Implementing the MPI Process Topology Mechanism*

Jesper Larsson Träff
C&C Research Laboratories, NEC Europe Ltd.
Rathausallee 10, D-53757 Sankt Augustin, Germany
traff@ccrl-nece.de

Abstract

The topology functionality of the *Message Passing Interface* (MPI) provides a portable, architecture-independent means for adapting application programs to the communication architecture of the target hardware. However, current MPI implementations rarely go beyond the most trivial implementation, and simply performs no process remapping.

We discuss the potential of the topology mechanism for systems with a hierarchical communication architecture like clusters of SMP nodes. The MPI topology functionality is a weak mechanism, and we argue about some of its shortcomings. We formulate the topology optimization problem as a *graph embedding problem*, and show that for hierarchical systems it can be solved by *graph partitioning*. We state the properties of a new heuristic for solving both the embedding problem and the "easier" graph partitioning problem.

The graph partitioning based framework has been fully implemented in MPI/SX for the NEC SX-series of parallel vector computers. MPI/SX is thus one of very few MPI implementations with a non-trivial topology functionality. On a 4 node NEC SX-6 significant communication performance improvements are achieved with synthetic MPI benchmarks.

1 Introduction

To achieve portability, the *Message Passing Interface* (MPI) consciously hides the communication architecture of the underlying parallel system from the application programmer, who works abstractly with sets of processes and communication modes like point-to-point, one-sided and collective communication [3, 11]. It is the responsibility of every MPI implementation to exploit its target architecture as efficiently as possible by using the most suitable communication channels and best possible algorithms for collective communication operations. But even the best MPI implementation cannot ensure that for instance frequently communicating processes are placed "close" to each other. On todays clustered/hierarchical

*0-7695-1524-X/02 $17.00 ©2002 IEEE

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 IEEE

or even wide-area systems where communication performance between processes is highly dependent on process placement, proper process assignment is an important performance issue. *Virtual communication topologies* and a possibility for process remapping is the means within MPI of adapting applications to specific communication architectures [11, Chapter 6]. The topology mechanism thus provides a handle for the application programmer to more efficiently utilize systems with a hierarchical communication structure as exhibited for instance by clusters of SMP nodes. Unfortunately, current MPI implementations do not go beyond the most trivial realization of the topology functionality, and essentially ignore the virtual topology information.

In this paper we present a general framework for implementing the topology functionality on systems whose communication system can be modeled by a (weighted) graph, and formulate the resulting optimization problems as *graph-embedding* and *graph-partitioning problems*. These problems are NP-complete, but there are a number of heuristics that can produce good solutions cheaply. We state the properties of a new Kernighan-Lin like heuristic for direct graph k-partitioning [12]. The heuristic can give better results than standard algorithms using recursive bipartitioning, and in particular allows to improve not only the value of a total cut, but also the maximum value of edges adjacent to any one subset. This is an important property for the topology application. The heuristic has been incorporated into the MPI/SX implementation for the NEC SX-series of parallel vector computers. In multi-node SX-configurations up to 8 processors per node can communicate via shared memory, and nodes are interconnected via the ultra-fast IXS crossbar switch. Multi-node SX-systems are examples of high-performance hierarchical systems. We give results for synthetic MPI benchmarks using various graph and grid communication patterns. These have been constructed so as to allow most communication to be mapped to intra-node, shared memory communication. The results show that the partitioning heuristic is able to achieve a good remapping, resulting in significant improvements in communication performance of up to a factor of 9.

2 MPI Process Topologies

In MPI a set of processes is represented by a *communicator*, which also represents a (static) mapping of MPI processes to physical processors. A *virtual process topology* is a communicator with user-provided, high-level information describing the preferred communication pattern for the processes in the communicator. There are two types of virtual topologies. In a *graph topology* the communication pattern is an undirected graph, and communication is assumed to be between processes that are adjacent in the graph. In a *Cartesian topology* the communication pattern is a d-dimensional grid, and communication is assumed to be among processes at neighboring grid-points, which in the MPI standard implicitly means along a dimension, not along a diagonal. From the application programmer's point of view both types of topologies describe a preferred or likely communication pattern: no communication between processes in a virtual topology is forbidden.

