Topology Aware Process Mapping

Sebastian von Alfthan[1], Ilja Honkonen[1,2], and Minna Palmroth[1]

[1] Finnish Meteorological Institute, Finland
[2] Department of Physics, University of Helsinki, Finland
sebastian.von.alfthan@fmi.fi

Abstract. A parallel program based on the Message Passing Interface (MPI) commonly uses point-to-point communication for updating data between processes, and its scalability is ultimately limited by communication costs. To minimize these costs we have developed a library that reduces network congestion, and thus improves performance, by optimizing the placement of processes onto nodes allocated to the parallel job. Our approach is useful on production machines, as irregular communication patterns can at run-time be optimally placed on non-contiguous node allocations. It is also portable as it supports multiple architectures: Cray XT, IBM BlueGene/P and regular SMP clusters. We demonstrate on a Cray XT5m and an Infiniband cluster that good placement of processes doubles the total bandwidth compared to random placement and, furthermore, by up to a factor of 1.4 compared to to the original placement. It is not only important to place processes well on individual nodes, minimizing the number of link traversals on the Cray XT5m provides up to 20 % of additional performance. The scalability of a real-world application, Vlasiator, is also investigated and the scalability is shown to improve by up to 35 %. For communication limited applications the approach provides an avenue to improve performance, and is useful even with dynamic load balancing as the placement is optimized at run-time.

Introduction

In high performance computing good performance and scalability is of great importance to maximize scientific output. In order to achieve good parallel efficiency on a distributed memory computer it is important to maximize the fraction of total time a program spends on computing. In addition to computing time, parallel applications also spend time in transferring data and waiting in synchronization points, which does not directly help to advance the simulation. For a given implementation of a parallel algorithm based on domain decomposition, synchronization and communication costs can be minimized through load balancing. Load balancing with widely used libraries such as Zoltan [1] does no take into account the actual network topology of the machine, and thus processes handling neighboring subdomains can be far away from each other on the actual hardware. This can lead to higher communication and synchronization cost due to network contention as multiple messages have to cross the same network links, and due to higher latencies as messages have to hop over several network links. Optimizing the placement of subdomains to the compute units of a machine with a particular network topology has been studied extensively over the years, both based on physical optimization algorithms for mapping subdomains onto compute units, as well as in heuristic approaches. The problem itself is NP-hard [2], but to realize performance improvements only nearly optimal solutions are required. Bhanot et al. [3] introduced a off-line simulated annealing approach for optimizing placement on IBM Blue Gene/L supercomputers. More recently Bhatele et. al [4] developed a set of heuristics for mapping communication graphs onto Cray XT and IBM Blue Gene machines. The heuristic approach is not easily generalized; applications often exhibit irregular communication patterns and are allocated irregular computational sets of nodes. The MPIPP tool [5] is able to both discover the topology at run-time through ping-pong tests, as well as optimize the mapping problem using a K-way graph partitioning algorithm. In another recent set of publications Mercier and Jeannot [6,7] introduce the well performing TreeMatch algorithm for optimizing placement on multi-node NUMA architectures, and achieved good results when optimizing for data volume. The main limitation in this work is that the physical topology of the network is considered flat. Subramoni et. al designed [8] a scalable network topology detection service for InfiniBand networks, and a topology aware MPI library that takes advantage of this information.

Vlasiator [9] is a new simulation code where good scalability is a key requirement for simulating space plasma, both in local setups and on a global scale where space-weather, namely the interaction between the highly varying solar wind and Earth's magnetosphere, is simulated. This system is of great interest as near-Earth space provides a extremely rich environment for studying a wide range of plasma phenomena unreachable in laboratories. By combining simulation results with measurements from satellites and ground-based instruments, new fundamental insights may be reached. There are also practical interests, as events such as solar flares and coronal mass ejections may cause space storms disturbing the operation of satellites, GPS signals, and even electric grids and other long conductors on earth through geomagnetically induced currents.

