the naive method. When $n > m$, the simulation is recursive and its initial levels are data transfers designed to bring in proximity of the processors the diamonds to be directly executed by naive simulation (referred to as "executable diamonds"). In other words, the recursive simulation consists of $\log(n/m)$ levels, each of which exclusively involves data relocation. The bottom of the recursion is the execution - by naive simulation - of an executable diamond.

Since an executable diamond is of type $D(m)$, there are $\log(n/m)$ data transfer levels, where level k involves the transfer of $\Theta(nm2^k)$ data items at distance $O(n2^{-k})$, taking $O(nm2^k.n2^{-k}.\log(n/m)) = O(n^2 m \log(n/m))$ time. In the ensuing naive-simulation phase, $\Theta((n/m)^2)$ diamonds of type $D(m)$ are to be executed, each in time $O(m^3)$, taking in all $O(n^2 m)$ time. Thus, $T_1 = O(T_n nm \log(n/m))$. When $n \leq m$, only the naive simulation applies. In summary, we have:

Theorem 3 *For $T_n \geq n$, a T_n-step computation of an $M_1(n, n, m)$ can be simulated by an $M_1(n, 1, m)$ with slowdown $T_1/T_n = O(n \min(n, m\ \ell og(n/m)))$.* *

We see that the locality slowdown is $O(m \log(n/m))$ for $m < n$, and $O(n)$ otherwise.

4.2 Simulation by multiprocessors: a modified Brent principle

We now consider the simulation of guest $M_1(n, n, m)$ by host $M_1(n, p, m)$. As above, we focus on a computation of the guest running for $T_n = n$ steps.

We begin by considering a parallel version of the naive simulation, whereby processor PE_i of $M_1(n, p, m)$ (located at abscissa $\lfloor n/p \rfloor i$) performs the actions of processors PE_i, PE_{i+1},...,$\mathrm{PE}_{i+n/p-1}$ of $M_1(n, n, m)$. It is simple to see that $T_p/T_n = O((n/p)^2)$.

* Hereafter, we let $\ell og(x)$ denote $\log_2(x+2)$. Note that $\ell og(x) \geq 1$ for any nonnegative x.

For $m < n/p$, an improvement can be achieved by a straightforward generalization of the simulation by $M_1(n, 1, m)$ (Subsection 4.1). A zig-zag band of width (n/p), as shown in Figure 2, is assigned to each processor. Different bands are executed in parallel (with special handling of the vertical boundaries of L_{n-1}). Each band is decomposed into $\sim 2p$ diamond-shaped domains of type $D(n/p)$, to be executed in sequence. Execution of any such domain (in time $O((n/p)^2 \min(n/p, m\ \ell og(n/pm)))$ must be preceded by the access to the domain's preboundary (in time $O((nm/p) \cdot (n/p)) = O((n/p)^2 m)$, since $(n/p)m$ data items are to be accessed at distance n/p. This results in an overall execution time $O((n^2/p) \min(n/p, m\ \ell og(n/pm)))$. In terms of slowdown,

$$T_p/T_n = O((n/p) \min(n/p, m\ \ell og(n/pm))).$$

However, the availability of $p > 1$ processors affords a strategy infeasible with just one processor. Indeed, imagine to split vertically a diamond-shaped domain $D(s)$ along its diagonal and to store the left and right halves of its preboundary in non-contiguous regions of memory. Referring to two processors PE_i and PE_j, we have two alternative strategies for the execution of $D(s)$: (i) to let one processor, say PE_i, access the preboundary (of size sm) and alone execute $D(s)$, or (ii) to let PE_i and PE_j access the left and right halves of the preboundary, respectively, and execute the corresponding semidiamonds, while exchanging the $O(s)$ data items supposed to flow across the diagonal. We denote the latter "execution in the cooperating mode". Clearly, depending upon the relative positions of $\mathrm{PE}_i, \mathrm{PE}_j$, left preboundary, and right preboundary, one alternative may be preferable over the other.

We seek a rearrangement of data that conveniently bounds the distance at which data communication occurs during the simulation, both for preboundary access and for interprocessor communication in the execution of shared diamonds.

