What MPI could (and cannot) do for Mesh-partitioning on Non-homogeneous Networks

Guntram Berti and Jesper Larsson Träff

C&C Research Laboratories, NEC Europe Ltd.
Rathausallee 10, D-53757 Sankt Augustin, Germany
{berti,traff}@ccrl-nece.de

Abstract. We discuss the *mesh-partitioning* load-balancing problem for non-homogeneous communication systems, and investigate whether the MPI *process topology functionality* can aid in solving the problem. An example kernel shows that specific communication patterns can benefit substantially from a non-trivial MPI topology implementation, achieving improvements beyond a factor of five for certain system configurations. Still, the topology functionality lacks expressivity to deal effectively with the mesh-partitioning problem. A mild extension to MPI is suggested, which, however, still cannot exclude possibly sub-optimal partitioning results. Solving instead the mesh-partitioning problem outside of MPI requires knowledge of the communication system. We discuss ways in which such could be provided by MPI in a portable way. Finally, we formulate and discuss a more general *affinity scheduling problem*.

1 Introduction

Applications involving large datasets are often parallelized using a data partitioning approach, as in mesh-based solution of partial differential equations. This leads to the following *mesh-partitioning problem*: A large mesh, represented as an undirected, weighted *problem graph* $G = (V, E, w)$ with edge (and possibly vertex) weights w is to be mapped onto a smaller set of processors P, such as to minimize application run time. This is approximated by minimizing communication costs, which are assumed to be a function of the value of the *edge cut* (sum of weights of edges in G crossing processor boundaries), while keeping computational load evenly distributed. Although this commonly used model is at best an approximation to the communication cost optimization problem (e.g. network contention is very hard to capture, communication volume may easily be overestimated, etc., see [4]), we will stick to it here.

Assuming a homogeneous, fully connected system, a common approach to solving the mesh-partitioning problem is to partition G into $|P|$ approximately equal-sized subsets, minimizing the value of the edge cut, i.e. finding a mapping $\pi : V \mapsto P$ such that

$$\sum_{\pi(u)\neq\pi(v)} w(u,v) \quad \text{is minimal} \tag{1}$$

under the *balancing condition* (which can be relaxed) that

$$|\pi^{-1}(p)| \leq \lceil |V|/|P| \rceil \tag{2}$$

This *graph partitioning problem* is NP-complete [3], but many good heuristics exist [2, 5, 8, 11], and are implemented in a number of libraries [7, 10, 16, 17].

Most of these algorithms can be extended to handle non-homogeneous *processor computing powers*, but it is more difficult to handle systems with non-homogeneous *communication systems*, for instance with a mesh or torus topology, or with a hierarchical structure like clusters of SMP nodes. The simple graph partitioning approach is not adequate here, since vertices of G with "heavy" edges might end up on processors connected by "weak" communication links. There is obviously no way a graph partitioner can exclude this possibility without additional knowledge of the underlying system. Different approaches to tackling this problem have been proposed and discussed.

In [16] the authors model the communication system as a complete *host graph* $H = (P, C, c)$ with a cost function c on edges $(p_0, p_1) \in C$ reflecting the "cost" of communication between processors p_0 and p_1. Deriving the costs $c(p_i, p_j)$ is not straightforward and to some extent even application-dependent. The authors favor a *quadratic path length* (QPL) metric, leading to a *network cost matrix* (NCM) penalizing connections going over many hops of the *physical* network. The *mapping problem* is then defined as a generalization of the partitioning problem (1): Find $\pi : V \mapsto P$ such that

$$\sum_{\pi(u) \neq \pi(v)} w(u, v) c(\pi(u), \pi(v)) \quad \text{is minimal} \tag{3}$$

subject to the balancing condition (2).

To solve this problem, they extend their homogeneous multi-level heuristic by mapping the coarsest problem graph to the host graph in an approximately optimal way (the exact solution is equivalent to the *quadratic assignment problem* and again NP-complete). The Kernighan-Lin heuristic used to derive partitions of the finer problem graph levels is modified to take the modified cost function and the resulting larger set of potential moves into account.