To investigate the impact of process mapping and to improve the scalability of Vlasiator, we have developed a library that is able to optimize the placement of subdomains obtained from a partitioning algorithm onto compute nodes connected by a network. The hypothesis is that one can improve performance of parallel applications by improving the placement of processes in real-world supercomputers. To support this library has to be adaptable; both regular and irregular point-to-point communication patterns in applications should be described and mapped to irregular resource allocations. It also has to be portable; network topologies describing different machines should be automatically discovered. Finally, the optimization cost has to be reasonable so that it can be done repeatedly while the simulation is running and the load balancing changes. The other hypothesis is that one can gain additional speedup by also utilizing information on the actual network topology, but improvements are also seen by optimizing for compact placement on nodes.

Here we present the approach we have developed to tackle these challenges. We benchmark our library using a simple test case with nearest neighbor communication on two contemporary supercomputers. We also apply the approach to Vlasiator.

Optimizing the Topology Mapping

The approach taken here is geared towards minimizing communication time for an application where the majority of the communication is sparse, and each process only exchanges data with a few other processes. This situation occurs quite frequently, e.g., in applications where the solution of a numerical model at a certain location in a problem domain only requires local information to be solved. Such problems are typically parallelized using a domain decomposition approach, where each process solves the problem on a subdomain of the total problem domain. At each iteration the required data along the subdomain boundary is exchanged with the processes of the neighboring subdomains. If the processes of neighboring subdomains are distant on the actual machine network topology, the latency and bandwidth may degrade compared with the situation where they are on the same node, or on nodes close on the network topology. The domain is split into subdomains such that the computational load is equally divided using a load balancing scheme. These schemes do not take into account the network topology of the machine, and thus an imbalance in communication performance may still remain, even if the computational load is perfectly balanced. In problems with a static and regular domain decomposition it may be possible to place processes close on the network, but when the problem is irregular and dynamic, optimal placement requires a more involved scheme, such as the one described here.

The communication pattern of the application is described by a directed graph, where each vertex corresponds to a subdomain of the problem, and each directed edge corresponds to a send operation that transfers boundary data to a neighboring subdomain. Typically the edge weight $w_c(u_c, v_c)$ is the amount of data sent from the tail vertex u_c to the head vertex v_c. This communication graph is specified by the user in the application; each process adds its own relevant send operations and their weights.

The network topology of the machine is described by a complete graph. Each vertex corresponds to a compute unit, and each edge weight $w_n(u_n, v_n)$ describes the cost of sending data between the compute units u_n and v_n. This network topology is automatically discovered by the library; we have support for Cray XT and IBM BlueGene/P where the core location on the torus is known (torus-topology) and clusters where we only know which cores are on the same node (node-topology). The node coordinate on the Cray XT torus can be read using an undocumented system API [4]. For Cray XT and IBM BlueGene/P the weight of an edge between two processes on the same node is zero, and the weight between two processes on different nodes is the square root of the rectilinear distance of the two nodes. This is a heuristic choice that proved to give better results than a pure hop-byte metric [4] that is implied by having the weight being equal to the rectilinear distance. To model clusters we set the weight of inter-node edges to zero and intra-node edges to one, as we have no specific information on the network topology of the machine. This means that placement is optimized to minimize communication between nodes. Explicitly storing each weight would be unfeasible at scale as the storage requirement would scale as the square of the number of processes; instead we recompute the weights when needed. Earlier work has succeeded in measuring the network topology through ping-pong tests [5], or by directly discovering the InfiniBand topology [8]. In this work ping-pong tests were also briefly attempted, but on the fat-tree InfiniBand network and the Cray XT torus network used here, the measurement were dominated by noise and were not useful for constructing the network topology.