For some parameter s (to be later selected), let $n = sq$ and $p|q$. We imagine the domain L_{n-1} partitioned into vertical strips of width s indexed $0, 1, \ldots, q-1$. Let μ_j denote the initial memory data pertaining to the vertical strip with left abscissa at js. We now rearrange this data, by transferring μ_j to the original position of $\mu_{\pi(j)}$, where π is the composition $\pi = \pi_2\pi_1$ of the following two permutations:

1. The index array $I = (0, \ldots, q-1)$ is subdivided into q/p segments $S_0, S_1, \ldots, S_{q/p-1}$, where $S_i = (ip, ip+1, \ldots, (i+1)p-1)$. Permutation π_1 reverses the order in the odd-indexed segments $S_1, S_3, \ldots$, producing array $\pi_1(I)$.

2. Permutation π_2 is a (q/p)-way shuffle on array $\pi_1(I)$, producing array $\pi_2\pi_1(I)$ (consisting of p segments of length q/p).

Notice that, under π_1, adjacent indices within the same segment of I remain adjacent in $\pi_1(I)$, and that adjacent indices in different segments of I become homologous (i.e., equally placed) within adjacent segments of $\pi_1(I)$. Moreover, adjacent indices within the same segment of $\pi_1(I)$ are mapped at distance (q/p) in $\pi_2\pi_1(I)$, whereas homologous indices in adjacent segments of $\pi_1(I)$ become adjacent in $\pi_2\pi_1(I)$. This proves that

- initially consecutive indices are either consecutive or at distance q/p in the rearranged array.

Finally, while π_1 does not alter the composition of a segment of I (of length p), π_2 distributes any segment of $\pi_1(I)$ uniformly among the segments of $\pi_2\pi_1(I)$ (of length p). This assures that (if processor PE_j is placed at abscissa $j(q/p)$ in I)

- for any j, there is an index of every segment of I at distance q/p from PE_j.

The simulation described below is based on the memory rearrangement defined above. When a different initial layout is given, the necessary memory rearrangement can be accomplished in a preprocessing phase of duration $O(n^2m/p)$. (Indeed, nm variables are transferred at distance $\Theta(n)$, with full parallelism p.) As this preprocessing is executed only once, its cost gives a contribution to the slowdown that vanishes as the number of simulated steps increases.

The simulation of a domain of type $D(n)$ consists of two cascaded regimes, both complying with the domain decomposition specified by the topological separator technique. The difference between the two regimes is as follows. In the first regime, called Regime 1, $D(n)$ is decomposed into domains of type $D(ps)$, and relocation occurs so that the domain preboundary is spread out in p (vertical) strips of width s, each in proximity of a processor. At this point the cooperating mode is activated (Regime 2), i.e., each vertical strip is executed either by one processor or by two adjacent processors.

Quantitatively, Regime 1 of the simulation involves *data relocations*, each stage of which halves the width of the domain to be executed, for a total of $\log(n/ps) - 1$ stages. Due to the adopted memory rearrangement, the distances at which transfers occur are reduced by a factor p with respect to the initial configuration (for a (data $\times$ distance) cost of $O(n^2m/p)$ per stage), yielding overall time

$$\tau_1 = O\left(\frac{1}{p}\frac{n^2m}{p}\log(n/ps)\right) = O\left(\frac{n^2m}{p^2}\log(n/ps)\right),$$

the factor $(1/p)$ being due to p-fold parallelism.

In Regime 2, considering $D(ps)$ partitioned vertically into p strips of width s, each processor executes individually the diamond-shaped domains of type $D(s)$ entirely within one such strip, and cooperates with two adjacent processors at distance (n/p) when executing a shared diamond. It follows that a $D(ps)$ is executed in $2p-1$ stages $\sigma_0, \sigma_1, \ldots, \sigma_{2p-2}$, where during $(\sigma_1, \sigma_3, \ldots, \sigma_{2p-3})$ each processor executes a diamond of type $D(s)$, and during $(\sigma_0, \sigma_2, \ldots, \sigma_{2p-2})$ each processor executes two semi-diamonds of type $D(s)$ and exchanges s data items with each of its two neighbors, wherever applicable (in time $O(sn/p)$). The execution of diamonds or semidiamonds by individual processors is done according to the recursive strategy of Subsection 4.1 (in time $O(s^2 \min(s, m\,\ell og(n/m)))$ –Theorem 3), for a total time τ_2'