The same model of the communication system is used in [9]. However, they first start with a complete conventional partitioning, and then use the host graph to guide an incremental improvement of the partitioning. Still other heuristics for solving the mapping problem were given in [5, 6, 10]. In contrast, the *Dynamic Resource Utilization Model* (DRUM) [1] uses measurements to derive a hierarchical scalar characterization of compute nodes, merging both computing power and network bandwidth into a single "power" value per node. As the hierarchy is explicit in the abstract model, general-purpose partitioners such as Zoltan [17] can be instrumented to use specific partitioning strategies at each level [1].

In all cases cited above, the description of the hardware architecture and the related network performance parameters have to be set up manually. The network models discussed so far represent compromises, aiming to be simple enough for the underlying optimization approach. The NCM ignores hierarchical

structures, thus excluding level-specific partitioning. On the other hand, the averaging DRUM model loses some fine-grained local structure and seems less suited for e.g. mesh architectures.

Applications requiring graph partitioning are frequently using the *Message-Passing Interface* (MPI) for process communication. MPI, being strictly a communication interface, has no functionality for solving the mesh-partitioning problem. Since the MPI interface has no notion of "cost" of communication, MPI also cannot supply the knowledge of the underlying system required to construct the weighted host graph needed by a partitioning/mapping package. However, the internal assumptions about the underlying system present in any MPI implementation could potentially be made useful to solve the mesh-partitioning problem. This could be done either *implicitly* via the *graph topology functionality* of MPI, using the two-stage approach to the mesh-partitioning problem discussed and evaluated in sections 2 and 3. An orthogonal solution, discussed in Section 4, is to make the assumptions of the MPI implementation *explicitly* accessible in an abstract, portable and non-constraining fashion, to be used to construct the desired host graph for a mapping package. Finally, in Section 5, we take a broader view and ask if graph partitioning does not solve a too narrow problem altogether.

2 Mesh-partitioning with MPI process topologies

Although not capable of solving the mesh-partitioning problem, MPI defines functionality to solve a *process re-mapping problem* that could be used as the second stage in a *two-stage approach*: first partition the mesh into $|P|$ subsets V_i, $i = 0, \ldots |P| - 1$ assuming a homogeneous communication system, second find an optimal mapping of the $|P|$ subsets onto the set of processors.

The *graph topology functionality* of MPI [12, Chapter 6] makes it possible to specify a *non-weighted communication graph*, abstracting the communication pattern of the $|P|$ processes. The MPI implementation in turn can use this information to create a new communicator representing a process remapping which is best suited for the given communication graph on the given system. It is up to the MPI implementation to provide a suitable remapping (which could be just the identity mapping). While the two-stage approach has often been discussed, e.g. in [16], using the MPI topology functionality for the second process remapping step has apparently not been considered previously.

Assuming that the MPI implementation at hand has a non-trivial implementation of the topology functionality, the problem arises how to specify the communication graph of the $|P|$ processes. Putting an edge between two processes whenever there is an edge in G between two partitions is likely to lead to a communication graph overstating weak connections, possibly to the point of being a complete graph without information. Using edge weights corresponding to communication load between partitions would be an informative alternative, but is unfortunately not permitted by the MPI functionality. Instead, an edge

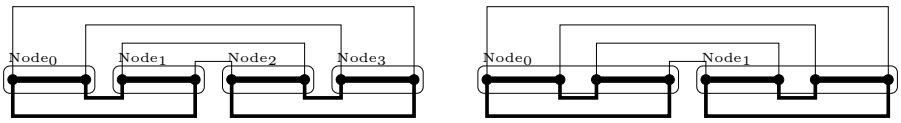

Fig. 1. Algorithm with hybercubic communication pattern. Communication intensity and/or volume decrease with increasing hypercube dimension. Heavier edges denote heavier communication. Left: optimal mapping of the algorithm onto a $2+2+2+2$ processor SMP cluster. Right: optimal mapping onto a $4+4$ processor cluster.

could be put if the total weight of edges between two partitions exceeds a certain *threshold*.

As the examples below will show both the threshold solution and the two-stage approach itself have limitations. We assume an SMP system with a marked difference in communication performance between processes on the same vs. on different SMP nodes. Similar examples can be constructed for systems with other, non-homogeneous interconnects.