A mapping $v_n = M(v_c)$ of the communication graph onto the network topology graph describes on which core (network graph vertex), each subdomain (application communication vertex) is located. The cost of one communication edge is obtained by multiplying the weight of the communication edge with the network edge weight between the two processes to which the subdomains map. The total cost E of the mapping is the total cost of all communication edges averaged over all communication vertices,

$$E = \frac{1}{|V_c|} \sum_{\{u_c v_c\} \in E_c} w_c(u_c, v_c) w_n(M(u_c) M(v_c)), \tag{1}$$

where V_c is the set of communication vertices and E_c is the set of communication edges.

To optimize the mapping we use a standard simulated annealing algorithm. In this stochastic optimization scheme the mapping is iteratively modified by doing trial steps, which are accepted with a probability given by the Metropolis-Hastings acceptance probability function,

$$p(E_i, E_{i+1}, T_i) = \exp\left(-\frac{E_{i+1} - E_i}{T_i}\right), \tag{2}$$

where E_{i+1} is the cost of the mapping after the trial step, E_i is the cost of the current mapping, and T_i is a scaling factor corresponding to a temperature. If the trial step is rejected, the state is not modified. If it is accepted, the state of the trial step becomes the new state. The trial steps randomly pick two subdomains, and swap the processes onto which they are mapped. The cooling schedule defines how T_i depends on the time step i. In general, the optimization starts by randomizing the mapping, followed by an anneal starting from a high temperature where a large fraction of all steps are accepted, down to a low temperature with few accepted steps.

To get a good coverage of the phase space of the optimization problem we parallelize the algorithm so that each process independently optimizes the mapping using different cooling schedules. First the initial temperatures of each process is automatically tuned to achieve a desired acceptance rate, ranging from 0.1 on process 0, up to 0.5 on the last process. Then the cooling proceeds by decreasing the temperature every M steps by a process dependent factor selected randomly from the range of 0.975 to 0.995, until the temperature has been reduced 1000 times. We elected to scale M linearly with the number of processes N, so that the total number of states sampled in the optimization is proportional to N^2 indicating a good coverage of phase space. This also implies that the optimization time scales as $O(N)$. When testing various cooling schedules a value of $M = 4N$ was found to provide a good match between performance and quality for the cases studied here. At lower values the mapping had a higher cost, while slower anneals provided only marginal improvement.

Implementation

The library supports C++ applications, and comes in the form of a set of header files describing three kinds of classes: a communication graph describing the send operations of the application, a network graph describing the machine and a mapping class that can optimize the placement of communication vertices onto network vertices. The machine, and its queuing system does not need to provide any special support for our approach. The actual migration of the work has to be done by the application, based on the optimized mapping.

The communication graph is described by a graph class, and the vertices and edges and their weights are set by the application. First each process adds a vertex corresponding to its rank using the `addVertex` function in the graph, then each process adds its communication vertices (send operations) by setting receiving processes and the corresponding communication weights through the `addEdge` function. Finally the actual graph is constructed using the collective `commitChanges` call, after which all processes will have a complete copy of the communication graph. For example, if each process has two neighbors to which it sends data then one could initialize the communication graph for all processes in the `MPI_COMM_WORLD` communicator using the following piece of code.

```
Graph g(MPI_COMM_WORLD);
g.addVertex(rank);
g.addEdge(rank,neighbor1,weight1);
g.addEdge(rank,neighbor2,weight2);
g.commitChanges();
```

For each supported architecture there is a network graph class derived from the graph class. The constructor of the class will create a graph with a vertex for each compute unit allocated to the job, and will index these by the rank of the process located on the compute unit. As for the communication graph, each process will construct a complete copy of the whole network graph. For general clusters where only on-node placement is optimized the nodes are identified using `MPI_Get_processor_name`, and the graph describing the network is constructed as follows:

```
NodeNetwork t(MPI_COMM_WORLD);
```

On a Cray XT machine system libraries are used to discover the position of each process on the network, but the actual torus size has to be given by the user. For example, to initialize a torus with a size of $1 \times 12 \times 16$ the following piece of code is used:

```
CrayXtTorus t(1,12,16,MPI_COMM_WORLD);
```

