# A library of static and dynamic communication algorithms for parallel computation [*]

Emmanouel A. Varvarigos

*Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA*

Communication efficiency is one of the keys to the broad success of parallel computation, as one can see by looking at the successes of parallel computation, which are currently limited to applications that have small communication requirements, or applications that use a small number of processors. In order to use fine grain parallel computation for a broader range of applications, efficient algorithms to execute the underlying interprocessor communications have to be developed. In this paper we survey several generic static and dynamic communication problems that are important for parallel computation, and present some general methodologies for addressing these problems. Our objective is to obtain a collection of communication algorithms to execute certain prototype communication tasks that arise often in applications. These algorithms can be called as communication primitives by the programmer or the compiler of a multiprocessor computer, in the same way that subroutines implementing standard functions are called from a library of functions in a conventional computer. We discuss both algorithms to execute *static* (deterministic) primitive communication tasks, as well as schemes that are appropriate for *dynamic* (stochastic) environments. Our emphasis is on algorithms that apply to many similar problems and can be used in various network topologies.

## 1. Introduction

The impressive progress in hardware technology during the past decades has enabled the use of thousands of processors within the same computer. $N$ processors working on some problem, however, do not necessarily solve it $N$ times faster than a single processor working alone. A major (but not the only) cause of inefficiency is the communication overhead. Processors, when doing computations, often have to exchange intermediate results. In many cases, the time required for routing the messages is more than the time spent by the processors doing computations. Thus, at least with current technology, communication seems to be the bottleneck of parallel computation.

Our aim is to describe a collection of communication algorithms to organize the movement of data within a message passing parallel computer in order to achieve efficient (close to 100%, if possible) utilization of the communication resources. We

[*] Research supported by NSF under Grant NSF-RIA-08930554. Part of the work was performed when the author was a visitor at T.U. Delft.

first study static communication problems, where all packets involved are available at the same time. For such communication tasks, the main objective is to design algorithms that execute them in the shortest possible time. We then consider dynamic routing problems, where packets are generated at random times over an infinite time horizon. In such dynamic environments, the main objective is to design algorithms that are of the on-line type and distributed, and that can perform the communications efficiently according to some probabilistic criteria, such as the average delay and the average throughput.

When choosing the communication tasks to investigate we are motivated by the underlying computations that give rise to the information exchange between the processors (e.g., matrix multiplication and transposition, image processing applications, fixed point computations). The performance of a parallel computing system will depend on the number of generic traffic patterns (scenarios) that its designers have anticipated, and the efficiency of the algorithms that they have precomputed and included in a communication library to execute them. Our emphasis is on methodologies that are applicable to a number of multiprocessor networks rather than being specific to a particular topology. Furthermore, some of these methodologies deal with whole classes of communication tasks rather than with individual tasks. We consider this important because one of the problems of parallel computation theory is the lack of unified approaches designed to deal with similar problems.

The traffic patterns that we discuss arise in both MIMD and SIMD machines. The results on static communication tasks, however, will be most useful for SIMD machines. The reason is that in SIMD machines all processors execute the same instructions; this makes the resulting communication pattern more regular, easily describable, and predictable, which leaves more space for clever organizations of the data movement. We are mainly interested in fine grain parallel computing systems for two reasons. The first is that such systems can achieve speedups of several orders of magnitude compared to conventional computers. A second reason is that in fine grain parallel computers, the communication part becomes more significant due to the larger size of the network and the information exchange that has to take place.

## 2. Primitive communication problems

We first define some of the primitive communication problems that we consider important for parallel computing applications. In particular, section 2.1 deals with static (deterministic) communication tasks, while section 2.2 deals with dynamic (stochastic) communication problems. Later, in section 3, we discuss general methodologies for obtaining efficient algorithms for some of these problems.

### *2.1. Static communication tasks*

There are some traffic patterns that arise frequently in applications. As a result it is desirable to know in advance communication algorithms that execute them in the minimum number of steps. The time required to execute these primitive tasks is a good performance measure for comparing parallel computers (see [3]). Saad and Shultz [20,21] were the first to consider such problems and to propose corresponding routing algorithms. Johnsson and Ho [13], Bertsekas et al. [2], and Varvarigos and Bertsekas [28–31] have developed minimum and nearly minimum completion time algorithms for similar routing problems using various communication models, for a number of topologies. Several other related works that deal with primitive communication problems can be found in [4,6–12,15,16,19,24–26].

In this section we describe briefly some of the static primitive communication tasks that we believe should be part of the communication library of parallel computers. Our description will follow a hierarchical order, going from the least expensive tasks (that is, those that require a small number of packet transmissions) to the most expensive ones, from those that involve a single node to those that involve multiple nodes, and from the special cases to their generalizations. This order gives a natural way to organize the communication library: the compiler or the programmer should look for the least expensive among the prototype tasks that execute the communication pattern at hand.

The simplest of all communication tasks is *one-to-one* (1–1) communication, where a node sends a packet to some other node, and it corresponds to a processor accessing a nonlocal memory location. The relevant performance criteria in this case are the diameter and the mean internodal distance of the network.

A more complicated task is the *single node broadcast* (SNB for brevity), where the same packet is sent (copied) from a given node to all other nodes. A single node broadcast can be accomplished by transmitting the packet along a spanning tree rooted at a given node. It corresponds to a variable being read by all processors simultaneously. A dual task is the *single node accumulation* (SNA), where a packet is sent to a given node from every other node; here, we assume that the packets can be combined on a link, with the combined transmission time being equal to the transmission time of a single packet. The operator used in combining the packets can be any associative operator, such as $+$, $\cdot$, min, max, and others. The SNA implements a concurrent write from all nodes to a single memory location. The spanning tree used for the SNB with the arcs reversed can be used for the SNA task as well; thus, the time complexity of both tasks is equal to the diameter of the network. A problem more general than the SNB is the *selective single node broadcast* (SSNB), where the same packet is sent from a given node to some, but not necessarily all, other nodes. A dual to the SSNB task is the *selective single node accumulation* (SSNA), where several nodes of the network send a packet to a particular node; here packets can be combined on any link (so the SSNA task is a generalization of the SNA task). A SSNB (respectively SSNA) corresponds to the case where a set of processors read (respectively write to) the same memory location.