$$\tau_2' = O(p(s^2 \min(s, m\,\ell og(s/m)) + sn/p)).$$

Since there are $\Theta((n/ps)^2)$ domains of type $D(ps)$ in $D(n)$, to be executed sequentially, the overall execution time τ_2 of Regime 2 is

$$\tau_2 = O\left(\frac{n^2m}{p}\left(\min\left(\frac{s}{m}, \ell og\frac{s}{m}\right) + \frac{n}{pms}\right)\right).$$

The sum $\tau_1 + \tau_2$ equals the total simulation time τ, which can be rewritten as

$$\tau = O(n(n/p)\Lambda(s))$$

where we have introduced the *locality slowdown* factor

$$\Lambda(s) = (m/p)\log(n/ps) + \min(s, m\,\ell og(s/m)) + n/ps.$$

An elementary but lengthy analysis will show that $\Lambda(s)$ is minimized by the following choices of s:

1. $s^* = (p/(p-1))(n/mp) \simeq n/mp$, for $1 \le m \le \sqrt{n/p}$, (range 1);
2. $s^* = \sqrt{n/p}$, for $\sqrt{n/p} < m \le \sqrt{np}$, (range 2);
3. $s^* = m/p$, for $\sqrt{np} < m \le n$, (range 3);
4. $s^* = n/p$, for $n \le m$, (range 4).

This establishes the case $d = 1$ of Theorem 1:

Theorem 4 *For $T_n \ge n$, a T_n-step computation of an $M_1(n, n, m)$ can be simulated by an $M_1(n, p, m)$ with slowdown*

$$T_p/T_n = O((n/p)\Lambda(n, m, p))$$

where the term Λ is defined as follows:

- *for* $m \leq \sqrt{n/p}$,
  $\Lambda(n,m,p) = (m/p)\log m + m\log(2n/pm^2)$,
- *for* $\sqrt{n/p} \leq m \leq \sqrt{np}$,
  $\Lambda(n,m,p) = (m/2p)\log(n/p) + 2\sqrt{n/p}$,
- *for* $\sqrt{np} \leq m \leq n$,
  $\Lambda(n,m,p) = (m/p)\log(2n/m) + n/m$,
- *for* $n \leq m$, $\Lambda(n,m,p) = n/p$.

More interesting than the analytical details is the interpretation of the results, which provides a physical explanation of the simulation. We shall examine the process as the parameter m is assumed to grow from the value $m = 1$. The four different ranges of m where Λ takes different expressions correspond to different mechanisms becoming dominant.

Initially (range 1), for $m \simeq 1$, due to the postulated memory-rearrangement, the domain to be executed by an individual processor, i.e., $D(s)$, is of type $D(n/p)$. This suggests that the memory rearrangement by itself provides a domain size (of width (n/p)) to be handled in the cooperating mode (Regime 2). Therefore Regime 1 is vacuous, as vacuous is the naive-simulation at the bottom of recursion for Regime 2. As m grows, however, Regime 1 becomes significant, as significant becomes the naive-simulation part of Regime 2. Indeed, for $m = \sqrt{n/p}$, the recursive part of Regime 2 disappears altogether, since $D(s) = D(\sqrt{n/p}) = D(m)$: At this point, $(1/2)\log(n/p)$ levels of Regime 1 are followed by naive-simulation execution. This behavior, where the optimal size s decreases from (n/p) to $\sqrt{n/p}$, results from a tradeoff between the cost of local accesses and the cost of interprocessor communication.

As m grows further, Regime 1 recedes and the naive simulation portion of Regime 2 becomes predominant, since the relocation of data becomes uneconomical with respect to the cost of the naive simulation. This recession of Regime 1 is negligible for $m \leq \sqrt{np}$ (range 2), but becomes substantial for larger values of m (range 3), and is complete for $m = n$. As m grows further, only the naive simulation is profitable, with constant locality slowdown n/p. Notice that this slowdown is attainable by the most unsophisticated simulation strategy: naive simulation of the original memory assignment in the guest (i.e., without the initial rearrangement).