Virtual topologies are created by collective operations. The calls `MPI_Graph_create(comm, ..., reorder, &graphcomm)` and `MPI_Cart_create(comm, ..., reorder, &cartcomm)` are collective over the communicator `comm` and return new communicators `graphcomm` or `cartcomm`

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

with the topology information associated. The processes in the new communicator may be *reranked* (if `reorder` is set to true): process i in the new communicator has rank i' in the "old" `comm`, and it may be that $i \neq i'$. A pair of processes (i, j) that will communicate in the virtual topology can thus be mapped to a pair (i', j') that in old `comm` happens to reside on, say, the same shared-memory node. After remapping data redistribution may be necessary: the "old" process i in `comm` may have to route data to process i in the new communicator, that is to process i' in the old `comm`. There is unfortunately no collective operation in MPI for permutation routing to assist with this kind of redistribution. In the `MPI_Graph_create` call all processes must supply the full communication graph. This is a non-scalable construct, and furthermore error-prone since the processes must indeed supply an identical description of the graph. Apart from the remapping information provided implicitly by comparing the old and new communicator, there are no high-level inquiry functions on virtual topologies to provide information on the potential communication improvements of a remapping. This could be helpful for the application programmer in deciding (at run time) whether the benefits of working with the reordered, new topology may outweigh the costs of data redistribution.

In MPI a virtual topology is a static description of a *communication pattern*. It is not possible to inform the MPI implementation about when communication will take place, neither about the load (frequency, amount of data) of particular links. It is also not possible to influence how the reranking should be performed, but as will be seen in the following there is more than one possible optimization criterion. The application programmer has no means of informing the MPI implementation which criterion is the most relevant. For the Cartesian topology it is application dependent whether neighbors are found only along the dimensions, or may also include the processes on the diagonals. The exact structure of the neighborhood could possibly have influence on the remapping, but again there is no means of informing the MPI implementation which neighborhood will be used in a given application. A possible shortcoming of a different nature has to do with the fact that the topology remapping problems are NP-hard (for most communication architectures), so that there will be a tradeoff between solution quality and invested time. It is not possible via MPI to specify how much time the MPI implementation should spend in finding a good remapping satisfying the optimization criteria, but only the application programmer can know how much he can afford to spend in a given situation.

A high-quality MPI will use the topology information provided by the user to reorder the processes in the communicator to fit well with the underlying communication system. The identity mapping, that is returning a communicator in which processes have not changed ranks, is a legal implementation of the topology mechanism. Apparently, no current MPI implementation goes beyond this trivial realization; a possible exception, though, is HP MPI [4].

3 The MPI Topology Mapping Problem

Assume that the communication system of the target machine can be described by an undirected, weighted *host graph* $H = (V', E')$. The vertices V' correspond to processors, and the weight $w'(a, b) \geq 0$ of edge $(a, b) \in E'$ models the cost of communication between processors

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

a and b. Thus weights are non-negative, and a larger weight shall mean more expensive communication (higher latency, higher transmission cost). We assume that communication from a to b has the same cost as communication from b to a. We assume that H is complete: the underlying routing system will ensure that every processor can communicate with every other processor. Note that this simplistic model does not account for traffic in the network which may affect communication costs.

Let a virtual topology be given as an undirected, unweighted *communication graph* $G = (V, E)$. We handle MPI Cartesian topologies by the corresponding grid graphs, constructed implicitly by the implementation. The nodes of G are MPI processes, and there is an edge (u, v) between processes u and v if they want to communicate in the virtual topology. In an MPI communicator there can be no overloading of processors, so $|V| \leq |V'|$; for simplicity we assume $|V| = |V'|$ in this section)=.

The *MPI topology mapping problem* is to find an embedding of G into H, that is an injective function $\pi : V \to V'$, that in some sense optimizes the placement of the processes V onto the processors V' with respect to overall communication costs. There are at least two relevant optimization criteria for the MPI topology mapping problem:

- Minimizing total *Communication Costs* by a mapping $\pi : V \to V'$ that minimizes

$$\mathrm{CC}(\pi) = \sum_{(u,v)\in E} w'(\pi(u), \pi(v))$$

  i.e. minimizes the total use of expensive communication links $(a, b) \in E'$ in the host graph.

- Optimizing *Load Balance* by a mapping $\pi : V \to V'$ that minimizes

$$\mathrm{LB}(\pi) = \max_{u\in V} \sum_{v\in \mathrm{ADJ}(u)} w'(\pi(u), \pi(v))$$

  where $\mathrm{ADJ}(u) = \{v|(u, v) \in E\}$, i.e. minimizes the maximum number of expensive communication links for any processor.