A task more expensive than the previous ones is the *single node scatter* (SNS), which involves sending a separate packet from a given node to every other node (here a node sends different packets to different nodes in contrast with the single node broadcast where a node sends the same packet to all other nodes). A dual problem, called *single node gather* problem (SNG), involves collecting a packet at a given node from every other node. An algorithm that solves the single node scatter problem consists of a schedule of packet transmissions on links that properly takes queueing into account. By reversing the schedule it can be seen that given an algorithm that solves the single node scatter problem, we can find an algorithm that solves the single node gather problem and takes the same amount of communication time. A generalization of the SNS (or SNG) problem is the *selective single node scatter* (respectively *selective single node gather*), abbreviated SSNS (respectively SSNG), where a different packet is sent from (respectively collected at) a given node to some (respectively from some), but not necessarily all, other nodes.

In all the previous tasks, there is a single node that is executing a command and is sending packets to (or collecting packets from) one or more nodes. Generalizations of the previous tasks, where the same action is taken by many or perhaps all processors, also appear very frequently in applications. For example, in SIMD machines where all processors are controlled by the same instruction stream (each processor has, of course, the option of ignoring a command) we expect the same task to be executed by all or a large subset of the nodes. Even in MIMD machines, there are many algorithms that require the tasks described above to be executed by many or all nodes.

A multinode version of the one-to-one communication is the *permutation task*, where every processor sends a packet to some other node, with no two destinations being the same. A multinode version of the single node broadcast task is the *multinode broadcast* (or MNB). In the MNB each node wishes to broadcast a packet to all the other nodes. A task more general than the MNB is the *partial multinode broadcast* (or PMNB) where an (arbitrary) subset of the nodes wants to broadcast a packet to all the nodes. The PMNB task, along with being important on its own merit, is also a critical component of the dynamic broadcasting algorithms discussed in section 3.4 for general interconnection networks. A dual to the partial multinode broadcast problem is the *partial multinode accumulation* (PMNA), where every node of the network sends a packet to each node in a given set of nodes; here, we assume that packets with the same destination can be combined on a link, with the operator used in combining the packets being any associative operator. The PMNA implements a concurrent write from all nodes to each one of a given set of memory locations (a different number is written at each location). The PMNB and the PMNA are dual tasks to each other: origin (destination) nodes in the PMNB correspond to destination (origin) nodes in the PMNA, and the copying operation in the PMNB corresponds to the combining operation in the PMNA.

A multinode version of the single node scatter is the *total exchange* (TE) task, where each node sends a separate (personalized) packet to every other node. A total exchange arises, for example, during the transposition of an $N \times N$ matrix stored by columns in an $N$-processor computer. To transpose the matrix, every processor $i$ has to send the $(i, k)$th entry to processor $k$, for all $k$, which is a total exchange. An even more general task than the total exchange is the *partial multinode scatter* (or PMNS),

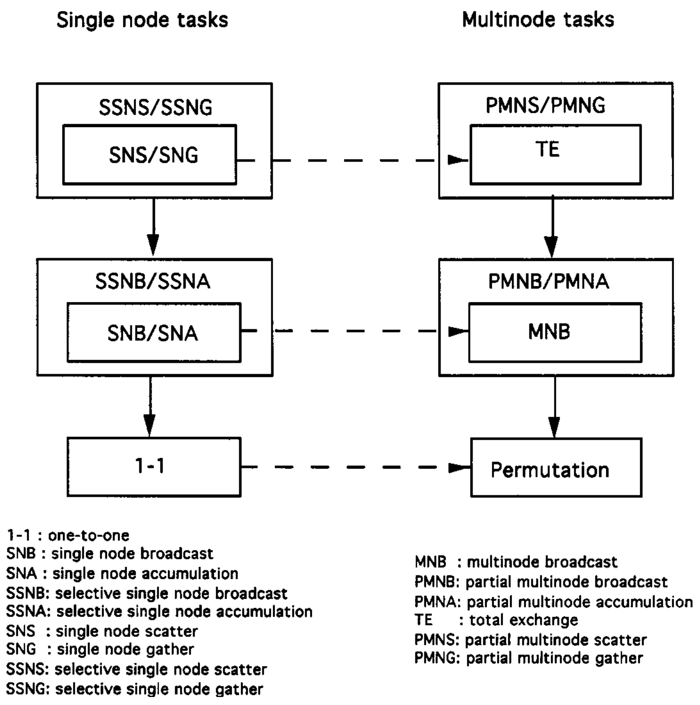

Single node tasks
Multinode tasks
SSNS/SSNG
SNS/SNG
PMNS/PMNG
TE
SSNB/SSNA
SNB/SNA
PMNB/PMNA
MNB
1-1
Permutation
1-1 : one-to-one
SNB : single node broadcast
SNA : single node accumulation
SSNB: selective single node broadcast
SSNA: selective single node accumulation
SNS : single node scatter
SNG : single node gather
SSNS: selective single node scatter
SSNG: selective single node gather
MNB : multinode broadcast
PMNB: partial multinode broadcast
PMNA: partial multinode accumulation
TE : total exchange
PMNS: partial multinode scatter
PMNG: partial multinode gather

Figure 1. The hierarchy of the primitive static communication tasks. A directed arc from problem A to problem B indicates that an algorithm that solves problem A can also solve problem B. The inclusion of the box corresponding to problem A in the box corresponding to problem B indicates that problem B is a generalization of problem A. The tasks at the left involve either a single transmitting node or a single collecting node. A dashed directed arc from problem A to problem B indicates that problem B is a multinode version of problem A.

where an (arbitrary) subset of the nodes of an $N$-processor system sends a separate packet to each of the $N-1$ other processors. A dual of the partial multinode scatter problem is the *partial multinode gather* problem (or PMNG), where each node in a subset of nodes of the network collects a separate packet from every other node of the network. Examples where the MNB, PMNB, PMNA, TE, PMNS, and PMNG tasks appear are given in section 3. The PMNS is the highest problem in hierarchy among the previously described problems in terms of difficulty. Given an algorithm for the PMNS we can find algorithms to execute all the other tasks that we presented, although usually not efficiently.