5 Simulations between meshes

With the intuition gained in the analysis of simulations between linear arrays, we shall now obtain analogous results for simulations between meshes. The developments are closely patterned after those of the preceding subsections. Therefore, a more succinct style is adopted. We assume that n is a perfect square.

Theorem 5 *For $T_n \geq \sqrt{n}$, a T_n-step computation of a $M_2(n,n,1)$ can be simulated by a $M_2(n,1,1)$ with slowdown $T_1/T_n = O(n\log n)$.*

Proof We simulate $\sqrt{n}$ steps of $M_2(n,n,1)$ and carry out $\lceil T_n/\sqrt{n}\rceil$ such simulation cycles. The dag to be executed is $F_{\sqrt{n}-1} \triangleq G_{\sqrt{n}-1}(M_2(n,n,1)) = (V,A)$, whose vertex set is a cube $V = [0,\sqrt{n}-1]\times[0,\sqrt{n}-1]\times[0,\sqrt{n}-1]$. We shall make use of two types of domains of the (x,y,z)-space:

(i) an octahedron $P(\sqrt{r})$, defined as the intersection of the following eight half-spaces: $z \pm x \geq -\sqrt{r}/2, z \pm x \leq \sqrt{r}/2, z \pm y \geq -\sqrt{r}/2, z \pm y \leq \sqrt{r}/2$. Note that $|P(\sqrt{r})| = r^{3/2}/3$ and $\Gamma_{in}(P(\sqrt{r})) = 2r = 2\sqrt[3]{9}|P(\sqrt{r})|^{2/3}$;

(ii) a tetrahedron $W(\sqrt{r})$, defined as the intersection of the following four half-spaces: $z \pm y \geq 0, z \pm x = \sqrt{r}/2$. Note that $|W(\sqrt{r})| = r^{3/2}/12$ and $\Gamma_{in}(W(\sqrt{r})) = r = (12)^{2/3}|W(\sqrt{r})|^{2/3}$.

Referring to Figure 3(a) we recognize that $P(\sqrt{r})$ has an ordered partition $(P_1, W_1, W_2, W_3, W_4, P_2, P_3, P_4, P_5, W_5, W_6, W_7, W_8, P_6)$, into 14 subdomains, in which the P_j's are of type $P(\sqrt{r}/2)$ and the W_i's are of type $W(\sqrt{r}/2)$ with $|P(\sqrt{r}/2)| = 1/8|P(\sqrt{r})|$ and $|W(\sqrt{r}/2)| = 1/32|P(\sqrt{r})|$. Analogously (Figure 3(b)), $W(\sqrt{r})$ has an ordered partition $(W_1, W_2, P_1, W_3, W_4)$ of which P_1 is of type $P(\sqrt{r}/2)$ and the W_i's are of type $W(\sqrt{r}/2)$, with $|P(\sqrt{r}/2)| = 1/2|W(\sqrt{r})|$ and $|W(\sqrt{r}/2)| = 1/8|W(\sqrt{r})|$. According to Definition 6 we conclude that both $P(\sqrt{r})$ and $W(\sqrt{r})$ have a $(2\sqrt[3]{9}x^{2/3}, 1/2)$-topological separator.

We now consider domain V (Figure 4), which has an ordered partition $(P_1, P_2, P_3, P_4 W_1, W_2, W_3, W_4, P_5, W_5, W_6, W_7, W_8, P_6, P_7, P_8, P_9)$, such that the W_i's are truncated versions of $W(\sqrt{n}/2)$, P_5 is a full-fledged octahedron of type $P(\sqrt{n}/2)$ and the remaining P_j's are truncated version of $P(\sqrt{n}/2)$. It is therefore sufficient to consider $P(\sqrt{n})$ and $W(\sqrt{n})$. For $P(\sqrt{n})$, using Proposition 3, we have $c = 2\sqrt[3]{9}, \gamma = 2/3, \delta = 1/2, a = 1, \alpha = 1/2$, whence

$$\begin{aligned} \sigma(|P(\sqrt{n})|) &= \sigma_0 n, \\ \tau(|P(\sqrt{n})|) &= \tau_0 n^{3/2}\log n^{3/2}, \end{aligned}$$

where σ_0 and τ_0 are appropriate constants. An analogous result holds for $W(\sqrt{n})$. Since the simulation is to be repeated $\lceil T_n/\sqrt{n}\rceil$ times, the overall simulation time is $T_1 = O(T_n n\log n)$. □

Based on the topological separator introduced in the proof of the preceding theorem, by an approach analogous to the one exploited for linear arrays, one can establish the case $d = 2$ of Theorem 1 (details in [BP95a]).

