Automatic Topology-Aware Task Mapping for Parallel Applications Running on Large Parallel Machines

Abhinav Bhatele
Laxmikant V. Kale

Problem Statement

Performance of communication-bound applications is greatly affected by message latencies and the available network bandwidth. On machines with a non-flat topology (such as a torus or mesh), message latencies depend upon:
- Number of hops the message has to travel
- Contention on the network

The fastest supercomputers today such as Blue Gene/L, Blue Gene/P, Cray XT3 and XT4 have a 3-dimensional (3D) torus network. Hardware latencies in such networks depend on the number of hops:

Blue Gene/L: < 1 us (1 hop), 7 us (68 hops)
Blue Gene/P: < 1 us (1 hop), 5 us (68 hops)

IBM Blue Gene/L

Cray XT3

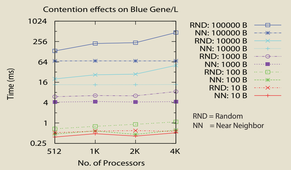
In presence of contention, messages can take even longer to arrive at their destination. In such scenarios, task mapping strategies should consider:
- The communication characterstics of the application
- The network topology of the machine

Using intelligent mapping can improve the efficiency and scaling of applications on large machines. We propose to develop an automatic framework which can generate efficient mappings for tasks to processors based on the communication graph and the processor topology.

Motivation and Examples

Message Latencies on Blue Gene/L

Plot on the right demonstrates the effect of message sizes and torus sizes on the message latency. It shows the average time for sending a message to the nearest processor and to a random processor.



Message latency increases with:
- Increase in the message size in both cases
- Increase in the torus size in the random processor case

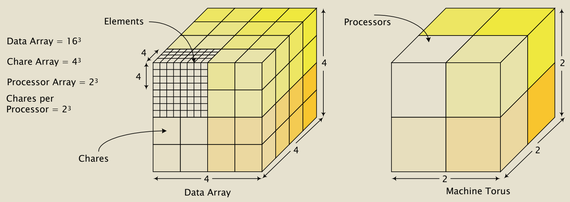
Hence, for applications with huge messages, it would be beneficial to place communicating tasks on the same or nearby processors to avoid this latency.

3D Stencil

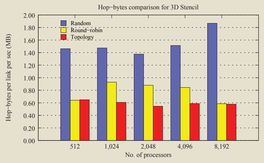
A simple 3-dimensional 7-point stencil (3D Stencil) application is used to demonstrate the high returns from topology-aware mapping of communicating tasks. Charm++ was used to implement this example code.

Charm++ is an asynchronous message-driven parallel runtime system. The concept of virtualization in Charm++ allows multiple objects on a processor. These objects are the basic unit of computation. Having multiple objects allows adpative overlap of communication and computation.

Communication in 3D Stencil is fairly straightforward
- Each object is responsible for the computation of a 3D block of data
- Every object communicates with six neighbors, two in each dimension
- Optimal mapping maps a 3D block of objects together on a processor



	1 object per PE			8 objects per PE		
Processors	RND	RR	TO	RND	RR	TO
512	407.25	156.09	153.85	416.26	147.18	146.61
1024	270.56	96.48	82.19	294.59	101.82	76.80
2048	181.45	49.74	42.11	161.94	46.43	40.24
4096	115.56	25.61	21.27	93.27	24.89	23.65
8192	53.44	10.67	10.81	32.59	11.41	11.34

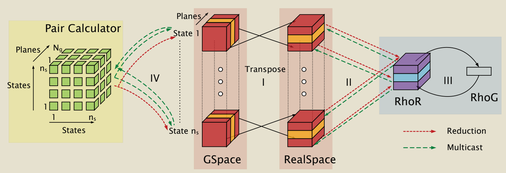


A simple mapping scheme which places contiguous 3D blocks of objects on nearby processors minimizes communication
- Results on Blue Gene/L for two different virtualization ratios shown above
- Improvement up to ~25% for some processor counts
- Corresponding reduction in the number of total hops (links traversed) for all messages in the program

These results suggest that reducing the hops travelled can reduce message latencies and contention for an application and lead to significant performance benefits.