These problems are NP-complete, except for special graphs; they are trivial for complete communication graphs since every edge of the host graph will be the target of an edge of the communication graph.

For systems with a hierarchical communication system we can make more assumptions about the host graph. We describe our hierarchical system by a tree as shown in Figure 1. The individual processors appear at the leaves, each at the same depth h. We call this a h-level system. The model can be extended to more complex systems, but we consider only balanced systems here. An interior node at level ℓ represents a set of processors that can communicate with each other with the same cost $w^\ell(a, b)$. At level 0 all processors can communicate via the most expensive medium of the system. At the leaves each processor can communicate with itself via memory copy, and at intermediate levels processors can communicate via for instance shared memory or other communication medium connecting a subset of the processors of the system. Without loss of generality we assume that $w^0 \geq$

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 IEEE

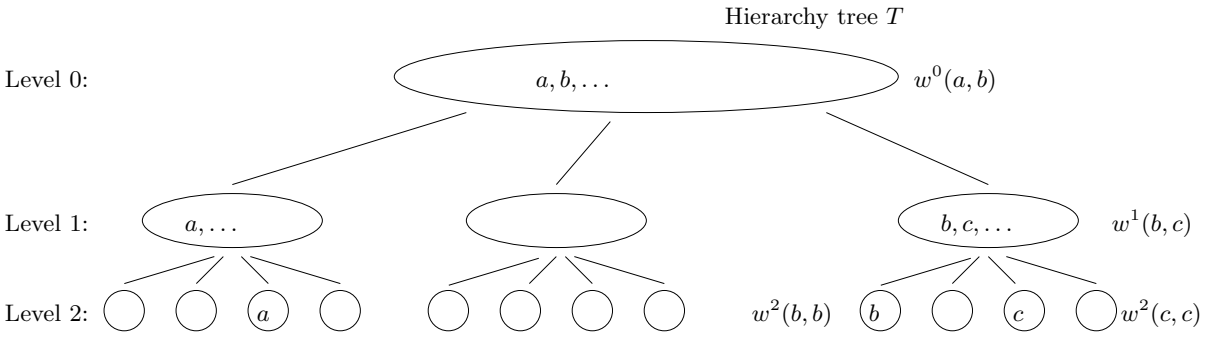

Figure 1: Hierarchy tree for a two-level system.

$w^1 \geq \ldots \geq w^\ell \geq \ldots \geq w^h$. The cost of communication between arbitrary processors a and b is

$$w'(a, b) \quad = \quad w^{\mathrm{lca}(\{a\},\{b\})}(a, b)$$

where $\mathrm{lca}(A, B)$ denotes the lowest common ancestor of two nodes A and B in the hierarchy tree T. In other words processors communicate via the cheapest medium through which they are connected. A homogeneous SMP cluster is an example of a 2-level hierarchical system. Processors on the same shared memory node communicate via shared memory, all intra-node communication can be assumed to be equally costly, and communication between processors on different nodes is performed by a network that can often be assumed to be homogeneous: inter-node communication cost is the same for all processor pairs.

We aim to show that the embedding problems for hierarchical systems can be solved by *(weighted) graph partitioning* with generalized cost function. Let again $G = (V, E)$ be a communication graph with possible edge weights $w(u, v)$ for $(u, w) \in E$. Let a $k > 1$ and subset sizes $s_1, \ldots, s_k$ with $\sum s_i = |V|$ be given. Let $V_1, \ldots, V_k$ be a partition of the vertex set V of the communication graph into k disjoint subsets with $|V_i| = s_i$.

We now introduce a *generalized cost function* $c(u, v)$ on G, which depends on both the edge $(u, v) \in E$ and on the current partition, expressed as a product $w(u, v)W(i, j)$ of an edge weight $w(u, v)$ and a inter-subset weight $W(i, j)$ for the subsets $u \in V_i$ and $v \in V_j$ containing u and v. This kind of cost function was also used in [5].