The mapping class is used to optimize the placement, and its constructor takes as arguments the communication graph, the network graph and the communicator for which the graphs are defined. The mapping is then optimized using a call to the optimization function. After optimizing the mapping, one can read to which network vertex (i.e. rank) the subdomain on a rank should be transferred using the `getNetworkVertex` function. The actual exchange of subdomains has to be handled by the application. The following shows an example of optimizing the mapping:

```
Mapping<Graph,CrayXtTorus> m(g,t,MPI_COMM_WORLD);
m.simulatedAnnealingOptimizer();
destinationRank=m.getNetworkVertex(rank);
```

Results

We have tested the approach on two machines: Curie and Meteo. Curie is a Bull Bullx InfiniBand cluster based in France at Commissariat à l'énergie atomique et aux énergies alternatives (CEA). On the fat-node partition of Curie that was used for this work there are four eight-core processors per node, connected with a quad data rate InfiniBand network with a fat tree topology. Meteo is a Cray XT5m machine based in Finland at the Finnish Meteorological Institute. The SeaStar2 network is a 2D torus spanning 2 cabinets, and each node comprises two six-core processors.

On Curie we used the node-topology to model edge weights. This enables a optimized placement of simulation subdomains on nodes, minimizing network traffic. On a machine such as Curie, with a large number (32) of cores per node the network is potentially a significant bottleneck. As we have no in-depth knowledge of the network, we were not able to model the network itself in more detail. On Meteo we used both the same node-topology that minimize traffic sent from the node, as well as the torus-topology model that also reduce the number of hops that messages traverse over the network as subdomains exchanging data are placed closer to each other on the network.

Results from two tests are presented here: a benchmark showing the effective bandwidth and a real-world application Vlasiator for simulating space plasma.

Bandwidth Test

The first test is a benchmark corresponding to the communication needs of a stencil operation on a regular three-dimensional Cartesian grid with periodic boundary conditions. We construct a grid using the DCCRG [10] library[1], so that each process is allocated exactly one cell with a constant amount of data per cell. This removes potential performance artifacts by as it provides perfect load balancing, and enables us to send point-to-point data between two processes in one message from a contiguous block of memory. In the communication phase each process sends the cell data to its 26 nearest neighbors, and also receive their cell data in return. To measure the performance of the data exchange we measure the total bandwidth achieved in this operation. The performance is measured for three mappings: 1) The default one, where the subdomains (cells) are placed according to their index, so that columns of cells in the grid are placed on the same nodes. 2) Optimized placement where the total mapping cost has been minimized. 3) Randomized placement of subdomains.

In Figure 1 we have plotted the bandwidth as a function of the message (cell) size for a case with 1024 and 2048 cores on Curie. The benefit from proper placement is clear; the optimized placement provides significantly better throughput. The default placement has 20-50 % better bandwidth than the random placement. The optimized node placement further increases performance by 20 % for messages with a few kilobytes, and by 40 % for larger messages of several megabytes in size. Compared to random placement performance is close to 100 % better, both for smaller messages of a up to a kilobyte as well as for large messages with several megabytes of data. This suggests that the optimized placement affects both latency and bandwidth sensitive communication patterns.

The performance increase is readily understandable as a decrease in the total network traffic. For randomized mapping most cells that exchange data will be placed on different nodes, and almost all messages have to traverse the network. In our tests each

