Topology Aware Mapping

Abhinav Bhavele

Charm++ Tutorial
Computer Network Information Center,
Chinese Academy of Sciences
Motivation

• Running a parallel application on a linear array of processors:
Motivation

- Running a parallel application on a linear array of processors:
Motivation

- Running a parallel application on a linear array of processors:
Motivation

- Running a parallel application on a linear array of processors:

- Typical communication is between random pairs of processors simultaneously
Benchmark Creating Artificial Contention

- Pair each processor with a partner that is $n$ hops away

1 hop

2 hops

3 hops
Results: Contention

Effect of distance on latencies (Torus - 8 x 8 x 16)

Bhatele A., Kale L.V., Quantifying Network Contention on Large Parallel Machines, Parallel Processing Letters (Special Issue on Large-Scale Parallel Processing), 2009. Best Poster Award, ACM Student Research Competition, Supercomputing 2008, Austin, TX.
Results: Contention

Effect of distance on latencies (Torus - 8 x 8 x 16)

Bhatele A., Kale L.V., Quantifying Network Contention on Large Parallel Machines, Parallel Processing Letters (Special Issue on Large-Scale Parallel Processing), 2009. Best Poster Award, ACM Student Research Competition, Supercomputing 2008, Austin, TX.
Interconnect Topologies

- Three dimensional meshes
  - 3D Torus: Blue Gene/L, Blue Gene/P, Cray XT4/5

- Trees
  - Fat-trees (Infiniband) and CLOS networks (Federation)

- Dense Graphs
  - Kautz Graph (SiCortex), Hypercubes

- Future Topologies?
  - Blue Waters, Blue Gene/Q

Roadrunner Technical Seminar Series, March 13th 2008, Ken Koch, LANL
Application Topologies

http://wrf-model.org/plots/realtime_main.php

http://www.ks.uiuc.edu/Gallery/Science/

http://math.lanl.gov/Research/Projects/meshing.shtml
Application Topologies

http://wrf-model.org/plots/realtime_main.php

http://www.ks.uiuc.edu/Gallery/Science/

We want to map communicating objects closer to one another

http://math.lanl.gov/Research/Projects/meshing.shtml
The Mapping Problem

• Applications have a communication topology and processors have an interconnect topology

• Definition: Given a set of communicating parallel “entities”, map them on to physical processors to optimize communication

• Goals:
  • Minimize communication traffic and hence contention
  • Balance computational load (when n > p)
Scope of this work

- Currently we are focused on 3D mesh/torus machines
- For certain classes of applications
Outline

• Case studies:
  • OpenAtom
  • NAMD

• Automatic Mapping Framework
  • Pattern matching

• Heuristics for Regular Graphs
• Heuristics for Irregular Graphs
Case Study I: OpenAtom

Performance on Blue Gene/L

Number of cores: 512, 1024, 2048, 4096, 8192

Time per step (s): 0.075, 0.15, 0.225, 0.3, 0.3

Default Mapping
Diagnosis

Timeline view (OpenAtom on 8,192 cores of BG/L) using the performance visualization tool, Projections
Mapping of OpenAtom Arrays

A. Bhatele, E. Bohm, and L.V. Kale. A Case Study of Communication Optimizations on 3D Mesh Interconnects. In Euro-Par, LNCS 5704, pages 1015–1028, 2009. Distinguished Paper Award, Feng Chen Memorial Best Paper Award
Mapping of OpenAtom Arrays

Paircalculator and GSpace have plane-wise communication

RealSpace and GSpace have state-wise communication

A. Bhatele, E. Bohm, and L.V. Kale. A Case Study of Communication Optimizations on 3D Mesh Interconnects. In Euro-Par, LNCS 5704, pages 1015–1028, 2009. Distinguished Paper Award, Feng Chen Memorial Best Paper Award
Mapping of OpenAtom Arrays

PairCalculator and GSpace have plane-wise communication

RealSpace and GSpace have state-wise communication

A. Bhavele, E. Bohm, and L.V. Kale. A Case Study of Communication Optimizations on 3D Mesh Interconnects. In Euro-Par, LNCS 5704, pages 1015–1028, 2009. Distinguished Paper Award, Feng Chen Memorial Best Paper Award
Performance Benefits from Mapping