Define

$$\begin{aligned} \mathrm{totalcut}(V_1, \ldots, V_k, E, c) &= \sum_{\{(u,v)\in E | u\in V_i, v\in V_j, i\neq j\}} c(u, v) \\ \mathrm{maxcut}(V_1, \ldots, V_k, E, c) &= \max_{1\leq i\leq k} \sum_{\{(u,v)\in E | u\in V_i, v\notin V_i\}} c(u, v) \end{aligned}$$

Similarly to the embedding problems, we define two optimization problems, namely

- $\min_{V_1,\ldots,V_k} \mathrm{totalcut}(V_1, \ldots, V_k, E, c)$, i.e. finding a partition that minimizes the total value of all edges between different subsets
- $\min_{V_1,\ldots,V_k} \mathrm{maxcut}(V_1, \ldots, V_k, E, c)$, i.e. finding a partition that minimizes the maximum number of edges adjacent to any subset of the partition

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

If we require all subsets to have size 1, and take the generalized cost function $c(u,v)$ to be $w'(a,b)$ when $u \in V_a$ and $v \in V_b$, that is, when processes u and v are mapped to processors a and b, it is clear that the embedding problem is a special case of the weighted partitioning problem (with generalized cost function) with the correspondence totalcut = CC and maxcut = LB. For hierarchical systems we can reduce the problem to a (more) standard graph partitioning problem, in particular reduce the number k of subsets of the partition.

Theorem 1 *Let $G = (V, E)$ be a communication graph. For a h-level hierarchical system the embedding problem of optimizing for total communication cost* CC *can be solved by weighted graph partitioning (with generalized cost function), optimizing for* totalcut *with subset sizes $|V_i|$ equal to the sizes of the nodes of the hierarchical system at level $h-1$.*

PROOF: We have seen above a trivial solution with $k = |V|$ and all $s_i = 1$. To reduce the number of subsets we make the following observations.

Let k be the number of nodes at level $h-1$ in T, and number the nodes N_i arbitrarily from 1 to k. Let the subset sizes for the partitioning problem be the sizes of the $(h-1)$-level nodes, $s_i = |N_i|$. Let w_0 be the cost of communication within the $(h-1)$-level nodes (which is assumed to be the same for all nodes N_i). We define the generalized cost function for the graph partitioning problem by

$$c(u,v) \quad = \quad w^{\mathrm{lca}(N_i,N_j)}(u,v) - w_0$$

when $u \in V_i$ and $v \in V_j$.

Assume now that we have a partition $V_1, \ldots, V_k$ that *minimizes* totalcut$(V_1, \ldots, V_k, E, c)$. From this partition we construct an embedding f of G by taking $f(u)$ for $u \in V_i$ to be an arbitrary processor a in N_i. The value CC(f) is clearly independent of the choice of a, since all $a \in N_i$ have the same edge weights. We have

$$\mathrm{CC}(f) \quad = \quad \mathrm{totalcut}(V_1, \ldots, V_k, E, c) + |E|w_0$$

since w_0 by the definition of $c(u,v)$ does not contribute to the cut edges, and since all node-internal (non-cut) edges have weight w_0.

We claim that this embedding is optimal. Assume for the sake of contradiction that a better embedding π exists. From π we construct a partition $V'_1, \ldots, V'_k$ of V by setting $V'_i = \{u \in V | \pi(u) \in N_i\}$ (note that by the numbering of the nodes N_i, $|V_i| = |V'_i|$). The value of this partition with respect to the generalized cost function c is

$$\mathrm{totalcut}'(V'_1, \ldots, V'_k, E, c) \quad = \quad \mathrm{CC}(\pi) - |E|w_0$$

Thus CC$(\pi) <$ CC(f) implies totalcut$'(V'_1, \ldots, V'_k, E, c) <$ totalcut$(V_1, \ldots, V_k, E, c)$, contradicting the optimality of the partition $V_1, \ldots, V_k$. □

For the important case of two-level systems, like clusters of SMP nodes, the embedding problem reduces to *unweighted* graph partitioning. Such a system has intra-node communication costs w_0 and inter-node communication costs w_1 with $w_0 < w_1$.

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

Theorem 2 *Let $G = (V, E)$ be a communication graph. For two-level hierarchical systems the embedding problem of optimizing for total communication cost can be solved by unweighted graph partitioning , optimizing for* totalcut *with subset sizes $|V_i|$ equal to the sizes of the level one nodes of the hierarchical system.*