A last class of communication tasks that will be of interest to us is the class of *isotropic tasks*, introduced in [28]. These are tasks defined in regular networks (as are most of the networks used in parallel computation), which are characterized by a type of symmetry with respect to each origin. For example, the total exchange problem in a hypercube, a $d$-dimensional mesh, a folded-cube, or a Manhattan Street network is an isotropic task; the communication problem “looks identical” to every node. Figure 1 shows the primitive communication tasks in a hierarchical order.

## *2.2. Dynamic communication problems*

All the tasks described in section 2.1 are *static* in the sense that there is some work to be performed once and for all. All the nodes know which task they execute, and (for the multinode tasks) they are synchronized to start at the same time; the only objective is to finish the job as fast as possible. Optimal or near-optimal algorithms (schedules) for static communication tasks are precomputed, and are called when needed in a parallel program.

Except for the static tasks, where conditions are rather favorable, one can envision situations where communication requests are not deterministic, but they are generated at random instants. We call such an environment *dynamic*. The execution of asynchronous computation algorithms is one such situation, but it is reasonable to expect that in many systems a dynamic, largely unpredictable communication environment may be the rule and not the exception. Multitasking, time-sharing, run-time generation of communication requests, and difficulty of identifying prototype tasks at compilation time are some other reasons that make the use of precomputed static algorithms difficult, and motivate us to look for dynamic schemes that will run continuously, and execute on-line the communication requests.

We have chosen to discuss two dynamic communication problems, which we call the dynamic broadcasting and the dynamic scattering problem, both because of their potential for increasing the general understanding of dynamic problems and for their importance in their own right. In these problems, communication requests (broadcasts or scatters) are generated at random instants at each node of a network according to a random process over an infinite time horizon. The performance criteria that we will use are the average number of communication requests that are executed per unit of time (throughput), and the average delay to serve a request. Communication schemes for the dynamic broadcasting and the dynamic scattering problems can only be useful if they are on-line, distributed, and easy to implement.

The fact that the communication requests (broadcasts or scatters) are generated randomly does not necessarily imply that they should be executed in a unorganized way. For example, in the case where broadcast requests are generated at random instants at each node, it was found in [32] that some global information is very helpful. Such information can be gathered on line at very little cost, and can be used to organize the packet transmissions.

## **3. Algorithms for primitive communication problems**

In this section we describe some general methodologies for obtaining algorithms for the primitive communication problems defined in the previous section. In particular, in section 3.1 we consider the class of isotropic tasks, which include a number of important communication problems as special cases. In section 3.2 we focus on static communication tasks that arise during the transposition of sparse matrices. In sections 3.3 and 3.4 we turn our attention to broadcasting problems; in particular, in section 3.3

we discuss static broadcasting tasks, while in section 3.4 we consider the dynamic broadcasting problem. Finally, in section 3.5 we discuss how the methodologies developed for dynamic broadcasting can be extended to other dynamic communication problems.

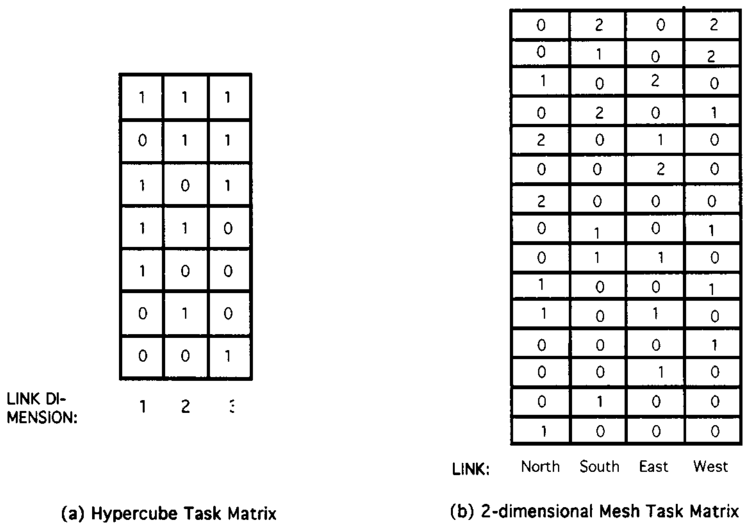
We assume throughout the section that information is transmitted in the form of packets, each of which requires one unit of time to be transmitted over a link. This is usually a realistic assumption. It also constitutes an abstraction that enables us to focus on the many remaining issues involved in the design of communication algorithms. This abstraction provides a way to measure the performance of communication algorithms, and compare it with lower and upper bounds. Some authors [13,20] divide the delay required for a packet transmission over a link into two parts: a start-up delay which is the same for all packets, and a transmission delay which is proportional to the length of a packet. We believe that such a distinction complicates the picture without providing enough additional insight. In any case, appropriate modifications can always be made with little effort to convert the complexity results that we obtain to the other model.

We assume that packets are transmitted in a store and forward fashion. This leaves out the option of wormhole routing [5], where a packet starts being transmitted before it has been fully received at a node. In this way, the packet delay is reduced due to pipelining of the packet bits over several links. Most of the static algorithms that we discuss achieve utilization close to 100% of some critical communication resource, and therefore they cannot be improved by using wormhole or some other kind of routing. Wormhole routing becomes very interesting when the load in the network is light and the messages are relatively long.

## *3.1. Isotropic and nearly isotropic tasks*

The class of *isotropic* tasks was introduced in [28] for hypercubes and $d$-dimensional meshes, and were later extended to other topologies [25,26]. Isotropic tasks are characterized by transmission requirements that are symmetric with respect to origin node. For example, the total exchange problem (see section 2.1) is an isotropic task; the communication problem "looks identical" to every node. The structure of isotropic tasks can be exploited particularly well in networks that have themselves a symmetric structure, such as a hypercube, a wraparound mesh, a folded-cube, a Manhattan Street, and other networks.