The first example shows that unweighted graphs and thresholds are too weak to enforce an optimal mapping, unless complete knowledge of the underlying system is available, thus defying the idea of a portable, system-independent solution to the mesh-partitioning problem: Selecting the correct threshold *a priori* without knowledge of the target system configuration is not possible.

Example 1. Consider a hypercube algorithm with strong communication along dimension 0, less strong along dimension 1, etc. that we want to map onto an SMP system. Clearly, the processors should be mapped such that as many of the lower-dimensional, heavily communicating edges are inside SMP nodes, with higher-dimensional, weaklier communicating edges between nodes, cf. Figure 1.

Consider first the two-dimensional case of 4 processes to be mapped onto a $2+2$ processor cluster. Selecting a threshold resulting in edges along dimension 0 only, would make it possible for the MPI implementation to place pairs of connected processors on the same node, such that the heaviest communication takes place inside SMP nodes. On the other hand, selecting a lower threshold and having edges both along dimension 0 and dimension 1 would make it impossible for the MPI implementation to make the right decision since each process would be marked as communicating with two other processes.

Moving to three dimensions, for a $4+4$ processor cluster the best threshold would put edges along dimension 0 and 1. For a $2+2+2+2$ processor cluster the best threshold would put edges only along dimension 0. □

The problem is aggravated for systems with more than two layers of communication. In such cases even *with* knowledge of the underlying system, it is in general not possible with an unweighted graph to provide enough information to the MPI implementation to permit an optimal solution.

Hierarchical structures are found e.g. in multi-physics codes, where coupling within the "single-physics" cores occurs much more often than across sub-problem boundaries. If such problems are part of a larger application, we get

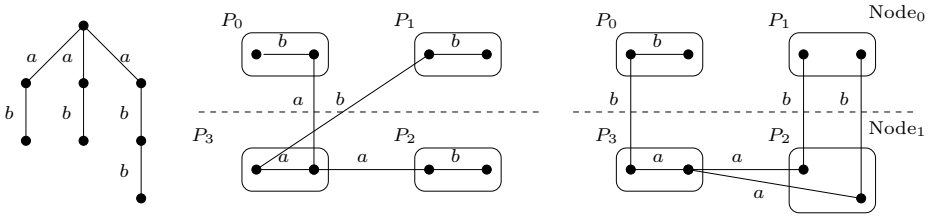

Fig. 2. Weighted 8 node tree (left). The optimal partition (middle) has cut weight $2a+b$. No matter how the 4 sets of this partition are mapped onto the $2+2$ processors, at least one a-edge crosses SMP nodes. A worse partition with cut weight $2a+3b$ (right) can be mapped such that the weight of edges crossing SMP nodes is only $3b$.

yet another weaker level of coupling. Concrete examples are found in climate research, where different models are coupled to achieve a more comprehensive global model [15]. For instance, we may have 3D/3D coupling of flow and chemistry components for both air and ocean, each coupling to their respective spatial neighbor partitions and a coupling occurring with lower frequency via the ocean/air interface.

A natural modification of the MPI graph topology mechanism would be to allow weighted graphs to model the intensity of communication along the edges. As the next examples show, the two-stage approach is strictly weaker than a direct solution of the processor mapping problem: A graph partitioner without knowledge of the target system (as modeled by the host graph) cannot compute the most suitable partition for the system.

Example 2. We consider a weighted tree of 8 nodes as shown in Figure 2. There are two different edge weights a and b with $a > b$. The minimum cut partition has cut weight $2a + b$ but is not optimal for mapping onto a $2 + 2$ processor SMP cluster, since at best an a and a b edge cross between nodes. Instead, the suboptimal partition with cut weight $2a + 3b$ is better suited, since the weight of the edges between processes on different SMP nodes can be arranged to be only $3b$. For appropriate values of bandwidth and edge weights, the ratio in communication load between the two partitionings can become arbitrarily large. □

Example 3. Example 2 may seem artificial. Figure 3 shows that the two-stage approach can give arbitrarily bad results even for mesh-based graphs. □

From the examples two conclusions can be drawn:

1. The non-weighted MPI topology functionality does not provide enough information for optimal process remapping in case of different communication requirements between different processes. This could easily be remedied by allowing *weighted communication graphs* in the MPI functionality. This and other problems (lack of scalability, lack of control of optimization criterion, etc.) was discussed in [14].