PROOF: Let an optimal partition $V_1, \ldots, V_k$ be given from which we trivially construct an embedding f as above. The value of the embedding is $\mathrm{CC}(f) = \mathrm{totalcut}(V_1, \ldots, V_k, E)w_1 + |\{(u, v) \in E | u \in V_i, v \in V_i, i = 1, \ldots, k\}|w_0$, that is the value of the cut plus the total weight of the intra-node (non-cut) edges. Again this embedding is optimal. Assume on the contrary that an embedding π with π with $\mathrm{CC}(\pi) < \mathrm{CC}(f)$ exists. The value $\mathrm{CC}(\pi)$ can be written as a cut value plus a contribution by the intra-node edges. It cannot be that $\mathrm{CC}(\pi) < \mathrm{CC}(f)$ by π having fewer intra-node edges than f, because then π would have more external edges, and since $w_0 < w_1$ this would mean $\mathrm{CC}(\pi) > \mathrm{CC}(f)$. Thus, the only way $\mathrm{CC}(\pi) < \mathrm{CC}(f)$ would be by having a smaller totalcut, contradicting that the partition $V_1, \ldots, V_k$ was optimal. □

Analogous observations holds for the *load balance embedding problem* (LB): we can use maxcut-partitioning to obtain an embedding which minimizes the maximum weight of edges adjacent to any node N_i. The embedding thus optimizes the load balance of the $(h-1)$-level nodes, but *not* the load balance of each individual processor.

4 Solving the optimization problems

Both the graph embedding and the graph partitioning problem are NP-complete, and thus cannot for even moderately large graphs be solved exactly. Many heuristics have been proposed for the partition problem that produce good results fast. Currently the best results are achieved by multilevel methods [5, 6, 13]. These methods at some point use a refinement heuristic to improve the quality of an intermediate partition.

For graph bipartitioning a particularly simple and elegant heuristic is the so called Kernighan-Lin heuristic [8]. An initial bipartition is improved by repeatedly constructing a sequence of swaps of pairs of nodes which overall improves the solution quality. The power of the heuristic stems from the fact that it can tolerate intermediate swaps which increase the cut-value, as long as the total sequence leads to a decrease. Experimental evidence suggests that the number of times such a swap sequence can be found is small (constant). Thus, the complexity of the Kernighan-Lin heuristic largely depends on the complexity of finding the sequence of swaps. A fast, but theoretically weaker (in terms of solution quality) algorithm for finding moves (instead of swaps) was by Fiduccia and Mattheyses [1], the so called "linear-time" heuristic.

The k-partitioning required to solve the topology optimization problems can be done by recursive bipartitioning. However, with recursive bipartitioning it is difficult to control the value of the overall maximum cut (as required by the load balance embedding problem (LB) and thus, by the theorems in Section 3, the maxcut minimization problem), and recursive bipartitioning can, for the totalcut problem, lead to bad results [10], even if bipartitioning is done optimally. A further, minor drawback is that the running time by recursive bipartitioning increases by a $\log k$ factor over the bipartitioning time. In [12] a new algorithm for

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 IEEE

direct k-partitioning is given, which shows that the complexity of constructing a sequence of vertex moves that improves the overall solution is independent of k. By having a global view of the current k-partition, this new heuristic makes it possible to maintain also the current value of maxcut, and thus find a sequence of swaps that improves maxcut. The algorithm is simple to implement, and is the basis for the MPI/SX topology implementation. The following results are proved in [12].

Let a graph $G = (V, E)$, $k > 1$ and subset sizes $s_1, \ldots, s_k, \sum s_i = |V|$ be given. Let the cost function $c(u, v)$ depend only on edge the (u, v) (and not on the subsets to which u and v currently belong), that is we consider the standard graph partitioning problem.

Theorem 3 *The data structures (priority queues) needed to select the next vertex to move can be initialized in $O(n+m)$ time. For weighted graphs G, the vertex resulting in the largest decrease in total cut value can be selected in $O(\log n)$ time, and the data structure updates for the next move can be done in $O(d \log n)$ time steps, where d is the degree of the selected vertex. A sequence of up to n moves that improves the current solution can thus be found in $O((n + m) \log n)$ time steps.*

Corollary 1 *For unweighted graphs, special data structures reduce the selection time for the best vertex to move to $O(1)$ and the update time to $O(d)$, where d is the degree of the selected vertex.*

The heuristic can also tolerate imbalance, i.e. work with the relaxation that $|V_i| = s_i + b(i)$, where $b(i)$ is some positive or negative amount of unbalance tolerated for subset V_i, and $\sum s_i = |V|$. For the MPI topology functionality we use this property to find a good embedding for the case where $|V| < |V'|$, that is, for the case where the host graph (in MPI implicitly given by the base communicator) has more nodes than the communication graph (in MPI given by the virtual topology). In this case there are $(h-1)$-level nodes V'_i that will not become full. We take $s_i = \min\{|V'_i|, |V| - \sum_{j<i} s_j\}$, and allow imbalance $b(i) = -s_i$ for the full sets $s_i = |V'_i|$ and imbalance $b(i) = |V'_i| - s_i$ for the remaining sets with $s_i < |V'_i|$. Allowing imbalance gives the heuristic more possibilities to find a partition with good cut value.

