Optimizing Memory Locality Using a Locality-Aware Page Table

Eduardo Cruz, Matthias Diener, Marco Alves, Laércio Pilla, Philippe Navaux

Presented by Francis Moreira
Introduction

- Memory hierarchy of parallel architectures
  - Data sharing → communication
  - Non Uniform Memory Access

- Locality-aware Thread Mapping
  - Map threads that communicate in cores close to each other in the memory hierarchy

- Locality-aware Data Mapping
  - Map memory pages to the NUMA node of the cores accessing them
Benefits of locality-aware mapping

- Locality-aware thread mapping
  - Less Cache Misses
    - Shared data in shared caches
  - Less traffic in interconnections
    - Less cache-to-cache transfers
    - Less cache line invalidations
- Locality-aware data mapping
  - Less accesses to remote NUMA nodes
  - Less traffic in interconnections
Overview

- Static mapping is not a good solution
  - Too many different architectures
  - Cannot handle changes on the behavior (between or during)

- Current Dynamic Mapping Mechanisms
  - Most perform only thread or data mapping alone
  - High trade-off between accuracy and overhead

- Our goal
  - Perform both thread+data dynamic mapping
  - Better trade-off between accuracy and overhead
  - How? Implement partial support in hardware
Virtual Memory
Virtual Memory

Application accesses memory
Virtual Memory

Application accesses memory

Entry in TLB?
Virtual Memory

Application accesses memory → Entry in TLB? → Evict old entry from TLB

No TLB miss
Virtual Memory

- Application accesses memory
- Entry in TLB?
- No TLB miss
- Evict old entry from TLB
- Fetch new TLB entry from page table
Virtual Memory

Application accesses memory

Entry in TLB?

No TLB miss

Evict old entry from TLB

Yes TLB hit

Perform address translation

Fetch new TLB entry from page table
Virtual Memory

Application accesses memory → Entry in TLB?

- No TLB miss → Evict old entry from TLB
- Yes TLB hit → Perform address translation:
  - Continue execution
  - Fetch new TLB entry from page table
LAPT
Locality-Aware Page Table
LAPT

Application accesses memory

Entry in TLB?

No TLB miss

Evict old entry from TLB

Yes TLB hit

Perform address translation

Fetch new TLB entry from page table

Continue execution
LAPT

Application accesses memory

Entry in TLB?

No TLB miss

Evict old entry from TLB

Yes TLB hit

Perform address translation

Continue execution

Fetch new TLB entry from page table

Update statistics of fetched entry
LAPT - added structures

- Page table entry
  - Add Communication Vector and NUMA Access Table

<table>
<thead>
<tr>
<th>Phys. Address</th>
<th>Communication Vector (CV)</th>
<th>NUMA Access Table (NAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>T0</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>T3</td>
</tr>
<tr>
<td></td>
<td>N0</td>
<td>N1</td>
</tr>
<tr>
<td></td>
<td>N2</td>
<td>N3</td>
</tr>
</tbody>
</table>

- Communication Matrix (CM)
  - Amount of communication between threads

```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
```
LAPT - Overview

Hardware

Software
LAPT - Overview

Fetch TLB entry along with Comm. Vector

Hardware

Software
LAPT - Overview

- Fetch TLB entry along with Comm. Vector
- Update Communication Matrix
- Update Communication Vector

Hardware

Software
LAPT - Overview

**Hardware**
- Fetch TLB entry along with Comm. Vector
- Update Communication Matrix
- Update Communication Vector

**Software**
- Fetch Communication Matrix
LAPT - Overview

**Hardware**

1. Fetch TLB entry along with Comm. Vector
2. Update Communication Matrix
3. Update Communication Vector

**Software**

1. Fetch Communication Matrix
2. Perform Thread Mapping
LAPT - Overview

**Hardware**
- Fetch TLB entry along with Comm. Vector
- Update Communication Matrix
- Update Communication Vector

**Software**
- Fetch Communication Matrix
- Perform Thread Mapping
- Update NAT
LAPT - Overview

**Hardware**

- Fetch TLB entry along with Comm. Vector
- Update Communication Matrix
- Update Communication Vector

**Software**

- Fetch Communication Matrix
- Perform Thread Mapping
- Update NAT
- Perform Data Mapping
LAPT - example - 6 threads, 4 nodes
LAPT - example - 6 threads, 4 nodes

Thread 3 causes TLB miss on page P
LAPT - example - 6 threads, 4 nodes

Thread 3 causes TLB miss on page P

Update Communication Matrix in row 3
Thread 3 causes TLB miss on page P

Update Communication Matrix in row 3

LAPT - example - 6 threads, 4 nodes
LAPT - example - 6 threads, 4 nodes

Update Communication Vector of page P

Thread 3 causes TLB miss on page P

Update Communication Matrix in row 3

Comm. Vector
0 1 2 4

Communication Matrix

Hardware  | Data structures  | Software
LAPT - example - 6 threads, 4 nodes

- Thread 3 causes TLB miss on page P
- Update Communication Vector of page P
- Update Communication Matrix in row 3

Communication Matrix:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comm. Vector:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LAPT - example - 6 threads, 4 nodes

Thread 3 causes TLB miss on page P

Update Communication Vector of page P

Update Communication Matrix in row 3

Communication Matrix

Comm. Vector

0 1 2 4
3 0 1 2

Fetch Communication Matrix

Software

Data structures

Hardware

Communication Matrix in row 3

+ + + +
LAPT - example - 6 threads, 4 nodes

- Thread 3 causes TLB miss on page P
- Update Communication Vector of page P
- Update Communication Matrix in row 3