Performance on Blue Gene/L

- **Default Mapping**
- **Topology Mapping**

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Time per step (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.3</td>
</tr>
<tr>
<td>1024</td>
<td>0.225</td>
</tr>
<tr>
<td>2048</td>
<td>0.15</td>
</tr>
<tr>
<td>4096</td>
<td>0.075</td>
</tr>
<tr>
<td>8192</td>
<td></td>
</tr>
</tbody>
</table>
Diagnosis of Improvement

Timeline of 1 iteration of OpenAtom running WATER_256M_70Ry on 8192 cores of BG/L

Timeline view using the performance visualization tool, Projections
OpenAtom Performance on Blue Gene/P

Application Performance

Blue Gene/P

Time per step (s)

Number of cores

Default Mapping

Topology Mapping
OpenAtom Performance on Blue Gene/P

**Application Performance**

- Default Mapping
- Topology Mapping

**Performance Counters**

- System Bandwidth (GB/step)
  - Default Mapping
  - Topology Mapping

---

December 16th, 2010

CNIC @ CAS © Abhinav Bhavele
OpenAtom Performance on Blue Gene/P

Application Performance

![Graph showing time per step vs. number of cores for Default Mapping and Topology Mapping]

- Time per step (s)
- Number of cores: 1024, 2048, 4096, 8192
- Default Mapping
- Topology Mapping

Performance Counters

![Bar chart showing system bandwidth for Default Mapping and Topology Mapping]

- System Bandwidth (GB/step)
- Number of cores: 1024, 2048, 4096, 8192
OpenAtom Performance on Cray XT3
OpenAtom Performance on Cray XT3

- Cray XT3:
  - Link bandwidth - 3.8 GB/s (XT3), 0.425 (BG/P), 0.175 (BG/L)
  - Bytes per flop - 8.77 (XT3), 0.375 (BG/P and BG/L)
OpenAtom Performance on Cray XT3

- Cray XT3:
  - Link bandwidth - 3.8 GB/s (XT3), 0.425 (BG/P), 0.175 (BG/L)
  - Bytes per flop - 8.77 (XT3), 0.375 (BG/P and BG/L)

- Job schedulers on Cray are not topology aware
OpenAtom Performance on Cray XT3

- Cray XT3:
  - Link bandwidth - 3.8 GB/s (XT3), 0.425 (BG/P), 0.175 (BG/L)
  - Bytes per flop - 8.77 (XT3), 0.375 (BG/P and BG/L)

- Job schedulers on Cray are not topology aware

- Performance Benefit at 2048 cores: 40% (XT3), 45% (BG/P), 41% (BG/L)
Case Study II: NAMD

Communication between patches and computes

NAMD Performance on Blue Gene/P

**Measured Hop-bytes**

- Topology Oblivious
- TopoAware Patches
- TopoAware Computes

![Graph showing measured hop-bytes for different numbers of cores](image)

**Evaluation Metric:**

**Hop-bytes**

\[ HB = \sum_{i=1}^{n} d_i \times b_i \]

- \( d_i = \text{distance} \)
- \( b_i = \text{bytes} \)
- \( n = \text{no. of messages} \)

- Indicates amount of traffic and hence contention on the network

- Previously used metric:

  \[ d(e) = \max \{ d_i | e_i \in E \} \]

In VLSI circuit design and early parallel computing work, emphasis was placed on reducing the number of links to minimize the contribution of communication to circuit and processor power. The most well known is the principle of persistence, which is that communication patterns tend to persist over time and even if they change, the change is gradual and referred to as the principle of persistence. This assumes that load patterns tend to persist over time and even if they change, the change is gradual and referred to as the principle of persistence.

Previously used metric: maximum dilation

\[ d(e) = \max \{ d_i | e_i \in E \} \]
NAMD Performance on Blue Gene/P

Measured Hop-bytes

- Topology Oblivious
- TopoAware Patches
- TopoAware Computes

The number of computes on a processor and their individual computational loads determines its computational load and the number of proxies on a processor indicates its communication load. Load balancing in NAMD is measurement-based. This assumes that load patterns tend to persist over time and even if they change, the change is gradual (referred to as the principle of persistence). The load balancing framework records information about object computational loads for some time steps. It also records the communication graph between the patches and proxies. This information is collected on one processor and based on the instrumentation data, a load balancing phase is executed. Decisions are then sent to all processors. The current strategy is centralized and we shall later discuss future work to make it fully distributed.
NAMD Performance on Blue Gene/P