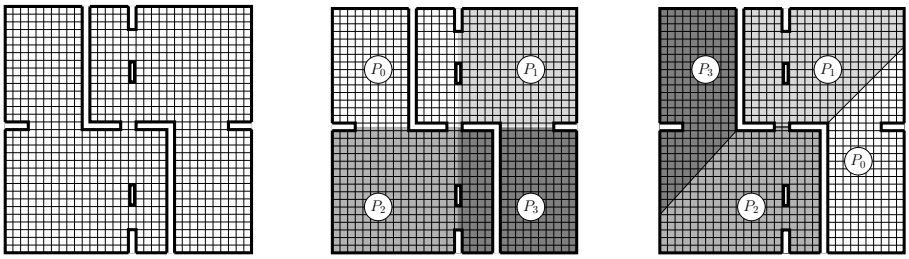

Fig. 3. Left: "Jagged" mesh. Middle: Homogeneous optimal 4-way partitioning. Right: Optimal partitioning for a 2+2 SMP cluster $\{P_0, P_1\}, \{P_2, P_3\}$, having minimal coupling between both SMP nodes.

2. Even with weighted graphs the two-stage approach to mesh-partitioning may deliver arbitrarily bad solutions. In [16], the two-stage approach is shown to be worse by a factor of about 2-3 on average for benchmark meshes on cluster architectures, using weighted edge cut as measurement (no actual timings are given).

Even though the two-stage approach is inferior to a direct solution, it can be a viable and user-friendly option in cases where the partitioning of the problem is fixed, as the next section will show.

3 An application kernel

To illustrate the possible performance benefits achievable by mesh partitioning using the (theoretically sub-optimal) two-stage approach with the final process remapping done by the MPI topology functionality, we consider a *communication kernel* with the hypercube communication pattern described in Example 1. This pattern is assumed to be the outcome of the first stage mesh partitioning, and the second stage consists in a process remapping to fit the target system. This is carried out by defining a communication graph which can be input to MPI to perform the appropriate process remapping. The kernel is written such that the process to processor mapping implied by `MPI_COMM_WORLD` (where MPI processes are distributed consecutively in increasing MPI rank order over the SMP nodes) is unsuited for SMP systems: the most frequent communication will be between processes on different nodes. In the kernel the communication frequency along hypercube edges increases exponentially with decreasing dimension of the edge. Increase factor as well as size of the data sent along the dimensions can be varied, but will not be of concern here.

In order to gain any effect a non-trivial implementation of the MPI graph topology functionality is required. This is fulfilled by MPI/SX [13], and the measurements shown in Table 1 have been conducted on a four node, 32 processor NEC SX-8 system. The difference in communication bandwidth between processes on the same SMP node and processes on two different nodes is about

Processes	Distribution	WORLD	random	topo[1]	topo[2]	topo[3]	topo[4]	topo[5]
8	8	5632	5620	5627	5958	5639		
	$4+4$	21021	22399	6653	**6517**	20987		
	$2+2+2+2$	18402	20379	6223	**6221**	18970		
16	$8+8$	321742	223179	66703	**55329**	68053	321769	
	$4+4+4+4$	221754	185305	62101	**51588**	65404	222387	
	$1+7+1+7$	291283	212762	166729	**160646**	166331	291008	
	$2+6+2+6$	265642	225786	**65491**	159097	65844	295374	
	$3+5+3+5$	239661	221891	170463	**157388**	164895	271440	
	$8+4+4$	320393	218326	68943	**53494**	66942	320363	
32	$8+8+8+8$	1090388	1197901	460068	**228929**	251979	532812	1093170

Table 1. Running time (in micro-seconds) for the hypercube kernel on an NEC SX-8. The first two columns describe the SMP configuration, and the remainder give the running time on various MPI process distributions: `MPI_COMM_WORLD` communicator, a `random` communicator, and communicators created by the topology functionality. Here `topo[`i`]` denotes a communication graph with edges along hypercube dimensions $0, \ldots, i-1$. The largest improvements over `MPI_COMM_WORLD` the distribution is shown in bold, and ranges from a factor of two to a factor of more than 5.

a factor of two, but more importantly, if several processes on a node attempt to communicate with processes on other nodes at the same time, the communication is serialized. Thus, mapping heavily communicating processes to the same node is doubly beneficial. As detailed in Example 1, in the absence of edge weights in the MPI topology functionality, graph edges must be chosen to reflect the SMP system. Too few edges (e.g. only along hypercube dimension 0) can give sub-optimal improvement, and too many edges makes too many processes indistinguishable such that a good remapping cannot be guaranteed.