[1] `http://gitorious.org/dccrg`

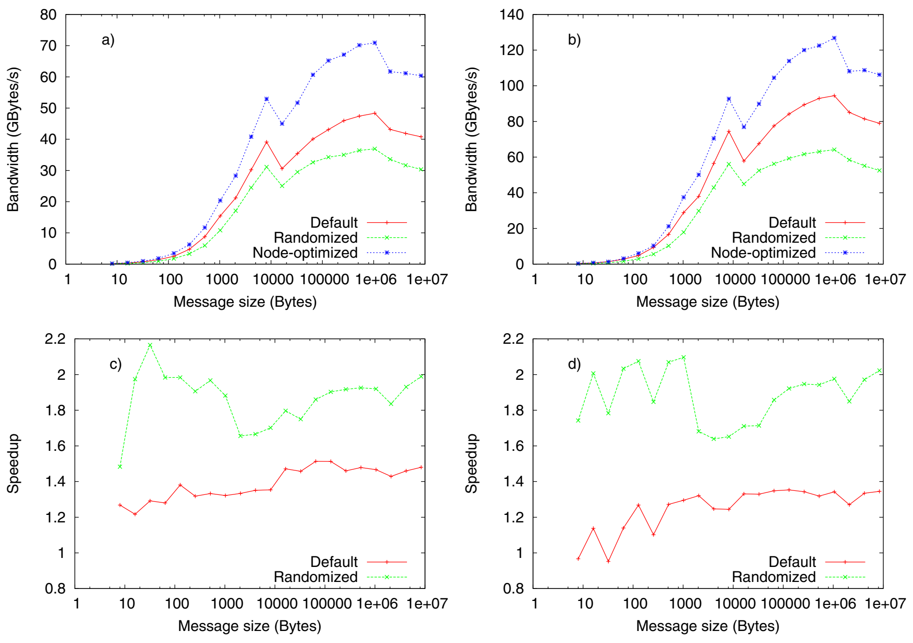

Fig. 1. Performance achieved using different mappings as a function of the size of each message on Curie, a quad data rate InfiniBand cluster with a fat tree topology. The total bandwidth is plotted for a) 1024 cores and b) 2048 cores. The speedup one can achieve by optimizing for the node-topology compared to random and default placement is also plotted for c) 1024 cores and d) 2048 cores.

process sent on average more than 25 out of a maximum of 26 messages over the network. The default placement is already better than the randomized one, as the column of cells assigned to a node is contiguous reducing the amount of messages to 21 in our tests. In the optimized placement the cells assigned to the processes on each node form a compact cluster, further decreasing the amount of data transferred over the network to on average 13.8 messages per process. The efficient on-node shared-memory MPI is thus able to handle nearly half of the messages, while the other half goes over the network. This corresponds closely to the doubling of bandwidth going from randomized placement to node-optimized placement.

Figure 2 shows the results on Meteo for 480 and 960 cores. The torus-topology optimized mapping shows up to 120 % better bandwidth than the random mapping and on the order of 30 % better bandwidth than the default mapping. If we only optimize for node placement the results are slightly worse, the added knowledge we have of the network gives us up to 20 % of additional performance.

Again the underlying reason for the performance increase is the reduced network traffic. In the default case on average 24 messages are sent over the network, showing that for the smaller nodes on Meteo the processes have neighbors on the same node only in one dimension of the grid. The randomized case again has close to the maximum