Measured Hop-bytes

```
Hop-bytes (MB per iteration)

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Topology Oblivious</th>
<th>TopoAware Patches</th>
<th>TopoAware Computes</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0</td>
<td>375</td>
<td>750</td>
</tr>
<tr>
<td>1024</td>
<td>750</td>
<td>1125</td>
<td>1500</td>
</tr>
<tr>
<td>2048</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
</tr>
<tr>
<td>4096</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
</tr>
</tbody>
</table>
```

Application Performance

```
Time per step (ms)

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Topology Oblivious</th>
<th>TopoAware Patches</th>
<th>TopoAware Computes</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>15</td>
<td>11.25</td>
<td>7.5</td>
</tr>
<tr>
<td>1024</td>
<td>11.25</td>
<td>7.5</td>
<td>3.75</td>
</tr>
<tr>
<td>2048</td>
<td>7.5</td>
<td>3.75</td>
<td>2.5</td>
</tr>
<tr>
<td>4096</td>
<td>3.75</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>8192</td>
<td>1.25</td>
<td>1.25</td>
<td>1.25</td>
</tr>
<tr>
<td>16384</td>
<td>1.25</td>
<td>1.25</td>
<td>1.25</td>
</tr>
</tbody>
</table>
```

The number of computes on a processor and their individual computational loads determines its computational load and the number of proxies on a processor indicates its communication load. Load balancing in NAMD is measurement-based. This assumes that load patterns tend to persist over time and even if they change, the change is gradual (referred to as the principle of persistence). The load balancing framework records information about object (compute) loads for some time steps. It also records the communication graph between the patches and proxies. This information is collected on one processor and based on the instrumentation data, a load balancing phase is executed. Decisions are then sent to all processors. The current strategy is centralized and we shall later discuss future work to make it fully...
The number of computes on a processor and their individual computational loads determines its computational load and the number of proxies on a processor indicates its communication load. Load balancing in NAMD is measurement-based. This assumes that load patterns tend to persist over time and even if they change, the change is gradual (referred to as the principle of persistence). The load balancing framework records information about object (compute) loads for some time steps. It also records the communication graph between the patches and proxies. This information is collected on one processor and based on the instrumentation data, a load balancing phase is executed. Decisions are then sent to all processors. The current strategy is centralized and we shall later discuss future work to make it fully...
Outline

• Case studies:
  • OpenAtom
  • NAMD

• Automatic Mapping Framework
  • Pattern matching

• Heuristics for Regular Graphs
• Heuristics for Irregular Graphs
Automatic Mapping Framework

---

**Pattern Matching Framework**

- **Application communication graph**
  - **Regular Graphs**
    - 2D Object Graph: MXOVL, MXOVL_AL, EXC, COCE, AFFN
    - 3D Object Graph: EXC, COCE, AFFN
  - **Irregular Graphs**
    - W/o coordinate information: BFT, MHT, Infer structure
    - W/ coordinate information: AFFN, COCE, COCE+MHT

**Choose best heuristic depending on hop-bytes**

**Output: Mapping file used for the next run**

---

**8.1 Communication Graph: Identifying Patterns**

Automatic topology aware mapping, as we shall see in the next few sections, uses heuristics for fast scalable runtime solutions. Heuristics can yield more efficient solutions if we can derive concrete information about the communication graph of the application and exploit it. For this, we need to look for identifiable communication patterns, if any, in the object graph. Many parallel applications have relatively simple and easily identifiable 2D, 3D or 4D communication patterns. If we can identify such patterns, then we can apply better suited heuristic techniques for such patterns.
Automatic Mapping Framework

Identifying regular patterns in communication graphs.

Automatic topology aware mapping, as we shall see in the next few sections, uses heuristics for fast scalable runtime solutions. Heuristics can yield more efficient solutions if we can derive concrete information about the communication graph of the application and exploit it. For this, we need to look for identifiable communication patterns, if any, in the object graph. Many parallel applications have relatively simple and easily identifiable 2D, 3D or 4D communication patterns. If we can identify such patterns, then we can apply better suited heuristic techniques for such.
Automatic Mapping Framework

8.1 Communication Graph: Identifying Patterns

Automatic topology aware mapping, as we shall see in the next few sections, uses heuristics for fast scalable runtime solutions. Heuristics can yield more efficient solutions if we can derive concrete information about the communication graph of the application and exploit it. For this, we need to look for identifiable communication patterns, if any, in the object graph. Many parallel applications have relatively simple and easily identifiable 2D, 3D or 4D communication patterns. If we can identify such patterns, then we can apply better suited heuristic techniques for such patterns.