For the partitioning problem with generalized cost function $c(u, v)$ depending also on the subsets to which u and v currently belong, we have the following.

Theorem 4 *Data structures can be initialized in $O(n + m)$ steps. The best vertex to move can be selected in $O(\log n)$ time, while the updates of data structures for the next move takes $O(kd \log n)$ time steps. Thus a sequence of up to n best moves can be found in $O((n + km) \log n)$ time steps.*

The generalized partition problem, which as remarked includes the embedding problem, is a factor k more expensive to solve with the Kernighan-Lin like heuristic than graph partitioning with a simple cost function. Theorem 1 makes it possible for hierarchical systems to reduce the number of subsets from $|V|$ to some $k < |V|$ equal to the number of nodes at level $h-1$ of the hierarchical system. For two-level systems Theorem 2 ensures that standard graph partitioning suffices, which for the heuristic solution saves a factor k in the running time.

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

5 An implementation for the NEC SX-systems

The NEC SX-systems are powerful parallel vector machines with a hierarchical communication architecture. For the most recent in the series, the NEC (Cray) SX-6, up to 8 vector processors, each with a peak performance of 8GFlops can be connected via a shared memory. Shared-memory nodes can be connected via a special, high-performance, bidirectional crossbar switch. The communication bandwidth achieved with MPI/SX, the NEC proprietary MPI implementation, is about 15GBytes/second for intra-node communication, and about 6GBytes/second for inter-node IXS communication. Also for one-sided and collective communication MPI/SX is optimized to take advantage of shared memory communication whenever possible. Because of the difference between intra- and inter-node bandwidth, process remapping via the MPI topology mechanism can potentially have a large impact on overall application performance.

In MPI/SX the communicator data structure stores information about the mapping of processes to processors and nodes; in particular, any process can in constant time determine the number of processes on its node as well as on all other nodes. In addition to the user-provided communication graph (explicit or as a Cartesian topology) node sizes are input to the k-partitioning procedure, which is executed only by the process with rank 0; the resulting partition which serves as the basis for the reordering is broadcast to the other processes participating in the collective topology creation call. Although sequential in nature (one vertex is selected after the other when building up the move sequence) aspects of the k-partitioning algorithm can be vectorized, in particular the expensive initialization of the necessary priority queue data structures. By further tuning of the cut-off criteria, acceptable running times (whatever that means, see Section 2) have been achieved. Parallel algorithms for graph partitioning have been proposed [2, 7, 9, 14], and may have to be considered for large systems. This, however, is future research.

6 Experimental evaluation

In this section we give a first experimental evaluation of the potential benefits of the topology mechanism, and our specific implementation in terms of graph partitioning. Evaluation so far has been done only with a synthetic benchmark with the following structure:

- a virtual topology over a *base communicator* is set up,
- (optional:) data are redistributed from processes in the base communicator that have been remapped in the new communicator,
- each process in the virtual topology sends/receives (non-blocking) a message to/from each of its neighbors, then waits locally for communication to complete. A number of repetitions is performed.

This is done for explicit communication graphs with varying structure and for different Cartesian topologies. For graph topologies the base communicator is `MPI_COMM_WORLD`, while for Cartesian topologies we give results for both a randomly reranked base communicator and

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 IEEE

for `MPI_COMM_WORLD`. Most applications can be expected to set up a Cartesian topology over `MPI_COMM_WORLD`, so the latter makes a fair comparison to the trivial topology implementation possible. For the Cartesian topologies communication is done with neighbors along the dimensions as suggested by the MPI standard. It is worth noting here that applications often communicate also with neighbors along the diagonals.

For graph topologies we use communication graphs with the following structure:

1. random graph, each vertex has random degree between 0 and 6, and randomly generated edges,
2. collection of random clusters of size 4, each vertex has 2 random edges within its cluster, and 2 random edges possibly crossing to other clusters,
3. collection of random clusters of size 16, internal vertex degree 8, 2 random edges to other clusters per vertex,
4. collection of cycles, each vertex v is connected to vertices $(v+1) \bmod n, (v+2) \bmod n, (v+3) \bmod n$, and
5. collection of 8-cliques.