Static Regular Communication: OpenAtom

OpenAtom is a production quantum chemistry code implemented in Charm++
- Implements a fine-grained parallelization of the Car-Parrinello ab-initio Molecular Dynamics (CPAIMD) method
- Every iteration consists of ten complex phases executed by fifteen arrays of Charm++ objects
- Each array communicates with a few other arrays in a specific regular pattern
- A heavily communication-bound application which is greatly affected by network performance (message latencies and performance under bandwidth contention)



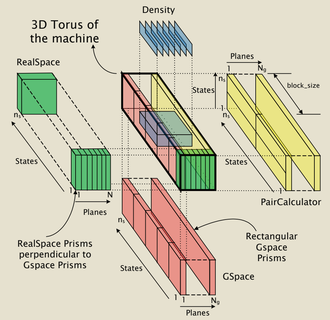
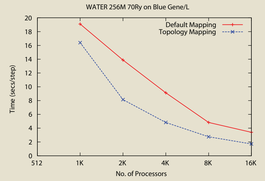


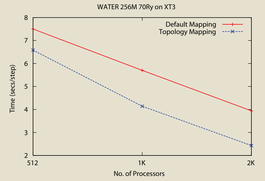
Figure above shows the important arrays in OpenAtom
- GSpace and RealSpace interact through transposes. G(s, *) interacts with R(s, *): state-wise communication
- Gspace and PairCalculator interact through multicasts and reductions. G(*, p) interacts with P(*, *, p): plane-wise communication
- RhoR(*, p) interacts with R(*, p): plane-wise communication

OpenAtom presents a communication scenario where arrays communicate along orthogonal dimensions with other arrays. This makes the mapping problem difficult because of conflicting requirements for the optimal mapping of different arrays.

A hybrid approach (see figure on top left):
- The 3D torus of the machine is divided into prisms along the longest dimension
- Each prism holds all states of a few planes of GSpace
- RealSpace is placed in perpendicular prisms formed by all planes of one state of GSpace (benefits state-wise communication)
- PairCalculator objects are placed in prisms similar to GSpace (plane-wise)

These mapping schemes improve the performance of OpenAtom on machines such as Blue Gene/L, Blue Gene/P and XT3
- Plots on the right show the execution time for the WATER_256M_70Ry system which has 256 water molecules.
- Benefit upto ~40% in some cases.
- Mapping also helps overcome scaling bottlenecks on small systems such as WATER _32M_70Ry when using more than 2048 processors on Blue Gene/L.



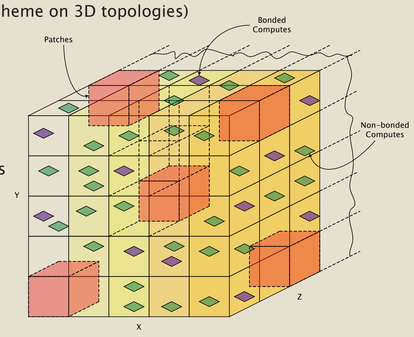


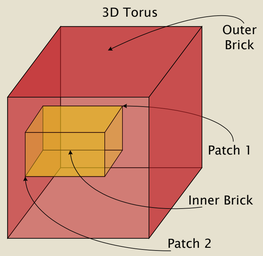
NAMD: Topology-aware Load Balancing

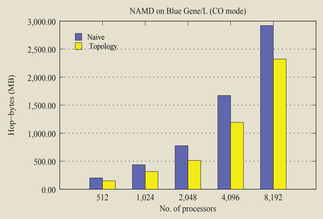
NAMD is a widely used highly scalable parallel production Molecular Dynamics (MD) code
- Objects called 'patches' multicast data to multiple other objects called 'computes'
- Every compute receives data from two patches
- Patches are assigned statically to processors (using an ORB scheme on 3D topologies)
- Computes are placed by a measurement-based load balancer

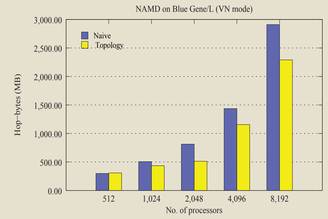
Load Balancing Strategy:
- Pick the heaviest compute
- Find an underloaded processor and place compute on it
- Order of preference:
  - -- Processor hosting one of the patches
  - -- Processor with which one of the patches already interacts
  - -- Any underloaded processor

Topology-aware Strategy:
- Divide the entire torus into two regions
- Inner box: region defined by the processors hosting the two patches with which the compute interacts
- Outer Box: region defined by Entire Torus minus Inner Box
- First look for a underloaded processor in the Inner Box
- Otherwise place in the Outer Box









In NAMD, we see a specific scenario where each target of the multicast receives messages from only two sources. In general, there can be n sources from which a target can receive messages. Hence, this research can be extended into a section-multicast and topology-aware load balancer for multiple multicast sources and targets. This is useful, for example, in matrix multiplication where every object sends its part of the array to other objects in its row and also receives from all other elements in its row. Such strategies do not need manual identification of the communication patterns in the application.