Choose best heuristic depending on hop-bytes

Output: Mapping file used for the next run

Application communication graph

Pattern Matching Framework

Regular Graphs

Irregular Graphs

2D Object Graph

3D Object Graph

W/o coordinate information

W/ coordinate information

MXOVLP, MXOV, AL, EXC, COCE, AFFN

EXC, COCE, AFFN

AffN, COCE, COCE+MHT

BFT, MHT, Infer structure

Processor topology information
Automatic Mapping Framework

Relieve the application developer of the mapping burden
Automatic Mapping Framework

Relieve the application developer of the mapping burden

No change to the application code
Topology Discovery

- Topology Manager API: for 3D interconnects (Blue Gene, XT)

- Information required for mapping:
  - Physical dimensions of the allocated job partition
  - Mapping of ranks to physical coordinates and vice versa

- On Blue Gene machines such information is available and the API is a wrapper

- On Cray XT machines, jump several hoops to get this information and make it available through the same API
Application communication graph

• Several ways to obtain the graph

• MPI applications:
  • Profiling tools (IBM’s HPCT tools)
  • Collect information using the PMPI interface
  • Manually provided by the application end user

• Charm++ applications:
  • Instrumentation at runtime
  • Profiling tools (HPCT): when n = p
Pattern Matching

- We want to identify regular 2D/3D communication patterns

**Input:** $CM_{n,n}$ (communication matrix)

**Output:** $isRegular$ (boolean, true if communication is regular)
- $dims[]$ (dimensions of the regular communication graph)

```
for $i = 1$ to $n$ do
    find the maximum number of neighbors for any rank in $CM_{i,n}$
end for

if max neighbors $\leq 5$ then
    // this might be a case of regular 2D communication
    select an arbitrary rank $start_{pe}$ find its distance from its neighbors
    $dist = \text{difference between ranks of } start_{pe} \text{ and its top or bottom neighbor}$
    for $i := 1$ to $n$ do
        if distance of all ranks from their neighbors $== 1$ or $dist$ then
            $isRegular = true$
            $dim[0] = dist$
            $dim[1] = n/dist$
        end if
    end for
end if
```
Pattern Matching

- We want to identify regular 2D/3D communication patterns

Input: \( CM_{n,n} \) (communication matrix)
Output: \( isRegular \) (boolean, true if communication is regular)
\[
dims[\ ] \text{ (dimensions of the regular communication graph)}
\]
for \( i = 1 \) to \( n \) do
    find the maximum number of neighbors for any rank in \( CM_{i,n} \)
end for
if max neighbors \( \leq 5 \) then
    // this might be a case of regular 2D communication
    select an arbitrary rank \( start_{pe} \) find its distance from its neighbors
    dist = difference between ranks of \( start_{pe} \) and its top or bottom neighbor
    for \( i := 1 \) to \( n \) do
        if distance of all ranks from their neighbors == 1 or dist then
            \( isRegular = true \)
            \( \text{dim}[0] = \text{dist} \)
            \( \text{dim}[1] = n/\text{dist} \)
        end if
    end for
end if
Pattern Matching

- We want to identify regular 2D/3D communication patterns

**Input:** $CM_{n,n}$ (communication matrix)

**Output:** $isRegular$ (boolean, true if communication is regular)
- $dims[\ ]$ (dimensions of the regular communication graph)

```
for $i = 1$ to $n$ do
    find the maximum number of neighbors for any rank in $CM_{i,n}$
end for

if max neighbors $\leq 5$ then
    // this might be a case of regular 2D communication
    select an arbitrary rank $start_{pe}$ find its distance from its neighbors
    $dist =$ difference between ranks of $start_{pe}$ and its top or bottom neighbor

    for $i := 1$ to $n$ do
        if distance of all ranks from their neighbors $== 1$ or $dist$ then
            $isRegular = true$
            $dim[0] = dist$
            $dim[1] = n/dist$
        end if
    end for
end if
```
Example

- WRF running on 32 cores of Blue Gene/P
Example

- WRF running on 32 cores of Blue Gene/P

Pattern matching to identify regular communication patterns such as 2D/3D near-neighbor graphs
Example

- WRF running on 32 cores of Blue Gene/P

Pattern matching to identify regular communication patterns such as 2D/3D near-neighbor graphs
Communication Graphs