An important data structure that is useful in dealing with isotropic tasks is that of the *task matrix* of node $s$, which will be denoted by $T(s)$. The task matrix $T(s)$ is the matrix that contains as rows the routing tags of all the packets that have to be sent by node $s$ for the task to be completed. Of course, the routing tags of the packets are defined in a different way for each regular topology (see [25,26,28]). For example, in $d$-dimensional hypercubes, the routing tag of a packet is defined as the bitwise exclusive OR operation $s \oplus t$ between the source $s$ and the destination address $t$ of the packet. In a $d$-dimensional wraparound mesh, the routing tag of a packet is



| LINK DIMENSION: 1 | 2 | 3 |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

**(a) Hypercube Task Matrix**

| LINK: North | South | East | West |
|---|---|---|---|
| 0 | 2 | 0 | 2 |
| 0 | 1 | 0 | 2 |
| 1 | 0 | 2 | 0 |
| 0 | 2 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 0 | 0 | 2 | 0 |
| 2 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |

**(b) 2-dimensional Mesh Task Matrix**

Figure 2. (a) The task matrix for the TE problem in a $d$-dimensional hypercube has $2^d - 1$ rows and $d$ columns. The figure illustrates the case where $d = 3$. The critical sum $h$ in this case is equal to 4 and it is also equal to the optimal completion time of the TE. (b) Illustrates the task matrix for the TE problem in a 2-dimensional mesh with 3 nodes along each dimension. The critical sum (and the completion time of the TE task) is equal to 8. Note that the ask matrix is defined in a different way for each topology.

a $2d$-dimensional vector that describes the relative location of the destination with respect to the source of the packet (this definition for the mesh is more convenient for our pursposes than the more common definition of the routing tag as a $d$-dimensional vector whose entries can take both positive and negative values; see [28] for more details). For a communication task in a regular topology to be isotropic, we require that $T(s) = T$ for all nodes $s$. Figure 2 indicates the task matrix $T$ that corresponds to a total exchange in a 3-dimensional hypercube and in a $3 \times 3$ wraparound mesh.

By limiting our attention to symmetric routing algorithms, we can show that executing isotropic tasks in regular topologies is equivalent to solving a matrix decomposition problem. This relates the routing problem, which is a scheduling problem with a combinatorial character, with a matrix decomposition problem, which is a problem in linear algebra. Such a connection provides a simpler and more powerful characterization of optimal routing algorithms for the total exchange and other related problems than in other works (e.g., [2,7,9,13,21]). It also allows simple and elegant analyses of minimum average delay algorithms and near-optimal greedy algorithms; see [28].

We define the *critical sum* $h$ of the task matrix $T$ to be equal to $\max_{i,j}(r_i, c_j)$, where $r_i$ is the sum of the entries of row $i$, $c_j$ is the sum of the entries of column $j$, and the maximization is performed over all rows $i$ and columns $j$ of $T$. It can be shown that the optimal completion time of an isotropic task in a hypercube, $d$-dimensional mesh, folded-cube, or Manhattan Street network is equal to the critical sum $h$ of its task matrix (recall that the routing tags and, as a result, the task matrices are defined in a different way in each of these topologies). It can also be shown using this formulation that a class of particularly simple-minded algorithms can achieve completion time which is larger than the optimal execution time by at most a small constant. These algorithms are required to satisfy just a very weak and natural restriction; they must never leave a communication link idle as long as there is a waiting packet that can reduce its distance to its destination by using this link (see [28]).

Isotropic tasks turned out to be a practically important and analytically interesting class of communication problems. They also motivate a number of new interesting research directions. First, it is clear that there is an incentive to formulate new routing problems in terms of isotropic or *"nearly isotropic"* tasks, whenever this is reasonable, to take advantage of the corresponding simple and elegant analysis. For this to happen, it is important to develop criteria that capture the notion of "closeness" to an isotropic task. One idea (used in [31]) is to characterize a task as nearly isotropic when it can be divided into an isotropic and a nonisotropic part, such that the transmission requirements of the latter are smaller than the transmission requirements of the former. A number of problems that arise in image processing and in numerical algorithms would fall into the class of nearly isotropic tasks according to this criterion. For tasks that satisfy this criterion, near-optimal algorithms could be found by expanding upon the theory of isotropic tasks.

The efficient communication algorithms obtained for hypercubes [28], $d$-dimensional meshes [28], folded cubes [25], and Manhattan Street [26] networks indicate that it may be both possible and fruitful to define isotropic and nearly isotropic tasks for other networks of interest. Another problem is to examine the properties of a network (e.g., regularity, hierarchical structure) that if present will make isotropic tasks on this network analytically tractable. This would permit us to deal with the total exchange and other important related problems in a unified way for a number of topologies.

### 3.2. *Transposition of sparse matrices*

We have mentioned that the transposition of a (dense) matrix stored by columns in a parallel computer is equivalent to a total exchange, which is the most expensive task among those defined in section 2.1 (e.g., it requires $N/2$ steps to execute in a hypercube). The question that arises is the following. Can we perform the transposition faster when the matrix is sparse? Since most large matrices are sparse, the importance of this question is evident.

An algorithm for transposing a *banded* matrix (i.e., a matrix that has nonzero entries only within a band around its diagonal) stored by columns in a hypercube was proposed in [31], and was shown to be of the optimal order of magnitude when the matrix bandwidth $B$ is $\Theta(N^c)$, for some constant $c > 0$, where $N$ is the number of processors and optimality is considered over all possible column-to-processor assignments. The method used in this work was to store the banded matrix in such a way that the communication requirements for the transposition task become "almost identical" for all nodes. It still remains an open question to find an algorithm that will be optimal

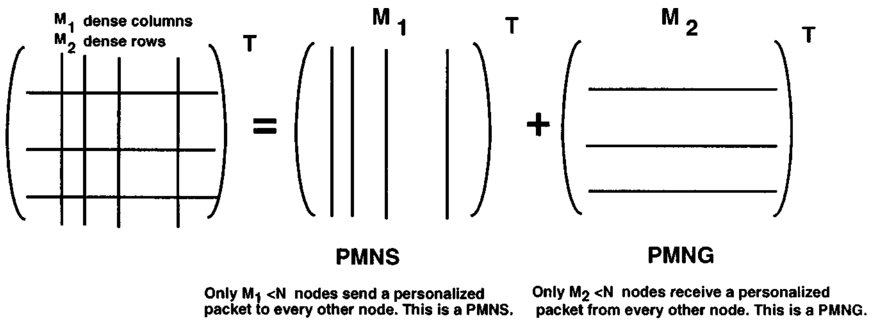

M1 dense columns
M2 dense rows
T
=
M 1
T
+
M 2
T
PMNS
PMNG
Only M1 <N nodes send a personalized packet to every other node. This is a PMNS.
Only M2 <N nodes receive a personalized packet from every other node. This is a PMNG.