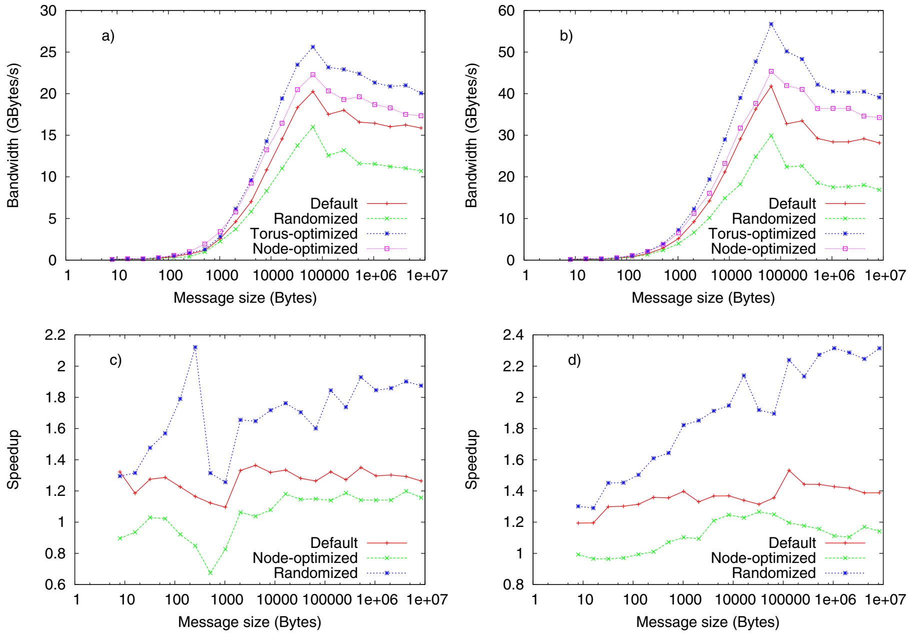

Fig. 2. Performance achieved using different mappings as a function of the size of each message on Meteo, a Cray XT5m machine with a SeaStar2 2D-torus network. The total bandwidth is plotted for a) 480 cores and b) 960 cores. The speedup achieved by optimizing for the torus-topology compared to random placement, default placement and placement optimized for the node-topology is plotted for c) 480 cores and d) 960 cores.

amount of messages sent over the network, with close to 25.7 messages being sent per process. As for the optimized case, only 18 cells on average need to be transmitted. This is higher than for Curie due to the fact that on Curie we can fit 32 cells (processes) per node, while on Meteo we can only fit 12 leading to a higher surface to volume ratio of the cluster of cells on each node.

Application Tests with Vlasiator

Vlasiator is based on the hybrid-Vlasov description, where ions are modeled as a 6-dimensional distribution function $f(\mathbf{r}, \mathbf{v}, t)$ in ordinary and velocity space, while electrons are modeled as a charge neutralizing massless fluid. The distribution function obeys the Vlasov equation,

$$\frac{\partial}{\partial t} f(\mathbf{r}, \mathbf{v}, t) + \mathbf{v} \cdot \nabla_r f(\mathbf{r}, \mathbf{v}, t) + \mathbf{a} \cdot \nabla_v f(\mathbf{r}, \mathbf{v}, t) = 0, \quad (3)$$

where $\mathbf{r}$ and $\mathbf{v}$ are the ordinary space and velocity space coordinates, acceleration $\mathbf{a}$ is given by the Lorentz force coupling the ion propagation to the electromagnetic fields, and $f(\mathbf{r}, \mathbf{v}, t)$ is the six-dimensional phase space density of ions with mass m and charge q. The distribution function is propagated forward in time with a finite volume method (FVM) method [11,12]. In the hybrid-Vlasov model electrons are not given a full Vlasov treatment, as that would be computationally too demanding for global simulations. Instead we couple the propagation of the ion distribution function to a field solver [13] for Maxwell's equation including Ohm's law, making self-consistent simulations possible. The ions couple to the field propagation through Ohm's law that includes the ion charge density and the bulk velocity from the zeroth and first moments of the velocity distribution.

We use DCCRG grid to construct a three-dimensional Cartesian mesh in the ordinary space, which is parallelized using a domain decomposition scheme with load balancing using the recursive coordinate bisection algorithm in the Zoltan [1] library. Each cell in ordinary space contains variables describing the electromagnetic field on its edges and faces, as well as a three-dimensional velocity mesh describing the full six-dimensional phase-space. In total the amount of data per cell is on the order of 1 MB, and a typical message size when sending data is 500 KB. Thus Vlasiator's main communication load is in the large-message range of the results from the bandwidth tests, where the benefit from the scheme was significant.

