Do we need a crystal ball for task migration?

Brandon {Myers,Holt}
University of Washington
bdmyers@cs.washington.edu
Large data sets
Spread data
Spread data

Data.0

Data.1

Data.2

Data.3
Resources: compute, bandwidth
Task migration

- move a running task to another node

- purpose:
  - increase utilization or manage resources
  - move task near tasks that share data
  - move task closer to data it will access

- costs:
  - moving local data required for the task to proceed
  - cpu time to stop and resume a task
Prior work

• task migration for:
  o efficient use of resources
  o load balancing

• thread placement on cache coherent systems using sharing information\(^1\)

• prediction for migration on NoC\(^2\)

1. F. Song et al. *Analytical modeling and optimization for affinity based thread scheduling on multicore systems*. CLUSTER ’09.
2. Chao Wang et al. *Packet Triggered Prediction Based Task Migration for Network-on-Chip*. 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Feb ’12
Non-uniform cost to access shared data
Exploit locality

node0

node1

node2

node3

Shared

PGAS

Local
load (<node1>)
node 0
load (<node1>)

node 1
MIGRATE

node 2

REMOTE

load (<node1>)
load (<node1>)
load (<node2>)
load (<node1>)

LOCAL
REMOTE
REMOTE
REMOTE
REMOTE
REMOTE
LOCAL
REMOTE
REMOTE
REMOTE
REMOTE
REMOTE
MIGRATE
MIGRATE
MIGRATE
MIGRATE
MIGRATE
MIGRATE

time
Question

• consider task migration as a prediction problem

• can we predict when it will be more efficient to move the data to the task, or move the task to the data?
Outline

- Motivation
- System model and cost metric
- Online migration predictors
- Evaluation
System model

• assumption: network is limiting resource
• simplification: flat network topology
  o only distinguish between local and remote shared memory
• cost metric: bytes transferred over the network
  o others are possible; this is enough to capture network usage
  o no timing required
Optimal task migration

- What is the best possible cost for a given execution?
- Find the schedule of migrations that minimizes bytes transferred
- Model excludes timing => schedules can be calculated independently for each task
Optimal schedule

node 2

node 1

node 0

load (<node1>)  load (<node1>)  load (<node2>)  load (<node1>)

time
Outline

- Motivation
- System model and cost metric
  - Online migration policies
- Evaluation
Online policies

• predict whether a migration will give benefit
• look at past access patterns
• similar to prefetch prediction in computer architecture
Migration predictors

history of past memory accesses
current node
target node
estimate of task size

predictor

{stay, migrate}
Stream Predictor policy

- influenced by *stream buffer* prefetch prediction
- migrate task when it has seen ‘enough’ references to the same node in the immediate past

Stream Predictor policy

- disadvantages of Stream:
  - do extra remote accesses before recognizing pattern
  - must do this every time
Hindsight Migrate policy

• insight:
  o same code may always have the same access pattern

• solution:
  o remember PCs that would have been good to migrate at
Hindsight: motivation

1. shared arrays[][
2. for particleArray in arrays:
3. totalWeight = 0
4. for p in particleArray:
5. totalWeight += p.weight
6. histogram[totalWeight]++

<table>
<thead>
<tr>
<th>PC</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>5</th>
<th>6.a</th>
<th>6.b</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>5</th>
<th>6.a</th>
<th>6.b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3...</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7...</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

---

Hindsight: motivation

1. shared arrays[][
2. for particleArray in arrays:
3. totalWeight = 0
4. for p in particleArray:
5. totalWeight += p.weight
6. histogram[totalWeight]++

<table>
<thead>
<tr>
<th>PC</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>5</th>
<th>6.a</th>
<th>6.b</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>5</th>
<th>6.a</th>
<th>6.b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3...</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7...</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Hindsight Migrate policy

If migrating at the first memory access in the window would have been worth it, add the PC to a migration set so the task can migrate next time.
Hindsight Migrate policy

<table>
<thead>
<tr>
<th>PC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Migration set
Hindsight Migrate policy

<table>
<thead>
<tr>
<th>PC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Migration set

4
Hindsight Migrate policy

<table>
<thead>
<tr>
<th>PC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Migration set

PC

4
Hindsight Migrate policy

<table>
<thead>
<tr>
<th>PC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

migrate? yes
Outline

• Motivation
• Simplified system model and cost metric
• Online migration policies
• Evaluation
Evaluation

• potential for task migration over no migration?
• how much of this can predictors achieve?

• procedure:
  o collect shared memory trace from program execution
  o simulate it in our model and measure total cost
  o run simulations with fixed task sizes

• benchmarks
  o NPB IntSort
  o PARSEC FluidAnimate
  o SSCA#2 Betweenness centrality
Simulation

1. annotate application code to choose a distribution for each shared memory allocation

2. collect shared memory trace for an execution

3. simulate:
   i. at each memory access, ask the policy whether the task should migrate
   ii. add the cost of the chosen action
IntSort

![Graph showing Bytes Transferred (normalized) vs. Task Size (bytes) for different strategies: Never, Optimal, Stream, and Hindsight.](image)
FluidAnimate

![Graph showing Bytes Transferred (normalized) vs Task Size (bytes) for different scenarios: Never, Optimal, Stream, Hindsight. The graph illustrates the increased bytes transferred as the task size increases, with Hindsight showing the highest increase.]
Betweenness Centrality

![Graph showing the relationship between task size and bytes transferred, with lines for Never, Optimal, Stream, and Hindsight.](image)
Results summary

• simple online predictors achieved up to 60% of optimal reduction in bytes transferred
• higher ratio of random access => lower potential for task migration to reduce network usage
Conclusion

• In this work:
  o task migration for reducing network usage, considered as a prediction problem

• Take-away:
  o migration predictors can make profitable choices based on past memory accesses
  o moving tasks to the data has the potential to improve performance of parallel applications if there is locality to exploit
A better cost metric

- message cost = \( \frac{\text{size}}{\text{BW}(\text{size})} \)

image: http://gasnet.cs.berkeley.edu/performance/
“Recoup rate”

![Graph showing recoup rate vs task size for different methods: Optimal, FA-Stream, FA-Hindsight, SSCA-Hindsight, SSCA-Stream.](image-url)
edgeData = (graphSDG *) malloc(sizeof(graphSDG));
track_memory(edgeData->startVertex, M, sizeof(VERT_T), BLOCK);
track_memory(edgeData->endVertex, M, sizeof(VERT_T), BLOCK);
track_memory(edgeData->weight, M, sizeof(WEIGHT_T), BLOCK);
BC = (double *) tm_malloc(N, sizeof(double), BLOCK);
elapsed_time = betweennessCentrality(G, BC);
tm_free(BC);
Instrumentation

• Use Pin to instrument the tracking functions and memory accesses

• On tracking functions
  o Update mapping of (address range) -> (allocation id)

• On each memory access
  o the callback looks up the access
  o If it is in a tracked region, save information about the access to trace file