Figure 3. The transposition of a matrix that has $M_1$ dense rows and $M_2$ dense columns can be performed by executing a PMNS followed by a PMNG. Here we assume that each processor stores a column of the matrix.

for *any* bandwidth $B$ in a hypercube. The problem is also still open, to the best of our knowledge, for all other topologies of interest.

Other matrix sparsity patterns also deserve attention. Suppose, for example, that we are given an $N \times N$ matrix with only $M_1$ dense (nonzero) columns, stored by columns in a multiprocessor computer. The transposition of this matrix results in the *partial multinode scatter* task (PMNS), where only $M_1$ arbitrary nodes send a (separate) packet to every other node. Similarly, if the matrix has $M_2$ dense rows then its transposition results in a *partial multinode gather* task (PMNG), where $M_2$ arbitrary nodes collect a (distinct) packet from every other node of the network (the combining of packets originated at different nodes is not allowed). By combining a PMNS and a PMNG algorithm we obtain an algorithm to transpose a matrix with $M_1$ dense columns and $M_2$ dense rows, located at arbitrary positions (see figure 3). The smaller $M_1$ and $M_2$ are, the more beneficiary it becomes to use efficient PMNS and PMNG algorithms instead of TE algorithms. PMNS and PMNG algorithms of optimal order were given in [30] for hypercubes and 2-dimensional arrays. Given the variety of applications where these two tasks arise we believe it is important to find algorithms to execute them in other topologies of interest. Even in the case of the hypercube and the 2-dimensional array, more work is required to improve the constants that appear in the time complexity results of the existing algorithms.

## 3.3. *The partial multinode broadcast problem*

Consider iterations of the form $x = f(x)$ where each processor computes an entry (or some entries) of the vector $x$. At the end of each iteration it is necessary that each processor broadcasts the updated value of the component that it computes to all other processors in order to be used at the next iteration. This a multinode broadcast (MNB). Algorithms to execute the MNB task in several topologies of interest are given in [1–3,9,13,21,29].

In iterations of the kind given above it is probable that only some of the components of the vector $x$ change appreciably during an iteration. As these iterations approach their convergence point, fewer and fewer of the processors need to broadcast the updated values of the components of $x$ that they compute. This gives rise to the partial multinode broadcast task (PMNB), where a strict (but unpredictable) subset of the processors has to broadcast a packet. The PMNB task arises also in clustering algorithms (see [18]), and other problems. The PMNB task, aside from being important on its own merit, is also a critical subroutine of the dynamic broadcast schemes that we will examine in the next subsection. Because of its many applications we believe that the PMNB deserves a position among the prototype tasks of a communication library.

Algorithms to execute the PMNB task have been proposed in [23] for hypercubes, and in [30,32] for hypercubes and $d$-dimensional meshes. The PMNB schemes in the previous references use edge-disjoint spanning trees to perform the broadcasts; each packet selects one of the trees and is broadcast on it in a greedy way, with conflicts being resolved arbitrarily (in the hybercube PMNB algorithm of [23], a special node gathers all the packets and decides the tree on which each of them is broadcast; in the PMNB algorithms given in [30,32] the choice of the tree is made locally at each node, based on some global parameters that each node obtains through a parallel prefix operation).

The PMNB problem in a general graph was considered in [27]. It was shown that if a graph $G$ has $k$ edge-disjoint spanning trees $T^1, \dots, T^k$ with diameters $L_1, \dots, L_k$, respectively, then PMNB task in $G$, can be executed in at most

$$T_{\mathrm{PMNB}} \leqslant \frac{M}{k} + \overline{L} + 2\delta$$

slots, where $M$ is the total number of packets that have to be broadcast, $\overline{L} = \sum_{i=1}^{k} L_i/k$ is the average of the tree diameters, and $\delta$ is the diameter of $G$. The algorithm consists of two phases: in the first phase, a parallel prefix operation is performed, where each node learns the total number of packets that have to be broadcast (and some additional information). This information is used by the nodes in the second phase to choose the spanning tree on which each packet is broadcast. The algorithm works even with the (more restrictive) half-duplex communication model, where a link cannot be used for both transmission and reception at the same time. A lower bound for the time required to execute a PMNB in a graph that has a maximum number $k$ of edge-disjoint spanning trees can be found to be

$$T_{\mathrm{PMNB}} \geqslant \max\left(\frac{M}{2k}, \delta\right),$$

for the full-duplex communication model, and

$$T_{\mathrm{PMNB}} \geqslant \max\left(\frac{M}{k}, \delta\right),$$

for the half-duplex communication model. These lower bounds hold for any choice of the initial locations of packets that have to be broadcast. Reducing the gap between the upper and the lower bounds on the time required to execute the PMNB task in a general graph (especially for the full-duplex model) is an open question.

It turns out that by finding an algorithm for the PMNB problem in a given network, we immediately obtain a scheme for the dynamic broadcasting problem in that network. The dynamic broadcasting problem is described next.

### *3.4. The dynamic broadcasting problem*

In *static* broadcasting tasks considered in section 3.3, we assume that at time $t = 0$ some particular (but probably unspecified) nodes have to broadcast a packet to all the nodes once and for all. Static broadcasting tasks of various kinds have been studied extensively in the parallel computer literature (see references in the previous subsection). In the *dynamic broadcasting problem*, we assume that broadcast requests are generated at each node according to a random process with rate $\lambda$ (we will assume independent Poisson processes in the analysis). We are interested in routing schemes that work correctly in such a stochastic environment, and in their performance.