- Regular communication:
  - POP (Parallel Ocean Program): 2D Stencil like computation
  - WRF (Weather Research and Forecasting model): 2D Stencil
  - MILC (MIMD Lattice Computation): 4D near-neighbor

- Irregular communication:
  - Unstructured mesh computations: FLASH, CPSD code
  - Many other classes of applications
Outline

• Case studies:
  • OpenAtom
  • NAMD

• Automatic Mapping Framework
  • Pattern matching

• Heuristics for Regular Graphs

• Heuristics for Irregular Graphs
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)

Object Graph: 9 x 8
Processor Graph: 12 x 6
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)

Object Graph: 9 x 8
Processor Graph: 12 x 6
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)

Object Graph: 9 x 8
Processor Graph: 12 x 6
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLGP)

Object Graph: 9 x 8
Processor Graph: 12 x 6
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)

Object Graph: 9 x 8
Processor Graph: 12 x 6
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)
  
  Object Graph: 9 x 8  
  Processor Graph: 12 x 6

- Maximum Overlap with Alignment (MXOV+AL)
  
  Alignment at each recursive call
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)
  
  Object Graph: 9 x 8
  Processor Graph: 12 x 6

- Maximum Overlap with Alignment (MXOV+AL)
  - Alignment at each recursive call

- Expand from Corner (EXCO)
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)
  Object Graph: 9 x 8
  Processor Graph: 12 x 6

- Maximum Overlap with Alignment (MXOV+AL)
  Alignment at each recursive call

- Expand from Corner (EXCO)
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLp)
  
  Object Graph: 9 x 8
  Processor Graph: 12 x 6

- Maximum Overlap with Alignment (MXOV+AL)
- Alignment at each recursive call
- Expand from Corner (EXCO)
Mapping Regular Graphs (2D)

- Maximum Overlap (MXOVLP)
  
  Object Graph: 9 x 8  
  Processor Graph: 12 x 6  

- Maximum Overlap with Alignment (MXOV+AL)

- Alignment at each recursive call

- Expand from Corner (EXCO)
More heuristics ...

- Corners to Center (COCE)
- Start simultaneously from all corners
More heuristics ...

- **Corners to Center (COCE)**
- **Start simultaneously from all corners**
More heuristics ...

- Corners to Center (COCE)
- Start simultaneously from all corners
More heuristics ...

- Corners to Center (COCE)
- Start simultaneously from all corners
More heuristics ...

- Corners to Center (COCE)
- Start simultaneously from all corners

- Affine Mapping (AFFN)

More heuristics ...

- Corners to Center (COCE)
- Start simultaneously from all corners

- Affine Mapping (AFFN)

More heuristics ...

- **Corners to Center (COCE)**
  - Start simultaneously from all corners

- **Affine Mapping (AFFN)**

  \[(x, y) \rightarrow (\lfloor P_x \cdot \frac{x}{O_x} \rfloor, \lfloor P_y \cdot \frac{y}{O_y} \rfloor)\]
More heuristics ...

- **Corners to Center (COCE)**
- **Start simultaneously from all corners**

- **Affine Mapping (AFFN)**

\[(x, y) \rightarrow ([P_x \times \frac{x}{O_x}], [P_y \times \frac{y}{O_y}])\]

More heuristics ...

- **Corners to Center (COCE)**
- **Start simultaneously from all corners**

- **Affine Mapping (AFFN)**

\[(x, y) \rightarrow ([P_x \ast \frac{x}{O_x}], [P_y \ast \frac{y}{O_y}])\]

Running Time

- Pairwise Exchanges (PAIRS)
  - Bokhari, Lee et al.
Running Time

- Pairwise Exchanges (PAIRS) - Bokhari, Lee et al.

![Graph showing running time and hops per byte for different numbers of nodes.](image)
Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Example Mapping

Object Graph: 9 x 8
Processor Graph: 12 x 6

Mapping of 9x8 graph to 12x6 mesh

MXOVLP: 1.66
MXOV+AL: 1.65
EXCO: 2.31
COCE: 1.91
Mapping of 9x8 graph to 12x6 mesh

MXOVLP: 1.66  MXOV+AL: 1.65  EXCO: 2.31  COCE: 1.91
Mapping of 9x8 graph to 12x6 mesh

STEP: 1.39  AFFN1: 1.77  AFFN2: 1.53  AFFN3: 1.91
Mapping of 9x8 graph to 12x6 mesh