The vertex set for these graphs is permuted randomly. Except for (3), the graphs have degree less than 8, so with 8 processors per node a good remapping should ideally map most communication to intra-node communication. For the graph (5) *all* communication can indeed be mapped to intra-node communication.

Experiments have been done on an NEC SX-6 with 4 nodes of 8 processors each. The results are shown in Tables 1 and 2. The maximum running time by a process is given in milliseconds, both for *topology creation* and for the actual *communication*, which is for 10 repetitions with 50000 integers sent per neighbor. For the *data redistribution* we have assumed 10 batches of 50000 integers; only the maximum redistribution time over all the topologies is reported. These counts are, of course, completely arbitrary, and only intended to give an impression of the relative costs.

We see that the extra overhead for creating a virtual topology with non-trivial remapping (here: 4-partitioning) is relatively small, between a factor 2 and 5. Topology creation time is only a small fraction (about 10%) of the resulting communication time, and the sum of creation time and resulting communication time is in all cases smaller than the communication time with the original communicator. The improvements in communication time are also large enough to outweigh the costs of data redistribution (with the parameters chosen here), except possibly for the 5×6 grid and the hypercube Cartesian topologies created over `MPI_COMM_WORLD`. Thus applications with a moderate communication load (amount of data, number of repetitions) can be expected to benefit from reordering. For 4-partitioning the topology creation time is more or less independent of whether we optimize for total communication cost (CC) or load balance (LB). For the Cartesian topologies optimal LB is the slightly more expensive criterion.

We give maximum communication time over all processes for the virtual topologies with (`reorder` set to 1) and without (`reorder` set to 0) process reranking. With reordering we

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

	Max time no `reorder`	Max time `reorder`	Optimization criterion
`MPI_Graph_create(MPI_COMM_WORLD, ..., reorder, &graphcomm)`			
Random graph (1), 144 edges			
Creation :	0.40	1.20	LB
Creation :	0.40	1.50	CC
Communication:	17.03	11.47	LB
Clustered graph (2), 141 edges			
Creation:	0.40	1.12	LB
Creation:	0.40	0.91	CC
Communication:	15.32	9.50	LB
Clustered graph (3), 273 edges			
Creation:	0.40	2.00	LB
Creation:	0.40	2.33	CC
Communication:	31.64	24.21	LB
Communication:	31.64	21.10	CC
Cycles graph (4), 192 edges			
Creation:	0.40	1.38	LB
Creation:	0.40	1.63	CC
Communication:	20.23	7.14	LB
Clique graph (5), 224 edges			
Creation:	0.40	0.96	LB
Creation:	0.40	0.82	CC
Communication:	24.72	3.41	LB
Maximum Redistribution:	0.00	3.57	LB/CC

Table 1: Graph topologies. Running times (in milliseconds) on a 4 node, 32 processor NEC SX-6. Creation time for setting up the virtual topology, without and with remapping, and resulting communication times. Optimization criterion is either *total communication cost* (CC) or *optimal load balance* (LB).

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

basecomm:	Max time no `reorder` random	Max time `reorder` random	Max time no `reorder` `MPI_COMM_WORLD`	Max time `reorder` `MPI_COMM_WORLD`	Optimization criterion
`MPI_Cart_create(basecomm, ..., reorder, &cartcomm)`					
2×16 Grid					
Creation:	0.30	0.75	0.30	0.77	LB
Creation:	0.30	0.64	0.30	0.67	CC
Communication:	15.16	3.33	8.56	3.30	LB
3×10 Grid					
Creation:	0.30	0.95	0.30	0.98	LB
Creation:	0.30	0.79	0.30	0.88	CC
Communication:	14.72	6.52	11.10	6.22	LB
4×8 Grid					
Creation:	0.30	1.19	0.30	1.23	LB
Creation:	0.30	0.94	0.30	1.02	CC
Communication:	14.00	7.15	9.13	7.18	LB
5×6 Grid					
Creation:	0.30	1.25	0.30	1.26	LB
Creation:	0.30	0.99	0.30	1.01	CC
Communication:	13.44	7.26	8.39	7.45	LB
$2 \times 2 \times 8$ Grid					
Creation:	0.30	1.35	0.30	1.30	LB
Creation:	0.30	0.95	0.30	0.98	CC
Communication:	21.70	15.09	17.82	14.52	LB
Communication:	21.70	10.24	17.82	10.32	CC
$3 \times 3 \times 3$ Grid					
Creation:	0.30	1.32	0.30	1.39	LB
Creation:	0.30	1.02	0.30	1.04	CC
Communication:	18.22	11.11	14.48	11.29	LB
2^5 Hypercube ($2 \times 2 \times 2 \times 2 \times 2$ Grid)					
Creation:	0.30	1.11	0.30	1.12	LB
Creation:	0.30	0.87	0.30	0.89	CC
Communication:	35.32	18.19	18.63	18.21	LB