In Figure 3 the strong scalability of Vlasiator on Meteo is compared for two system sizes: The first has 27×27×1 cells in ordinary space, and the other has 54×54×1 cells. In the tests with Vlasiator we compared the default placement, with the one obtained by optimizing for torus-topology. The benefit from the placement optimization is most pronounced for larger core counts. For the large system we get at 1536 cores up to 35 % better performance through topology optimization. Contrary to the bandwidth benchmark, this test not only contains MPI communication, but also a significant portion of computation. At low process counts the compute time dominates, and improvements to MPI communication have little influence on the total performance. At higher process counts the MPI communication becomes dominant, as the amount of cells per process decreases. At low process count there is also a greater number of inner cells that can

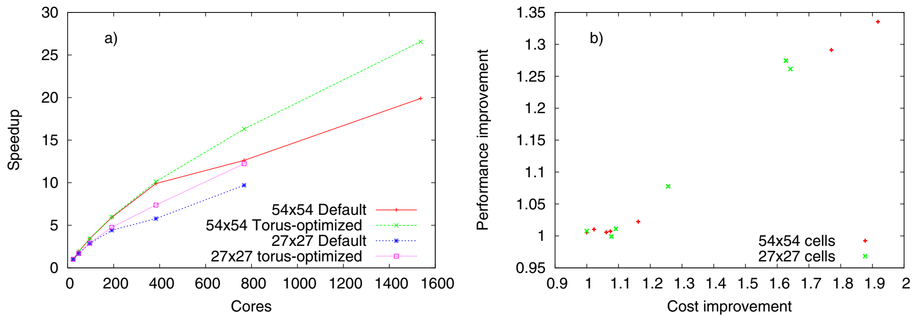

Fig. 3. The strong scalability of Vlasiator is shown in a) for two system sizes, using default placement as well as optimized placement for torus-topology. The speedup is computed relative to the performance on two nodes with 24 cores in total. In b) the correlation between the improvement in the cost of the optimized mapping and the improvement in the performance is plotted for the same two simulations.

be computed during communication, enabling one to better hide communication costs. Finally it is also evident from Figure 3 that the improvement in the cost closely correlates with the improvement in performance. Similar linear correlation also holds for the bandwidth tests.

Discussion

We have described the problem of minimizing MPI communication time of domain decomposed programs by optimizing the placement of subdomains on the physical hardware. This is done by mapping a graph representing the subdomains and their communication, to a graph representing cores and the network connecting them. The total cost of communicating with a certain mapping is minimized using simulated annealing, and thus irregular communication graphs can be mapped to irregular network topology graphs. This is important as that enables the approach to be used on production machines with multiple users, where the nodes allocated to users are not always close to each other on the network.

The optimization procedure does not guarantee that the global minimum is found; the final state will most likely be a good local minimum. The slower the cooling schedule is, the better minima are in general found as the algorithm is able to sample a greater portion of phase space. For optimizing the placement of subdomains on a network this is in general not a problem; as long as a local minimum with close to optimal cost is found the performance will increase. The time required to optimize the mapping was on 1000 cores on the order of 15 s, which is still reasonable if done infrequently. Slower cooling schedules only improved the cost marginally showing that close to optimal mappings are obtained. It should also be noted that the cooling schedule is most likely not optimal, and optimization time could be reduced by tuning it. As the time required to optimize the mapping scales linearly with the number of processes the mapping optimization would at petascale level (100 000 cores) already take almost half an hour for parallel applications utlizing only MPI, indicating that in the present form this technique is only useful for medium scale. However, the optimization problem should be scalable to higher core counts when optimizing a hybrid OpenMP-MPI program due to two reasons: the number of processes is smaller and each process could parallelize the optimization algorithm using threads. In modern supercomputers with tens of cores per node, this should enable the present approach to be viable at tens of thousands of cores.

On the two machines where we tested the approach we observed up to 40 % increase in total bandwidth compared to the default placement, and up to 100 % increase compared to randomized placement. This was true both for large and small messages, suggesting both improved bandwidth and latency. We also showed that the bandwidth was closely related to the amount of network traffic. Optimized placement of ranks on nodes through static placement rules is a common optimization technique, but it is not possible to do this if the program uses dynamic load balancing. This work shows how it is possible to achieve the same benefit even in this case. These results are very similar to the ones achieved by Subramoni et al. [8], with a similar 3D stencil benchmark they also observed up to 40 % increased performance. On the Cray XT5m machine we furthermore show that even a simple model for the cost of sending messages over multiple hops on the network boosts performance by up to 20 % over just minimizing the total amount of inter-node communication. For a hybrid OpenMP-MPI application with one process per node, the placement is already optimal for the node-topology. Optimizing for torus-topology can thus provide performance improvements even for these applications. For the Vlasiator application we achieved up 35 % speedup at higher core-counts enabling the code to scale further and thus achieve its scientific goals in the field of space-weather. This can be contrasted with the 15 % speedup achieved by Mercier et al.[7] for the ZEUS-MP astrophysics code, and the 10-15 % speedup observed for the MILC code by Subramoni et al [8].