STEP: 1.39
AFFN1: 1.77
AFFN2: 1.53
AFFN3: 1.91
Evaluation

- MXOVLP
- MXOV+AL
- EXCO
- COCE
- AFFN
- PAIRS

Hops per byte

- 27x44 to 36x33 (~1k nodes)
- 100x40 to 125x32 (~4k nodes)
- 128x128 to 512x32 (~16k nodes)
- 320x200 to 125x512 (~64k nodes)
Mapping 2D Graphs to 3D

• Map a two-dimensional object graph to a three-dimensional processor graph

• Divide object graph into subgraphs once each for the number of planes
  • Stacking
  • Folding

• Best 2D to 2D heuristic chosen based on hop-bytes
Results: 2D Stencil on Blue Gene/P

Hop-bytes

Default Mapping
Topology Mapping

Number of cores

Hops per byte

512 1024 2048 4096 8192 16384
Results: 2D Stencil on Blue Gene/P

**Hop-bytes**

- Default Mapping
- Topology Mapping

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>hops per byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>5</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
</tr>
<tr>
<td>2048</td>
<td>15</td>
</tr>
<tr>
<td>4096</td>
<td>20</td>
</tr>
<tr>
<td>8192</td>
<td>10</td>
</tr>
<tr>
<td>16384</td>
<td>5</td>
</tr>
</tbody>
</table>

**Performance**

- Default Mapping
- Topology Mapping

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>time per step (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>417.5</td>
</tr>
<tr>
<td>1024</td>
<td>435</td>
</tr>
<tr>
<td>2048</td>
<td>452.5</td>
</tr>
<tr>
<td>4096</td>
<td>470</td>
</tr>
<tr>
<td>8192</td>
<td>470</td>
</tr>
<tr>
<td>16384</td>
<td>495</td>
</tr>
</tbody>
</table>
Increasing communication

- With faster processors and constant link bandwidths
  - computation is becoming cheap
  - communication is a bottleneck
- Trend for bytes per flop
  - XT3: 8.77
  - XT4: 1.357
  - XT5: 0.23

### 2D Stencil on BG/P

- Default Mapping
- Topology Mapping

<table>
<thead>
<tr>
<th>Message size</th>
<th>Time per step (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 B</td>
<td>0.1</td>
</tr>
<tr>
<td>2 KB</td>
<td>1</td>
</tr>
<tr>
<td>8 KB</td>
<td>10</td>
</tr>
<tr>
<td>32 KB</td>
<td>100</td>
</tr>
<tr>
<td>128 KB</td>
<td>100</td>
</tr>
</tbody>
</table>
Results: WRF on Blue Gene/P

Hops from IBM HPCT

- Default
- Topology
- Lower Bound

Average hops per byte

Number of nodes

- 256
- 512
- 1024
- 2048
- 4096

December 16th, 2010
Results: WRF on Blue Gene/P

- Performance improvement negligible on 256 and 512 cores

Hops from IBM HPCT

- Default
- Topology
- Lower Bound

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Average hops per byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>512</td>
<td>2</td>
</tr>
<tr>
<td>1024</td>
<td>3</td>
</tr>
<tr>
<td>2048</td>
<td>4</td>
</tr>
<tr>
<td>4096</td>
<td>3</td>
</tr>
</tbody>
</table>

December 16th, 2010
Results: WRF on Blue Gene/P

- Performance improvement negligible on 256 and 512 cores

- On 1024 nodes:
  - Hops reduce by: 63%
  - Time for communication reduces by 11%
  - Performance improves by 17%
Results: WRF on Blue Gene/P

- Performance improvement negligible on 256 and 512 cores

- On 1024 nodes:
  - Hops reduce by 63%
  - Time for communication reduces by 11%
  - Performance improves by 17%

Hops from IBM HPCT

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Average hops per byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>512</td>
<td>1.7</td>
</tr>
<tr>
<td>1024</td>
<td>1.1</td>
</tr>
<tr>
<td>2048</td>
<td>1.06</td>
</tr>
<tr>
<td>4096</td>
<td>1.04</td>
</tr>
</tbody>
</table>

17% lower bound.
Results: WRF on Blue Gene/P

- Performance improvement negligible on 256 and 512 cores

- On 1024 nodes:
  - Hops reduce by: 63%
  - Time for communication reduces by 11%
  - Performance improves by 17%

![Graph showing hops per byte for different node counts]