6 Conclusions

In this paper, we have assumed a technological scenario in which the time to send a message is proportional to the distance between source and destination of the message itself. In other words, set-up time is negligible with respect to transmission time. In this scenario, we have shown that the potential speed up of a machine with n processors with respect to one with p processors is $O((n/p)\Lambda(n,p,m))$ when both machines have total memory mn. Of course, this potential can be realized only if n-fold parallelism with full locality is achievable for the application at hand. This is indeed the case for a wide class of important applications. The presence of the locality term Λ indicates that the advantages of parallelism under bounded-speed transmission can be greater than those achievable under instantaneous transmission, which are limited to n/p.

Therefore, our analysis indicates that an efficient architecture ought to exhibit a degree of parallelism proportional to the total memory size. This can be achieved by making $p = n$, in the type of machines considered in this paper. An intermediate alternative consists in the deployment of $p < n$ processors whose memory is enhanced with pipelining capabilities that would permit issuing a memory request before all the previous ones have been satisfied. For such a machine, it is not difficult to devise simulation schemes that incur no locality slowdown. On the other hand, the hardware necessary to make the memory pipelinable would be proportional to n, making the cost of such machine closer to the one with n fully-fledged processors.

An obvious question left open by this paper is whether the locality slowdown would be present in three dimensional machines. A natural conjecture is that Theorem 1 could be extended to $d = 3$ by the techniques developed in this paper, the critical step being the development of a suitable topological separator for four-dimensional domains.

Another direction for further exploration is the simulation between machines with different values of the parameter m. In fact, if an algorithm for n processors actually requires m' memory cells per processor, with $m' < m$ (recall that m is the number of cells per unit space), more locality will result in implementations with p processors.

References

[AACS87] A. Aggarwal, B. Alpern, A.K. Chandra and M. Snir. A Model for Hierarchical Memory. In *Proc. of the 19th ACM Symposium on Theory of Computing*, (1987), 305–314.

[BP92] G. Bilardi and F.P. Preparata. Horizons of Parallel Computing. In *Proc. of INRIA 25th Anniversary Symposium*, invited paper, 1992, LNCS Springer-Verlag. To appear in *Journal of Parallel and Distributed Computing*, 1995.

[BP95a] G. Bilardi and F.P. Preparata. Processor-Time Tradeoffs under Bounded-Speed Message Propagation: Part I, Upper Bounds. TR, CS Dept., Brown University, May 1995.

[BP95b] G. Bilardi and F.P. Preparata. Lower Bounds on Processor-Time Tradeoffs under Bounded-Speed Message Propagation. Manuscript, 1995.

[B74] R.P. Brent. The Parallel Evaluation of General Arithmetic Expressions. *Journal of the ACM*, (21)2:201–206, 1974.

[CR73] S.A. Cook and R.A. Reckhow. Time Bounded Random Access Machines. *Journal of Comput. System Science*, 7:354–375, 1973.

[J92] J. JáJá. *An Introduction to Parallel Algorithms* Addison-Wesley Reading Mass., 1992.

[S95] J.E. Savage. Space-Time Tradeoffs in Memory Hierarchies. TR, Dept. of Comp.Sci., Brown University, 1995.

[S86] L. Snyder. Type Architectures, Shared Memory, and the Corollary of Modest Potential. *Annual Review of Computer Science*, 1:289–317, 1986.

