Topology, Bandwidth and Performance: A New Approach in Linear Orderings for Application Placement in a 3D Torus

Carl Albing
Cray Inc., University of Reading;
Norm Troullier, Stephen Whalen,
Ryan Olson, Joe Glenski
Cray Inc.;
Hugo Mills
University of Reading, UK

SIAM 2012, September 2012
Savannah, Georgia, USA
Acknowledgements

We are very grateful for the help of several people in the benchmarking, MPI, Tools, ALPS and support groups within Cray Inc.

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.
Overview

- Challenge: Topological Placement
- Solution: Linear Ordering – early success
- Improving Bisection Bandwidth
- New Challenges
  - scale
  - anisotropisms
- New Ordering
- Real Application Example
  - before, after
- What's Next
The Placement Problem

- Supercomputer
  - 100 to 20,000 nodes
  - 3D torus interconnect
- Application wants, e.g., 2054 cores/ranks/instances
  - Which nodes?
- Compact Placement
- High Utilization
- Application “Agnostic”
  - No user input
- Wide Range of Sizes
Aside: “Obvious” Solution

- Application-centric
  - The “shape” of an application
- Requires user knowledge
  - Parameters supplied by user
- The “bin packing” problem
  - Packing suitcases in a car
  - Tetris – but 3D!
- NP-hard
- Utilization suffers
Preserving Processor Locality

- Linear Ordering of Nodes
- Preserving Processor Locality
  - Application Geometry
  - Linear List
  - Torus Placement
- Application Level Placement Scheduler (ALPS)
  - simple greedy algorithm
  - dealing cards
- Original Ordering:
  - Node ID # order
  - Not good for placement
Early Solution

- Improve placement for better performance
- Topologically informed
- Encourage wrap-around
- Still “dealing cards”
  - but stacking the deck
- How to order is the key
  - Minimum Dimension First
  - MDF a.k.a. XYZ ordering
Did it matter?
New Challenge

- Scale! - Large systems
- Farther to go for wrap-around
- Planar allocations - wasting 3D interconnect
New Challenge

- Scale! - High Core counts
  - Large jobs become small(er)
  - e.g. 8192 cores:
    - XT4 needs 2048 nodes, 64 cages, 21+ cabinets
    - XT6 needs only 11 cages in 4 cabinets
- Planar allocations - *bandwidth wasted*
New Solution

- Third dimension
  - “Thicker” allocation
  - Doubles the bisection b/w
- Allocate in $2 \times 2 \times 2$ blocks
- Blocks are then tiled
  - Across the machine
  - In “MDF” order
New Solution

- Reflected Gray code ordering within blocks
- Contiguous enumeration along first filled dimension
- I.e., start next where prev left off

![Diagram showing reflected Gray code ordering]
New Challenge

- New interconnect (Cray XE Gemini)
- X and Z axes have double links, Y axis has single links compared to Cray XT
New Challenge

- First try:
  - On XE6, we enforce that Y should be the last dimension to be blocked, regardless of its size
  - X and Z have double links, Y has single links
  - During initial XE6 development, all links were equal speed, so Y was the least-preferred dimension
  - Note that a $2 \times 2 \times 2$ block of nodes is a $2 \times 1 \times 2$ block of Geminis
New Challenge

- Anisotropic interconnect
- X, Z double links
- Y speed differs
  - faster w/in mezz.
- Z speed differs
  - faster across backplane
Rethinking the Order

The optimal placement algorithm will

1. Minimize job-job interaction
2. Minimize variation in job execution time with respect to random starting points within the system
3. Minimize latencies between nodes within a job
4. Maximize each job’s
   • Global bandwidth (a.k.a. network capacity)
   • Node connectivity
   • Bisection bandwidth

All while providing high utilization!
Rethinking...

1. Minimize job-job interaction

- Linear arrangement, (e.g. slices)
  - hurts bisection bandwidth

- restrict job sizes to fit predetermined rectangular prisms –
  - hurts utilization or turnaround
2. Minimize variation in job execution time \textit{wrt} random starting points within the system

- Job shape must have translational invariance throughout the system (same shape no matter where placed)
- Network inhomogeneity $\Rightarrow$ restriction on job sizes
3. Minimize latencies between nodes within a job
   - “compactness”
   - nearest neighbor
   - vs.
   - all-to-all

- theoretical minimum:
  - for 3D
    - “manhattan distance”
    - octahedrons
  - hard to pack – utilization
Rethinking...