Proposed Work

Building on the experience gained from mapping of specific applications, we propose to develop an automatic topology-sensitive mapping framework
- It will obtain topology information about the machine at runtime
- It will gather information about the communication in the application
- Using various heuristics, it will try to arrive at a near-optimal solution

Topology Information

At runtime, when a partition is allocated for a job, we need information about the processor graph:
- Number of processors in the partition
- Size of each dimension (if the topology is a 3D torus)
- Mapping of ranks to physical processors (XYZT or TXYZ or ...)
- Number of cores per node

Communication Graph Information

Different communication scenarios:
- Regular static communication, as in 3D Stencil or 3D implementation of matrix multiplication (point-to-point messages or multicasts to specific members of the 3D array)
- Arbitrary static communication, as in meshing frameworks
- Arbitrary dynamic communication graph, as in MD applications

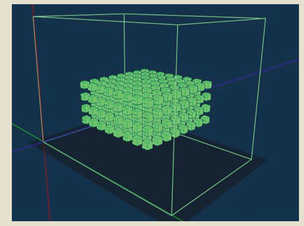


Image generated using http://bigben-monitor.psc.edu

The communication graph can be obtained at compile time (if the code is written in a static data-flow language) or at runtime (by instrumenting the code and saving the communication information).

Finding an Optimal Mapping

Strategy changes depending on the scenario:
- In case of regular scenarios, we can use pattern matching to find regular communication patterns and use straightforward mapping schemes
- In case of arbitrary static graphs, we can do an intelligent initial mapping based on heuristics
- If the communication graph is dynamic, then we can use different topology-aware strategies in the load balancing framework

Why does topology mapping make a difference?

Virtual cut-through and wormhole routing suggest that message latencies should be independent of the number of hops travelled by a message:

$$(L_f/B)*D + L/B$$

But tests using simple benchmarks and performance improvements for production applications suggest otherwise. While hardware latencies on 3D torus machines might be one factor, we need to investigate other possible causes for increased message latencies in the default case which gets reduced by topology-aware mapping:
- Might be due to network congestion, or
- Might be due to link contention

We plan to use performance tools on Blue Gene/L and Blue Gene/P (UPC, PAPI, HPM) to find the resource for which contention happens.

Summary

Topology-aware mapping can improve the performance and scaling of parallel applications significantly. We have obtained encouraging results for 3D Stencil, OpenAtom and NAMD on IBM Blue Gene/L, Blue Gene/P and Cray XT3.

We propose to build an automatic mapping framework which will arrive at near-optimal mapping solutions for most communication scenarios and non-flat topologies. We also plan to extend this work to clusters built from n-way SMPs.

References

[1] Abhinav Bhatele, Laxmikant V. Kale, Application-specific Topology-aware Mapping for Three Dimensional Topologies, *Proceedings of Workshop on Large-Scale Parallel Processing (IPDPS '08)*, 2008
[2] Abhinav Bhatele, Eric Bohm, Laxmikant V. Kale, Improving parallel scaling performance using topology-aware task mapping on Cray XT3 and IBM Blue Gene/L, *Submitted to ICPP '08*
[3] Abhinav Bhatele, Sameer Kumar, Chao Mei, James C. Phillips, Gengbin Zheng, Laxmikant V. Kale, Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms, *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2008
[4] Abhinav Bhatele, Application-specific Topology-aware Mapping and Load Balancing for three-dimensional Torus Topologies, *Master's Thesis, Department of Computer Science, University of Illinois*, 2007
[5] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. C. Sexton, R. Walkup, Optimizing task layout on the Blue Gene/L supercomputer, *IBM Journal of Research and Development, Volume 49, Number 2/3*, 2005
[6] IBM Blue Gene team, Overview of the IBM Blue Gene/P project, *IBM Journal of Research and Development, Volume 52, Number 1/2*, 2008
[7] Deborah Weisser, Nick Nystrom, Chad Vizino, Shawn T. Brown, John Urbanic, Optimizing Job Placement on the Cray XT3. *48th Cray User Group Meeting 2006 Proceedings*, 2006