We use three criteria to evaluate the performance of a dynamic broadcasting scheme in a given network topology. The first criterion is the *maximum broadcast throughput*, which is the maximum generation rate per node $\lambda$ that can be accommodated by a broadcasting scheme with the queueing delays being finite. The second criterion is the *average broadcast delay* $\mathcal{B}$, which is the average time that elapses between the generation of a packet at a node and the time its broadcast to all the other nodes is completed. The third criterion is the *average reception delay* $\mathcal{R}$, which is the average time that elapses between the generation of a packet at a node and the time a particular node $s$ receives a copy of the packet, averaged over all nodes $s$ of the network. Since for the broadcast of a packet to be completed all nodes must receive a copy of it, we have $\mathcal{R} \leqslant \mathcal{B}$, for any broadcasting scheme. We set two objectives for a dynamic broadcasting scheme: stability for as big a load as possible, and average broadcast and reception delays that are of the order of the diameter for any fixed load in the stability region.

The dynamic broadcasting problem has been considered in [22] for hypercubes, [17] for 2-dimensional meshes, and [25,26,30,32] for hypercubes, $d$-dimensional meshes, folded-cubes and Manhattan Street networks, respectively. The two hypercube algorithms proposed in [22] that are most interesting from a theoretical point of view are the nonidling scheme with priority rule, and the idling scheme. The stability objective described above is met by the nonidling scheme, but its average delay analysis is approximate. The idling scheme, on the other hand, provably meets the delay objective, but its stability region is roughly $2/3$ of the maximum possible. The schemes proposed in [30,32] for hypercubes and meshes have a stability region that tends to the maximum possible as the number of nodes tends to infinity, and an average delay that is of the order of the diameter of the network for any *fixed* load in the stability region. However, if the asymptotics of the delay are taken *simultaneously* with respect to the load *and* with respect to the network size, then the delay has not been proved to be optimal (there is a gap between the average delay of the scheme, and a universal lower bound on the average delay of any broadcasting scheme). Recently [33], a priority broadcast scheme that outperforms the previous schemes for the hypercube was proposed; its analysis, however, is approximate.

The dynamic broadcasting problem for an *arbitrary* topology has been examined in [27], where an *indirect* and a *direct* dynamic broadcasting scheme were proposed. The indirect broadcasting scheme essentially consists of the successive execution of PMNB algorithms, each starting when the previous one has finished. Even though the duration of each PMNB period is random (because packet arrivals are random), it is known to all the nodes of the network, because each node learns during the parallel prefix operation of the PMNB algorithm the total number of packets $M$ that have to be broadcast, and, from there, the duration of the PMNB period (which is at most $M/k + \overline{L} + 2\delta$). The performance of the indirect broadcasting scheme is given by the following theorem.

**Indirect broadcasting scheme theorem.** Let $G$ be a (bidirectional) network that has $k$ edge-disjoint trees. Let $\overline{L}$ be the average diameter of the edge-disjoint spanning trees, and $\delta$ be the diameter of $G$, and assume that the broadcasts are generated at each node of the network according to a Poisson process with rate $\lambda$, independently of the other nodes. Then the indirect broadcasting scheme that uses the PMNB of section 3.3 is stable for

$$\lambda < \frac{k}{N},$$

and has average broadcast delay $\mathcal{B}$ that satisfies

$$\mathcal{B} \leqslant \frac{\rho^2 - \rho + 2}{2k(1-\rho)} + \frac{(\overline{L} + 2\delta)(1+\rho)(3-\rho)}{2(1-\rho)},$$

where $\rho = k/N$. Moreover, the average broadcast delay for light load satisfies

$$\mathcal{B} \leqslant 1.5 \cdot \overline{L} + 3\delta + \frac{1}{k}, \quad \rho \approx 0.$$

In the second broadcasting scheme (called the *direct* broadcasting scheme) each packet selects upon its generation one of the $k$ edge-disjoint spanning trees, and is broadcast on it in a greedy way. A packet that is broadcast on tree $T^j$ is referred to as a packet of class $j$. At every slot, every node $v$ considers each of its incident links $(v, w) \in T^j$. If $v$ has received a packet of class $j$ that it has neither sent already to $w$ nor it has yet received from $w$, then $v$ sends such a packet on link $(v, w)$. If $v$ does not have such a packet it sends nothing on $(v, w)$. When more than one packets are eligible for transmission on link $(v, w)$, one of them is transmitted and the remaining are queued. The performance of the direct broadcasting scheme can be evaluated analytically without using any approximating assumptions and is given by the following theorem.

**Direct broadcasting scheme theorem.** Consider a graph $G$ that has $k$ edge-disjoint trees $T^j$, $j = 1, 2, \ldots, k$, and let $l^j_{st}$ be the distance between nodes $s$ and $t$ using only links of tree $T^j$. Let $T^{j,1}(s)$, $T^{j,2}(s), \ldots, T^{j,m}(s)$ be the subtrees in which $T^j$ is partitioned when $s$ is removed, and $n_{j,q}(s)$ the number of nodes of tree $T^{j,q}(s)$. If the broadcasts are generated at each node of the network according to a Poisson process with rate $\lambda$, independently of the other nodes, then the average reception delay $\mathcal{R}$ of the direct broadcasting scheme is

$$\mathcal{R} = \frac{1}{2} + \bar{l} + \frac{\lambda}{2k_{\max}N(N-1)} \sum_{s=1}^{N} \sum_{j=1}^{k_{\max}} \sum_{q=1}^{m} \frac{n_{j,q}(s)^2}{1 - n_{j,q}(s)\lambda/k_{\max}},$$

where

$$\bar{l} = \sum_{j=1}^{k} \sum_{s=1}^{N} \frac{l^j_s}{kN}$$

is the mean internodal distance of the spanning trees, averaged over all nodes in a tree and over all trees. The direct broadcasting scheme is stable if and only if

$$\lambda < \frac{k}{N-1}.$$

Note that even though the expression for the average reception delay in the preceding theorem may look complicated, all the parameters involved in it are known once the spanning trees have been chosen.