Table 1 gives some results of running the kernel on various number of processes and distributions over the SMP nodes. In each case good results are achieved when each subcube of the communication graph fits onto one SMP node. Bad results are generally achieved when the subcubes are too large for the SMP nodes (e.g. column `topo[4]`, corresponding to communication graphs with 16 process subcubes to be mapped onto 8 process SMP nodes), and good or even best results are achieved with smaller sized subcubes than the size of the SMP nodes (most of the best results are in column `topo[2]`). The best overall improvements exceed a factor 5 for distributions with 16 and 32 processes.

4 Portable MPI topology introspection

As shown, using the MPI topology functionality to solve the mesh-partitioning problem has inherent limitations, even if weighted graphs would be allowed. The alternative is to do the mesh-partitioning completely outside of MPI. This requires information on the communication system, either *a priori*, by measuring, or both. Measuring alone is of limited value on a loaded system, and in general

has difficulties capturing effects of contention (cf. end of Section 5 for possible solutions).

Instead, we propose to leverage the implicit knowledge on the system which is present in any MPI implementation (no matter how rudimentary). For an MPI *communicator* we model the part of the communication system used by the processes in the communicator as a complete graph with multiple weight functions. Nodes correspond to processes, with edge weights modeling either the number of abstract *hops* between two processes, or the *relative bandwidth*, or *relative latency*, ... A hop measures the number of communication layers between two processes. Processes on the same node of an SMP cluster would be one hop distant (the number of hops from a process to itself being 0), and processes on different node would be two hops distant. In a 1D linear array, each process (in `MPI_COMM_WORLD`) has two neighbors which are one hop away, two neighbors that are two hops away and so on.

We believe that it is possible to make this abstract representation available to an MPI application (e.g. mesh-partitioner) in a meaningful and portable way, regardless of the actual system. We suggest the following functionality.

- Functions returning the number of neighbors of the calling process that are exactly n hops away, the list of such neighbors, and the maximal hop distance to any other process in the communicator.
- Functions returning the hop distance, relative bandwidth and relative latency between the calling process and any other process in the communicator. Relative bandwidth, e.g., could be expressed as the ratio to the bandwidth for the process communicating with itself. This issue is bound to be contentious.
- A function returning the maximum number of simultaneous communication operations to processes at a given hop distance.
- Possibly more involved functions for estimating the effects of contention, e.g. returning the load of the communication path between the calling process and any other process in the communicator, given that (a) the two processes are the only processes communicating, (b) the load under the worst bisection with the two processes belonging to different parts.

These proposals are portable in the sense that each processor is only required to be able to return information about its own neighborhood (relative to the given communicator). A trivial implementation is possible, and would map all processes as being one hop away.

This functionality would clearly make it possible to build the concrete graphs as used in the NCM approach [16], or to construct hierarchical models like DRUM, as well as other imaginable representations. For instance, a hierarchical graph can be built by an application as follows:

1. Get all neighbors with hop distance 1,
2. Compute local graph components
3. While the graph is not connected:
   (a) introduce a new hierarchical node for the current component
   (b) Get all neighbor graph components for the next larger hop distance n
   (c) Compute the resulting larger components

5 Beyond partitioning: Affinity scheduling

The discussion so far assumed that a host graph $H = (P, C)$ is given. In general, however, H is only a subgraph of the (available part of the) global machine graph H^G, and is selected by a system scheduler, typically based on the number of compute nodes specified by the user. This places the burden of specifying an adequate subset of the machine on both the user (who may not know about it), and the scheduler (who does not know about the application). Giving the scheduler more knowledge about the resource requirements of an application, it could choose an optimal subset of the machine matching high-level user preferences:

- The user could demand just enough processors to finish within one hour
- The application is partitioned into largely independent tasks and can therefore be distributed to weakly connected nodes
- It is found that the application will not achieve good parallel performance on the currently available set of nodes, and it is scheduled for a later time

These tasks cannot by solved in the narrow frame of graph mapping. Instead, we propose to consider the following *optimal subgraph scheduling problem*: Given the time dependent *global machine graph* $H^G(t) = (P^G(t), C^G(t)),\ t > 0$ ("free processors at time t"), a *utilization cost function* $K = K(P, \tau, t)$ (cost for using processor set $P \subset P^G(t)$ for duration interval $[t, t + \tau]$), and a *user preference function* $\Phi = \Phi(K, t)$ (preference of finishing the task until time t with total cost K), find a *starting time* t_0 and a mapping $\pi : V \mapsto P^G(t_0)$ such that

$$\Phi\left(K(\pi(V), t_0, T_{app}), t_0 + T_{app}\right) \quad \text{is minimized} \quad (T_{app} = T_{app}(\pi(V)) \tag{4}$$

Here, the total (expected) time T_{app} is an application-specific performance estimation based on the partitioning and the available network (sub)topology. In (4), a hidden constraint is that the subgraph $H^G(t)$ must be available for all times $t = t_0 + \tau, 0 \le \tau \le T_{app}(\pi(V))$.

For homogeneous architectures, $K(P, \tau, t) = \alpha P \tau$ and $\Phi(K, T) = KT$ would be reasonable choices. Changing Φ, a user could slant the result in favor of cheaper or faster computation. Information about $H^G(t)$ is generally available only in the system scheduler, thus, a solution to problem (4) would have to access this information. Using scheduler information together with actual network measurements might also permit to estimate the bandwidth available to new applications, thus combining the advantages of static and dynamic network information.

6 Summary

We investigated two orthogonal paths to solving the mesh-partitioning problem for systems with a non-homogeneous communication system. A two-stage approach, consisting of ordinary graph partitioning followed by a remapping relying on the MPI topology functionality, and probably requiring the least change

on behalf of the application, is limited by the restriction to non-weighted communication graphs of the MPI standard. As an orthogonal approach, we discussed additional, portable, system-independent MPI functionality, which could aid the application programmer in constructing the desired graph model of the system to be used as input to sophisticated mesh-partitioners. An artificial, but not unrealistic kernel showed the large potential gains by performing an appropriate process mapping.

References

1. K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. *Appl. Numer. Math.*, 52(2–3):133–152, 2005.
2. 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.
3. M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.
4. B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519–1534, 2000.
5. B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, 1995.
6. B. Hendrickson, R. Leland, and R. V. Driessche. Skewed graph partitioning. In *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.
7. G. Karypis. METIS. `http://www-users.cs.umn.edu/~karypis/metis/`.
8. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.
9. I. Moulitsas and G. Karypis. Architecture aware partitioning algorithms. Technical Report DCT Research Report 2006/02, Digital Technology Center, University of Minnesota, Jan. 2006.
10. F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking (HPCN), Europe*, volume 1067 of *Lecture Notes in Computer Science*, pages 493–498. Springer-Verlag, 1996.
11. J. E. Savage and M. G. Wloka. Parallelism in graph-partitioning. *Journal of Parallel and Distributed Computing*, 13:257–272, 1991.
12. 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.
13. J. L. Träff. Implementing the MPI process topology mechanism. In *Supercomputing*, 2002. `http://www.sc-2002.org/paperpdfs/pap.pap122.pdf`.
14. J. L. Träff. SMP-aware message passing programming. In *Eigth International Workshop on High-level Parallel Programming Models and Supportive Environments (HIPS03)*, pages 56–65, 2003.
15. S. Valcke, D. Declat, R. Redler, H. Ritzdorf, R. Vogelsang, and P. Bourcier. The PRISM coupling and I/O system. In *Proceedings of VECPAR 2004*, 2004.
16. C. Walshaw and M. Cross. Multilevel mesh partitioning for heterogeneous communication networks. *Future Generation Comput. Syst.*, 17(5):601–623, 2001.
17. Zoltan: Data-management services for parallel applications. `http://www.cs.sandia.gov/Zoltan/`.