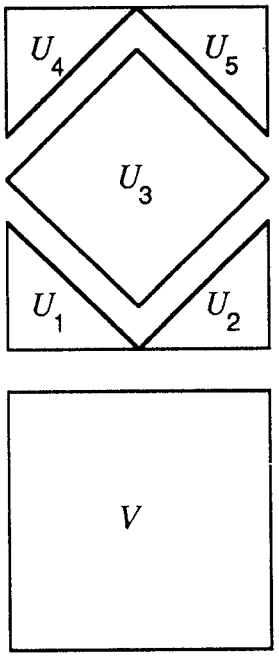

Figure 1. Partition of domain V into full or truncated instances of diamonds used by the topological-separator technique for $d = 1$.

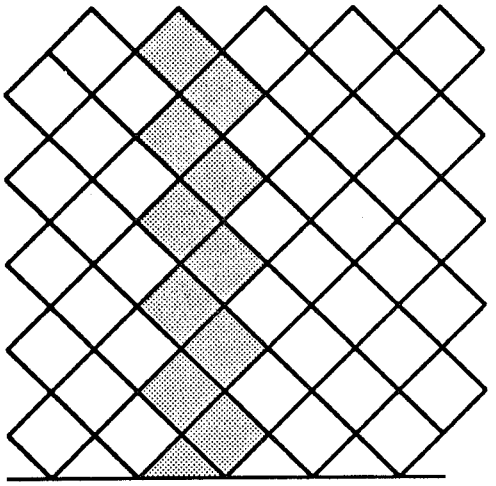

Figure 2. Zig-zag band of diamonds forming the subdomain assigned to an individual processor for $d = 1$.

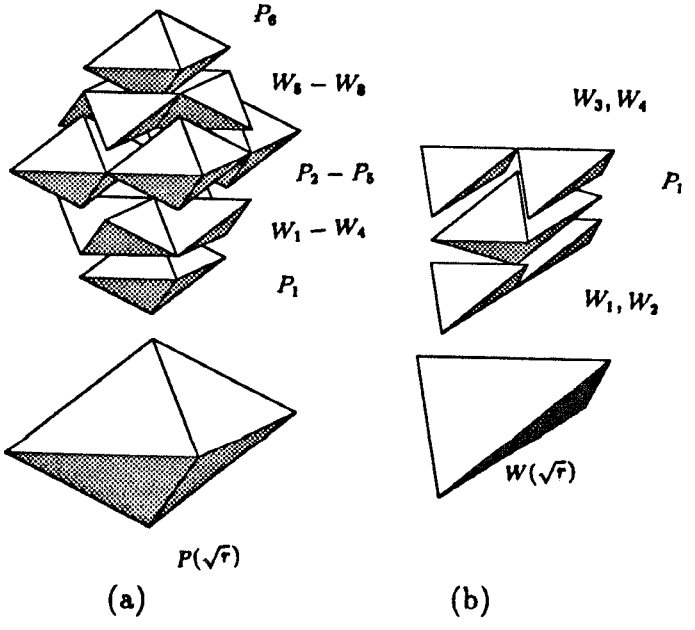

Figure 3. Recursive decomposition of octahedron (a) and tetrahedron (b) domains.

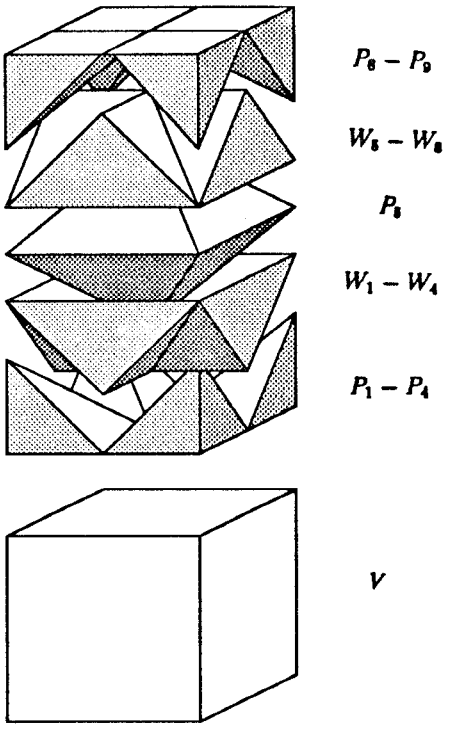

Figure 4. Partition of domain V into full or truncated instances of octahedra/tetrahedra used by the topological-separator technique for $d = 2$.