4. Maximize each job’s connectivity / bisection b/w
   • How?
     Spread each job over the full machine
     (or a large part of it)
4. Maximize each job’s connectivity / bisection b/w

- How?
  Spread each job over the full machine (or a large part of it)
Rethinking...

4. Maximize each job’s connectivity / bisection b/w
   • How?
     Spread each job over the full machine
     (or a large part of it)

Nodes for Job A
Nodes for Job B
Rethinking...

4. Maximize each job’s connectivity, bisection b/w
   • How?
     Spread each job over the full machine
     (or a large part of it)

   • Result:
   • **Maximizes** job-job interaction
Rethinking...

1. Minimize job-job interaction
   - Linear arrangement, or restrict job sizes to fit predetermined rectangular prisms – hurts utilization or turnaround

2. Minimize variation in job execution time with respect to random starting points within the system
   - Job shape must have translational invariance throughout system
   - Network inhomogeneity $\Rightarrow$ restriction on job sizes

3. Minimize latencies between nodes within a job
   - Differences between nearest neighbor and all-to-all; octahedrons

4. Maximize each job’s bandwidth/connectivity/bisection
   - Spread each job over the full machine, or a large part of it
   - maximizes job-job interaction

No one solution addresses all issues. We give precedence to the first two, as these issues are imperative to many customers.
Building Cray's Solution

- Build on existing xyz-by2 infrastructure
- Select a larger block size to accommodate frequent customer job sizes [1K–8K]
- Recursive doubling in favored dimensions to maximize bisection
- 4 × 4 × 4 block holds 1536 cores
- Block replicated across Z dimension, then Y, finally X; takes advantage of higher average link speeds in Z and Y
- Also use 3 × 4 × 4 and 5 × 4 × 4 blocks to fill all possible X extents

192 cores  
double Y  
384 cores  
double Z  
768 cores  
double X  
1536 cores
Ordering Within a Block

- $4 \times 4 \times 4$ node block = $4 \times 2 \times 4$ Gemini block
- First dimension filled is “Gemini dimension”
  - Reduces possible job-job interaction
- Serpentine ordering resembles a space-filling curve
  - Path turns whenever possible
    - Increases connectivity
    - Increases bisection
    - Lowers hop counts
    - Helps to keep jobs on distinct links
  - Starting and ending points line up, to allow path contiguity when blocking first dimension
Rapid Prototyping
Benchmark runs with 68 cabinets, comparing 2x2x2 and the improved (serpentine) 4x4x4 ordering.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>28-cab 2x2x2</th>
<th>68-cab 2x2x2</th>
<th>68-cab 3x2x2</th>
<th>68-cab 4x5x4x4</th>
<th>Speedup c.f. 68c. 2x2x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM</td>
<td>346</td>
<td>338</td>
<td>340</td>
<td>338</td>
<td>0%</td>
</tr>
<tr>
<td>GAMESS</td>
<td>1386</td>
<td>1858</td>
<td>1572</td>
<td>1478</td>
<td>20%</td>
</tr>
<tr>
<td>GTC</td>
<td>1234</td>
<td>1226</td>
<td>1226</td>
<td>1226</td>
<td>0%</td>
</tr>
<tr>
<td>ImpactT</td>
<td>604</td>
<td>629</td>
<td>600</td>
<td>597</td>
<td>5%</td>
</tr>
<tr>
<td>MAESTRO</td>
<td>1969</td>
<td>2233</td>
<td>2105</td>
<td>1982</td>
<td>11%</td>
</tr>
<tr>
<td>MILC</td>
<td>909</td>
<td>1355</td>
<td>1237</td>
<td>919</td>
<td>32%</td>
</tr>
<tr>
<td>PARATEC</td>
<td>386</td>
<td>413</td>
<td>395</td>
<td>383</td>
<td>7%</td>
</tr>
</tbody>
</table>
Which Ordering?

- No one single “best” ordering
- “MDF” or MaxMajor order
  - smaller, single-core systems
  - good when one dimension is short
- “Thicker” solutions
  - High core-count nodes
  - Large systems
  - Equal size systems
    - no “short” dimension
What's Next?

- Measure effect over time
  - coalesces automatically vs.
  - fragmentation, dispersal
- Alternate algorithms, same ordering
  - first fit, best fit, etc.
- Involve queueing system
  - online vs. offline algorithms
- Application communication styles
- Exascale orderings
  - next interconnect architecture
Conclusions

- We improved performance
  - across larger systems
  - across a wider array of jobs
  - across a mix of jobs
- Smarter list made for:
  - higher bisection bandwidth
  - no user intervention
  - very low cost, very low risk
  - for both regular, anisotropic networks

Questions?