Table 2: Cartesian topologies. Running times (in milliseconds) on a 4 node, 32 processor NEC SX-6. Creation time for setting up the virtual topology, without and with remapping, and resulting communication times. Optimization criterion is either *total communication cost* (CC) or *optimal load balance* (LB). For the Cartesian topologies `basecomm` is either a randomly reranked communicator or `MPI_COMM_WORLD`.

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

give results for load balance optimization (LB); only where a notable difference was observed, we also give the communication time for the reordering optimized for total communication cost (CC). With only 4 nodes, the difference between the two criteria is small, except for the Cartesian $2 \times 2 \times 8$ grid, where the CC criterion gives better communication time. A conclusion as to which is better is not warranted. For all virtual topologies we see a notable improvement in communication time. For the graph topologies, we note especially the improvement of a factor of almost 9 for the clique graph (5) – all communication is indeed mapped to intra-node communication. For the other graphs improvements are between a factor 1.5 and 3. For Cartesian topologies created over a random base communicator we observe similar improvements. As expected the trivial, block-wise process-to-processor mapping implicit in `MPI_COMM_WORLD` (where processes $0, \ldots, 7$ are mapped to node 0, processes $8, \ldots, 15$ to node 1, and so forth) provides a better Cartesian mapping than a random communicator, but for many of the virtual topologies the block-wise mapping is far from optimal. For instance, for the 2×16, 3×10 and for the 3-dimensional grids (very) worthwhile improvements over the running times with `MPI_COMM_WORLD` can be obtained by reordering; for the other two-dimensional grids and the hypercube, the process-to-processor mapping of `MPI_COMM_WORLD` cannot be improved much.

7 Concluding remarks

We have described a general framework for implementing the MPI topology mechanism on current hierarchical/clustered multiprocessor systems, and in particular shown that the general embedding problem can be solved more easily by graph partitioning. We stated the properties of a new Kernighan-Lin like heuristic for direct graph k-partitioning, which is used for the process reranking within MPI/SX. With R12.2 of MPI/SX the implementation described in this paper is available to users, and MPI/SX is thus one of very few MPI implementations with a non-trivial realization of the MPI topology mechanism. Experiments with simple benchmarks show considerable improvements in communication performance, and demonstrate the potential of the MPI topology mechanism for better utilization of parallel systems with a hierarchical communication system.

Acknowledgments

The author thanks Hubert Ritzdorf for continued discussions on MPI and much help with the concrete topology implementation. Also Bruce Hendrickson and Robert Preis are thanked for patient and helpful discussions on graph partitioning and the particular k-partitioning heuristic used in the implementation.

References

[1] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th ACM/IEEE Design Automation Conference (DAC)*, pages 175–181, 1982.

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**

[2] J. R. Gilbert and E. Zmijewski. A parallel graph partitioning algorithm for a message-passing multiprocessor. *International Journal of Parallel Programming*, 16(6):427–449, 1987.

[3] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI – The Complete Reference*, volume 2, The MPI Extensions. MIT Press, 1998.

[4] T. Hatazaki. Rank reordering strategy for MPI topology creation functions. In *5th European PVM/MPI User's Group Meeting*, volume 1497 of *Lecture Notes in Computer Science*, pages 188–195, 1998.

[5] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, 1995.

[6] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[7] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.

[8] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[9] J. E. Savage and M. G. Wloka. Parallelism in graph-partitioning. *Journal of Parallel and Distributed Computing*, 13:257–272, 1991.

[10] H. Simon and S.-H. Teng. How good is recursive bisection. *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[11] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The Complete Reference*, volume 1, The MPI Core. MIT Press, second edition, 1998.

[12] J. L. Träff. Direct graph k-partitioning with a Kernighan-Lin like heuristic. Manuscript (submitted for publication), 2001.

[13] C. Walshaw and M. Cross. Mesh partitioning: a multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.

[14] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 26(2):1635–1660, 2000.

Proceedings of the IEEE/ACM SC2002 Conference (SC'02)
0-7695-1524-X/02 $17.00 © 2002 **IEEE**