Acknowledgments. We acknowledge that the results in this paper have been achieved using the PRACE Research Infrastructure resource Curie based in France at CEA.

The project has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC Starting Grant agreement number 200141-QuESpace. The work of SVA, IH and MP is supported by the Academy of Finland.

References

1. Devine, K., Boman, E., Heapby, R., Hendrickson, B., Vaughan, C.: Zoltan data management service for parallel dynamic applications. Computing in Science and Engg. 4(2), 90–97 (2002)
2. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Comput. Surv. 34(3), 313–356 (2002)
3. Bhanot, G., Gara, A., Heidelberger, P., Lawless, E., Sexton, J.C., Walkup, R.: Optimizing task layout on the blue gene/l supercomputer. IBM J. Res. Dev. 49(2), 489–500 (2005)
4. Bhatele, A.: Automating topology aware mapping for supercomputers. PhD thesis, Champaign, IL, USA, AAI3425400 (2010)
5. Chen, H., Chen, W., Huang, J., Robert, B., Kuhn, H.: Mpipp: an automatic profile-guided parallel process placement toolset for smp clusters and multiclusters. In: Proceedings of the 20th Annual International Conference on Supercomputing (ICS 2006), pp. 353–360. ACM, New York (2006)
6. Jeannot, E., Mercier, G.: Near-Optimal Placement of MPI Processes on Hierarchical NUMA Architectures. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010, Part II. LNCS, vol. 6272, pp. 199–210. Springer, Heidelberg (2010)
7. Mercier, G., Jeannot, E.: Improving MPI Applications Performance on Multicore Clusters with Rank Reordering. In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S., Dongarra, J. (eds.) EuroMPI 2011. LNCS, vol. 6960, pp. 39–49. Springer, Heidelberg (2011)
8. Subramoni, H., Potluri, S., Kandalla, K., Barth, B., Vienne, J., Keasler, J., Tomko, K., Schulz, K., Moody, A., Panda, D.K.: Design of a scalable infiniband topology service to enable network-topology-aware placement of processes. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2012), pp. 70:1–70:12. IEEE Computer Society Press, Los Alamitos (2012)
9. Palmroth, M., Honkonen, I., Sandroos, A., Kempf, Y., von Alfthan, S., Pokhotelov, D.: Preliminary testing of global hybrid-vlasov simulation: Magnetosheath and cusps under northward interplanetary magnetic field. J. Atm. Solar Terr. Phys. (in press), http://dx.doi.org/10.1016/j.jastp.2012.09.013

10. Honkonen, I., von Alfthan, S., Sandroos, A., Janhunen, P., Palmroth, M.: Parallel grid library for rapid and flexible simulation development. Comp. Phys. Comm. (in press), http://dx.doi.org/10.1016/j.cpc.2012.12.017
11. LeVeque, R.J.: Wave propagation algorithms for multidimensional hyperbolic systems. J. Comput. Phys. 131(2), 327–353 (1997)
12. Langseth, J.O., LeVeque, R.J.: A wave propagation method for three-dimensional hyperbolic conservation laws. J. Comput. Phys. 165(1), 126–166 (2000)
13. Londrillo, P., Zanna, L.D.: On the divergence-free condition in godunov-type schemes for ideal magnetohydrodynamics: the upwind constrained transport method. J. Comput. Phys. 195(1), 17–48 (2004)