It can be shown [27] that a necessary stability condition for a general graph is $\lambda < 2k/N$, for the full-duplex communication model, and $\lambda < k/N$, for the half-duplex model, where $k$ is the maximum number of edge-disjoint spanning trees of the graph, and $N$ is the number of nodes. Therefore, for the half-duplex model the stability region of the indirect broadcasting scheme is equal to the maximum possible, while for the half-duplex model the stability regions of both the direct and the indirect broadcasting scheme are half of that given by the corresponding upper bound (note that the direct broadcasting scheme assumes the full-duplex communication model, while the indirect broadcasting scheme works under both the full-duplex and the half-duplex communication model).

For any fixed load in the stability region, the average broadcast delay $\mathcal{B}$ of the indirect scheme is O($\overline{L}$), and the average reception delay $\mathcal{R}$ of the direct scheme is O($\bar{l}$), where $\overline{L}$ and $\bar{l}$ denote the average of the diameters and the average of the mean internodal distances of the edge-disjoint spanning trees, respectively. For topologies where the edge-disjoint spanning trees can be chosen so that $\overline{L}$ (or $\bar{l}$) is O($\delta$), the indirect (or direct, respectively) broadcasting scheme is of the optimal order in terms of average delay (networks where this is possible include trees, and meshes of arbitrary dimension with or without wraparound). This is because, under our communication model, the network diameter is a lower bound on any broadcasting task. Note also that there is a trade-off between stability region and average broadcast delay. If we use the maximum possible number of edge-disjoint trees (in order to have the best stability region), we may have to select trees of large diameter (increasing the delay for light load).

### *3.5. Other dynamic communication problems*

The indirect dynamic broadcasting scheme theorem suggests that there is an interesting connection between static and dynamic routing problems, which needs to be examined in a more general context. The following theorem essentially reduces a dynamic communication problem to its corresponding static problem, which may be an easier problem to solve.

**Static-to-dynamic communication theorem.** Assume that for a given $N$-processor network there exists an algorithm that executes a static communication task in time

$$XM + V,$$

where $M$ is the number of communication requests that are involved and $X$, $V$ are scalars that are independent of $M$ (they may depend on the network under consideration). Assume also that during the static algorithm each node learns the value of $M$. Then the indirect dynamic communication scheme that consists of the successive repetition of static communication algorithms, each starting when the previous one has finished, solves the corresponding dynamic communication problem, and has the following performance characteristics. If the communication requests are generated at each node of the network according to a Poisson process with rate $\lambda$, independently of the other nodes, the average delay $\mathcal{T}$ to serve a request satisfies

$$\mathcal{T} \leqslant W + X + \min(W - V, \rho W),$$

where $\rho = \lambda N X$ and

$$W = \frac{\rho X}{2(1-\rho)} + \frac{V}{2} + \frac{V}{1-\rho}.$$

Also, the dynamic scheme is stable for any load $\lambda$ that satisfies

$$\lambda < \frac{1}{NX}.$$

To give an example, consider the *dynamic scattering problem*, where scatter requests are generated at each node at random times with rate $\lambda$. To perform a scatter, a node has to send a *separate* packet to every other node (i.e., a node sends a total of $N-1$ distinct packets). This problem has not, to the best of our knowledge, appeared in the literature so far. We set the following performance objectives for a dynamic scattering scheme: (1) stability for the maximum possible $\lambda$, and (2) average packet delay for light load ($\lambda \to 0$) that is of the order O($T_{\text{SNS}}$), where $T_{\text{SNS}}$ is the time required to execute a single node scatter in the given network. For example, in a $d$-dimensional hypercube, a single node scatter requires $T_{\mathrm{SNS}} = \lceil (N-1)/d \rceil$ steps (see [3]), and a total of $dN/2$ packet transmissions. Since there are $Nd$ links in a hypercube, and the total arrival rate of scatter requests is $N\lambda$ a necessary condition for stability is $(N\lambda) \cdot (dN/2) \leqslant Nd$, or $\lambda \leqslant 2/N$. Therefore, the maximum stable load (measured in scatters per node and unit of time) that we can hope to have for the dynamic scattering problem in a hypercube is $\lambda \leqslant 2/N$, and the average delay to complete a scatter cannot be smaller than $\Theta((N-1)/d)$ for any load in the stability region.

One approach for solving the dynamic scattering problem might be to try to find an algorithm for the static version of the problem, which is the PMNS problem described in section 2.1, and then use the static-to-dynamic communication theorem given above. Note that for this theorem to apply, the execution time of the static algorithm has to be linear in the number of active nodes $M$. Also, since the constant of proportionality $X$ is intimately related to the stability the dynamic scheme, it is important that it is the smallest possible. Efficient algorithms for the PMNS problem have not appeared in the literature so far (the PMNS algorithms given in [30] for hypercubes and 2-dimensional meshes are of the optimal order of magnitude, but they involve large constants, and they do not result in efficient dynamic scattering algorithms), but we believe they are important components of a communication library for parallel computers.

## 4. Conclusions

Parallel computing is essentially a different way to organize and access the memory of a computer. The critical resources in a multiprocessor computer are the communication bandwidth and the memory capacity; this is underscored by the fact that these two alone occupy more than 90% of the area of a parallel computer (see [5,12]). Since all the components of a parallel computer (memory, processors, routers) are made with the same technology, the cost is proportional to the area. Therefore, the processors are relatively cheap, and there is no great penalty for processors being idle. The true waste occurs when the communication bandwidth and the memory are not efficiently used. Since our ability to access nonlocal memory locations fast is also determined by the efficiency of the communications, efficient solutions to primitive communication problems are central in the success of parallel computation.

We have reviewed some important communication tasks, which we believe should be part of the communication library of parallel computers. The algorithms and methodologies that we presented deal in a unified way with many topologies and communication tasks (e.g., isotropic and nearly isotropic tasks, dynamic broadcasting and scattering). This is especially desirable, since the solutions obtained for particular problems can be easily modified to solve other problems. For example, by modeling the total exchange problem as an isotropic task one can solve it simultaneously for a number of regular topologies (hypercubes, $d$-dimensional meshes, folded-cubes, Manhattan Street networks, and possibly others). Also, it is easier for the designer of a parallel computer to implement simple rules (such as the simple-minded scheme for isotropic tasks mentioned in section 3.1, or the greedy scheme used in the direct dynamic broadcasting algorithm) that when followed perform the communications at hand than it is to implement a large number of specific schedules. We also presented some results on the critical relation between static and dynamic communication problems; these results may also be of interest in other areas of engineering.