- Hops from IBM HPCT
  - Default: 17%
  - Topology: 5%
  - Lower Bound: 5%
Outline

• Case studies:
  • OpenAtom
  • NAMD

• Automatic Mapping Framework
  • Pattern matching

• Heuristics for Regular Graphs

• Heuristics for Irregular Graphs
Mapping Irregular Graphs

Object graph: 90 nodes

Processor Mesh: 10 x 9
Two different scenarios

- There is no spatial information associated with the node
  - Option 1: Work without it
  - Option 2: If we know that the simulation has a geometric configuration, try to infer the structure of the graph

- We have geometric coordinate information for each node
  - Use coordinate information to avoid crossing of edges and for other optimizations
No coordinate information
No coordinate information

- Breadth first traversal (BFT)
  - Start with a random node and one end of the processor mesh
  - Map nodes as you encounter them close to their parent
No coordinate information

- **Breadth first traversal (BFT)**
  - Start with a random node and one end of the processor mesh
  - Map nodes as you encounter them close to their parent

- **Max heap traversal (MHT)**
  - Start with a random node and one end/center of the mesh
  - Put neighbors of a mapped node into the heap (node at the top is the one with maximum number of mapped neighbors)
  - Map elements in the heap one by one around the centroid of their mapped neighbors
Mapping visualization

BFT: 2.89

MHT: 2.69
Inferring the spatial placement
Inferring the spatial placement

- Graph layout algorithms
  - Force-based layout to reduce the total energy in the system
- Use the graphviz library to obtain coordinates of the nodes
Inferring the spatial placement

- Graph layout algorithms
  - Force-based layout to reduce the total energy in the system
- Use the graphviz library to obtain coordinates of the nodes
With coordinate information

- **Affine Mapping (AFFN)**
  - Stretch/shrink the object graph (based on coordinates of nodes) to map it on to the processor grid
  - In case of conflicts for the same processor, spiral around that processor

- **Corners to Center (COCE)**
  - Use four corners of the object graph based on coordinates
  - Start mapping simultaneously from all sides
    - Place nodes encountered during a BFT close to their parents
Mapping visualization

AFFN: 3.17

COCE: 2.88
• COCE+MHT Hybrid:
  • We fix four nodes at geometric corners of the mesh to four processors in 2D
  • Put neighbors of these nodes into a max heap
• Map from all sides inwards
  • Starting from centroid of mapped neighbors

COCE: 2.78
Time Complexity
Time Complexity

- All algorithms discussed above choose a desired processor and spiral around it to find the nearest available processor
- Heuristics generally applicable to any topology
Time Complexity

- All algorithms discussed above choose a desired processor and spiral around it to find the nearest available processor.
- Heuristics generally applicable to any topology.
- Depending on the running time of findNext:

<table>
<thead>
<tr>
<th></th>
<th>BFT</th>
<th>COCE</th>
<th>AFFN</th>
<th>MHT</th>
<th>COCE+MHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n logn)</td>
<td>O(n logn)</td>
<td></td>
</tr>
<tr>
<td>O(n (logn)^2)</td>
<td>O(n (logn)^2)</td>
<td>O(n (logn)^2)</td>
<td>O(n (logn)^2)</td>
<td>O(n (logn)^2)</td>
<td></td>
</tr>
</tbody>
</table>
Running Time

- COCE+MHT
- COCE
- BFT
- AFFN
- MHT

Time (ms)

Number of nodes

256 1024 4096 16384
Results: simple2D

Hops per byte

Number of nodes in communication graph

- Default
- BFT
- MHT
- COCE
- COCE+MHT
- AFFN
- PAIRS
Summary

• Contention in modern day supercomputers can impact performance: makes mapping important

• Certain classes of applications (latency sensitive, communication bound) benefit most
  • OpenAtom shows performance improvements of up to 50%
  • NAMD - improvements for > 4k cores

• Developing an automatic mapping framework
  • Relieve the application developer of the mapping burden
Summary

• Topology discovery: Topology Manager API
• Object Communication Graph: Profiling, Instrumentation
• Pattern matching
  • Regular graphs
  • Irregular graphs
• Suite of heuristics for mapping
• Distributed strategies with global view
Future Work

• More sophisticated algorithms for pattern matching and mapping
  • Multicast and many-to-many patterns

• Handling multiple communication graphs
  • Simultaneous or occurring in different phases

• Extension of the work on distributed load balancing
Thanks