Communication Matrix:

```
0 1 2 3 4 5
0 + + + +   0
1          1
2          2
3          3
4          4
5          5
```

Communication Vector:

```
0 1 2 3 4 5
0 1 2 4 3 0
```

Fetch Communication Matrix

Perform Thread Mapping:

0->0 | 1,2,3->1
LAPT - example - 6 threads, 4 nodes

Update Communication Vector of page P
Thread 3 causes TLB miss on page P
Update Communication Matrix in row 3

Comm. Vector
0 1 2 4
3 0 1 2

Communication Matrix

Fetch Communication Matrix
Perform Thread Mapping 0->0| 1,2,3->1

Hardware

Data structures

Software

NAT 0 0 0 0
LAPT - example - 6 threads, 4 nodes

- Thread 3 causes TLB miss on page P
- Update Communication Vector of page P
- Update Communication Matrix in row 3

Communication Matrix:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Update Communication Vector:

0 1 2 4
3 0 1 2

Update NAT

Fetch Communication Matrix

Perform Thread Mapping:

0->0, 1, 2, 3->1

Hardware

Data structures

Software
LAPT - example - 6 threads, 4 nodes

Update Communication Vector of page P
Thread 3 causes TLB miss on page P
Update Communication Matrix in row 3

Communication Matrix

Comm. Vector

0 1 2 4
3 0 1 2

Fetch Communication Matrix
Perform Thread Mapping
0->0 | 1,2,3->1

Update NAT

NAT

1 3 0 0

Data structures
LAPT - example - 6 threads, 4 nodes

- Thread 3 causes TLB miss on page P
- Update Communication Vector of page P
- Update Communication Matrix in row 3
- Communication Matrix
- NAT
- Data structures
- Fetch Communication Matrix
- Perform Thread Mapping: 0->0 | 1,2,3->1
- Update NAT
- Perform Data Mapping
- Software
- Hardware
Evaluation of LAPT
Methodology

- Performance evaluation in a real machine
  - 2 NUMA nodes
  - One Intel Sandy Bridge Xeon E5-2650 per node

- LAPT implemented in Pin DBI tool
  - Generate mapping information for execution on real machine

- Benchmarks
  - NAS Parallel Benchmarks (OpenMP version)

- Measurements
  - Execution Time, L3 cache MPKI, QPI traffic, Energy per Instruction
  - Mappings: OS (baseline), LAPT, Oracle, Random
Results

Normalized Execution Time (%)
Lower is better

Random Mapping  Oracle Mapping  LAPT

BT  CG  EP  FT  IS  LU  MG  SP  UA  Avg
Most applications benefit from mapping. The amount of benefit depends on the application’s behavior.
Results

BT, CG, LU, SP and UA are large and complex applications with high memory usage. Highest improvements are expected.

Random Mapping
Oracle Mapping
LAPT

Normalized Execution Time (%)

Lower is better
Results

Normalized Execution Time (%)
Lower is better

EP almost does not communicate. No performance improvement is expected from thread mapping.
Results

FT, IS and MG have high memory usage but low execution time. Less improvements are expected.
Results

Normalized L3 MPKI (%)

Lower is better

-60% -50% -40% -30% -20% -10% 0% 10% 20%

Random Mapping Oracle Mapping LAPT

BT CG EP FT IS LU MG SP UA Avg
Results

Mostly influenced by thread mapping.

Normalized L3 MPKI (%) (Lower is better)

<table>
<thead>
<tr>
<th></th>
<th>BT</th>
<th>CG</th>
<th>EP</th>
<th>FT</th>
<th>IS</th>
<th>LU</th>
<th>MG</th>
<th>SP</th>
<th>UA</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random Mapping</td>
<td>Oracle Mapping</td>
<td>LAPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results

Normalized L3 MPKI (%)
Lower is better

More communication within sub-groups of threads. Thread mapping reduces cache misses.
Results

Normalized L3 MPKI (%)
Lower is better

Communication Pattern of CG is all-to-all.
No thread mapping reduces cache misses.
Results

Normalized QPI traffic (%)
Lower is better

Random Mapping  Oracle Mapping  LAPT

BT  CG  EP  FT  IS  LU  MG  SP  UA  Avg
Results

Mostly influenced by data mapping.

Normalized QPI traffic (%)
Lower is better

Random Mapping
Oracle Mapping
LAPT
Results

Normalized QPI traffic (%)
Lower is better

Although communication in CG is all-to-all, its threads still access private data. Data mapping reduces QPI traffic.
Results

Normalized Energy per Instruction (%)
Lower is better
Results

Locality-aware mapping also reduces energy consumption.

Normalized Energy per Instruction (%)  
Lower is better

-15%  -10%  -5%  0%  5%  10%  15%

BT  CG  EP  FT  IS  LU  MG  SP  UA  Avg

Random Mapping  Oracle Mapping  LAPT
Results

Reduction of energy is similar to reduction of execution time.

Normalized Energy per Instruction (%)

Lower is better

Random Mapping  Oracle Mapping  LAPT
Results

Reduction of Energy per Instruction shows that we obtain a more efficient execution.

Normalized Energy per Instruction (%)
Lower is better
Conclusions
Conclusions

- Locality-aware mapping is important
  - Less cache misses
  - Less remote memory accesses

- LAPT
  - Partial support in hardware
  - Better trade-off between accuracy and overhead

- Results
  - Performance improvements (avg: 6.7%)
  - Energy consumption improvements (avg: 5.3%)
  - Low overhead (less than 1% of execution time)
Thanks

Questions???

ehmcruz@inf.ufrgs.br