## References

[1] M.M. Azevedo, N. Bagherzadeh and S. Latifi, Broadcasting algorithms for the star-connected cycles interconnection network, Journal of Parallel and Distributed Computing 25(2) (1995) 209–222.

[2] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng and J.N. Tsitsiklis, Optimal communication algorithms for hypercubes, Journal of Parallel and Distributed Computing 11 (1991) 263–275.

[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[4] K. Coolsaet and V. Fack, Optimal data exchange algorithms on star graphs, Computers and Mathematics with Applications 27(33) (1994) 21–25.

[5] W.J. Dally, Network and processor architecture for message-driven computers, in: *VLSI and Parallel Computation*, eds. R. Suaya and G. Birtwistle (Morgan Kaufmann, San Mateo, CA, 1990) pp. 140–222.

[6] A. Edelman, Optimal matrix transposition and bit reversal on hypercubes: All-to-all personalized communication, Journal of Parallel and Distributed Computing 11 (1991) 328–331.

[7] P. Fragopoulou and S.G. Akl, Optimal communication algorithms on star graphs using spanning tree constructions, Journal of Parallel and Distributed Computing 24 (1995) 55–71.

[8] P. Fragopoulou, S.G. Akl and H. Meijer, Optimal communication primitives on the generalized hypercube network, Journal of Parallel and Distributed Computing 32 (1996) 173–187.

[9] P. Fraigniaud, Complexity analysis of broadcasting in hypercubes with restricted communication capabilities, Journal of Parallel and Distributed Computing 16 (1992) 15–26.

[10] M.D. Grammatikakis, D.F. Hsu, M. Kraetzl and J.F. Sibeyn, Packet routing in fixed-connection networks: A survey, to appear in Journal of Parallel and Distributed Computing.

[11] S.M. Hedetniemi, S.T. Hedetniemi and A.L. Liestman, A survey of gossiping and broadcasting in communication networks, Networks 18 (1988) 319–349.

[12] S.L. Johnsson, Communication in network architectures, in: *VLSI and Parallel Computation*, eds. R. Suaya and G. Birtwistle (Morgan Kaufmann, San Mateo, CA, 1990) pp. 223–389.

[13] S.L. Johnsson and C.T. Ho, Optimum broadcasting and personalized communication in hypercubes, IEEE Transactions on Computers 38 (1989) 1249–1268.

[14] Y. Lan, A.-H. Esfahanian and L. Ni, Multicast in hypercube multiprocessors, Journal of Parallel and Distributed Computing 8 (1990) 30–41.

[15] O.A. McBryan and E.F. Van de Velde, Hypercube algorithms and their implementations, SIAM Journal on Scientific and Statical Computing 8 (1987) 227–287.

[16] J. Misic and Z. Jovanovic, Communication aspects of the star graph interconnection network, IEEE Transactions on Parallel and Distributed Systems (1994) 678–687.

[17] E. Modiano and A. Ephremides, Efficient algorithms for performing packet broadcasts in a mesh network, IEEE/ACM Transactions on Networking 4(4) (1996) 639–648.

[18] S. Ranka and S. Sahni, *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition* (Springer, New York, 1990).

[19] Y. Saad and M.H. Schultz, Topological properties of hypercubes, IEEE Transactions on Computers 37 (1988) 867–872.

[20] Y. Saad and M.H. Schultz, Data communication in hypercubes, Journal of Parallel and Distributed Computing 6 (1989) 115–135.

[21] Y. Saad and M.H. Schultz, Data communication in parallel architectures, Parallel Computing 11 (1989) 131–150.

[22] G.D. Stamoulis and J.N. Tsitsiklis, Efficient routing schemes for multiple broadcasts in hypercubes, IEEE Transactions on Parallel and Distributed Systems 4 (1993) 725–739.

[23] G.D. Stamoulis and J.N. Tsitsiklis, An efficient algorithm for multiple simultaneous broadcasts in the hypercube, Information Processing Letters (July 1993) 219–224.

[24] Q.F. Stout, Intensive hypercube communication, Journal of Parallel and Distributed Computing 10 (1990) 167–181.

[25] E.A. Varvarigos, Optimal routing algorithms for folded-cubes, in: *Proc. of IEEE Internat. Phoenix Conf. on Computers and Communications* (1995) pp. 143–151.

[26] E.A. Varvarigos, Optimal communication algorithms for Manhattan Street networks, Discrete Applied Mathematics 83(1–3) (March 1998) 303–326.

[27] E.A. Varvarigos and A. Banerjee, Routing schemes for multiple random broadcasts in arbitrary network topologies, IEEE Transactions on Parallel and Distributed Systems (August 1996) 886–895.

[28] E.A. Varvarigos and D.P. Bertsekas, Communication algorithms for isotropic tasks in hypercubes and wraparound meshes, Parallel Computing 18 (1992) 1233–1257.

[29] E.A. Varvarigos and D.P. Bertsekas, Multinode broadcast in hypercubes and rings with randomly distributed lengths of packets, IEEE Transactions on Parallel and Distributed Systems 4(2) (1993) 144–154.

[30] E.A. Varvarigos and D.P. Bertsekas, Partial multinode broadcast and partial exchange in $d$-dimensional wraparound meshes, Journal of Parallel and Distributed Computing 23 (1994) 177–189.

[31] E.A. Varvarigos and D.P. Bertsekas, Transposition of banded matrices in hypercubes: A nearly isotropic task, Parallel Computing 21 (1995) 243–264.

[32] E.A. Varvarigos and D.P. Bertsekas, Dynamic broadcasting in parallel computing, IEEE Transactions on Parallel and Distributed Systems 6(2) (1995) 120–131.

[33] C.H. Yeh, E.A. Varvarigos and H. Lee, The priority broadcast scheme for dynamic broadcast in hypercubes and related networks, in: *Proc. of 7th Symp. on the Frontiers of Massively Parallel Computation*, Annapolis (February 1999) pp. 294–301.