Node-Based Memory Management for Scalable NUMA Architectures

International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2012)

Stefan Lankes¹, Thomas Roehl², Christian Terboven², Thomas Bemmerl¹

¹Chair for Operating Systems, RWTH Aachen University
²Center for Computing and Communication, RWTH Aachen University
Outline

- Motivation
- Illustration of a common memory management
- Design of the node-based memory management
- Critical analysis
- Future prospects
- Benchmark results
- Conclusions and outlook
Scalable NUMA Interconnect

Node 1
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 2
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 3
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 4
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 5
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 6
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 7
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 8
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 9
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 10
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 11
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 12
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 13
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 14
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 15
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Node 16
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory
Core
Core
Memory

Storage Subsystem

Scalable NUMA Interconnect

Network Interface
Performance Characteristics (NumaScale-Cluster)

- 2 systems with 2 AMD QuadCores of type 8378 combined via NumaConnect
- all data on node 0
Eight-Socket Configuration (Westmere-EX)
Performance Characteristics (Westemere-EX)

- 8 Intel Xeon CPU E7-8850 (Westmere-EX)
- 8 * 10 Cores / 8 * 20 Cores via HyperThreading
- all data on node 0
Common Memory Management
Process/thread creation

1st level page table

First memory access!
Page fault!

2nd level page table

Node 0

MEM0

CPU0

Node n

MEMn

CPUn

- Pointer to the 1st level page table is also part of the process control block
- All threads use the identical address space
  → Same entry point on all cores

Interconnect
Page Table per Node

Basic idea

Node 0

1st level page table

2nd level page table

MEM0

CPU0

Interconnect

Node n

1st level page table

2nd level page table

MEMn

CPU0

First memory access!
Page fault!
Page Table per Node
Replication of read-only regions

Node 0

1st level page table

2nd level page table

MEM0

CPU0

Node n

1st level page table

2nd level page table

MEMn

CPU n

First memory access!
Page fault!

Interconnect

Page Replication

RO

RO
Advantages & Disadvantages

■ Pro:
  ➔ Reflecting actual hardware at mapping layer
  ➔ After duplication only accesses to local memory
  ➔ Easy preparation of applications to use \texttt{mprotect()}

■ Contra:
  ➔ Memory overhead
    » One page table per NUMA node
    » Duplicated pages
  ➔ Replication time
  ➔ Searching for mappings at all NUMA nodes
    (\texttt{page fault, mprotect()}, \texttt{free()})
Avoid PGT-Traversal at Mapping Search

- **Current Approach**
  - Searching for mappings at all NUMA nodes
  - On which node should we start?

- **Under development**
  - Use node-distance based search
    - Does not guarantee less work
  - Add new management structure
    - Derived page table stores virtual address-to-nodemask mappings
    - Needs 2 page table traversals per search,
    - First resolve location, then address
    - Increases memory footprint
Detection of Performance Issues

- Page tables include access/dirty bits to record memory accesses.
  → Usable to detect performance issues?
Common usage of the access / dirty bits

- Normally used to realize demand paging.
  - Approximation of Least Recently Used (LRU)
  - Classical concept
    - Managing of two lists of active and inactive page frames
    - State transition realized via access bits
    - Doubling the number of accesses via a reference bit to move pages from the inactive to active list.
Transfer to the Node-based Memory Management

- Usage of two reference bits
  - One to signalize local and one to signalize remote memory accesses

- Abstract of the new state graph

```
inactive
referenced=00
remote access

inactive
referenced=01
no access

active
referenced=00

active
referenced=01
performance issue
```
Jacobi solver as Application Benchmark

- Solving of $A \cdot x = b$, $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^{n}$, $x \in \mathbb{R}^{n}$
- Iterative rule:
  \[ x_{i}^{m+1} = \frac{1}{a_{i,i}} \left( b_{i} - \sum_{j \neq i} a_{i,j} x_{j}^{m} \right) \]
- Abstract code for the new memory management (sequential) initialization of $A$, $b$ and $x_0$
  - forbid write access to $A$ and $b$
  - while(!found_solution)
    - parallel for over the iterative rule
  - allow write access to $A$ and $b$

- Straightforward implementation
Jacobi solver as Application Benchmark

- Solving of $A \cdot x = b$, $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $x \in \mathbb{R}^n$

- Iterative rule:

$$x_{i}^{m+1} = \frac{1}{a_{i,i}} \left( b_{i} - \sum_{j \neq i} a_{i,j} x_{j}^{m} \right)$$

- Abstract code:

(sequential) initialization of $A$, $b$ and $x_0$

forbid write access to $A$ and $b$

while(!found_solution)

parallel for over the iterative rule

allow write access to $A$ and $b$
Jacobi solver as Application Benchmark

- Solving of \( A \cdot x = b \), \( A \in R^{n \times n} \), \( b \in R^n \), \( x \in R^n \)
- Iterative rule:
  \[
  x_i^{m+1} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j \neq i} a_{i,j} x_j^m \right)
  \]
- Abstract code
  (sequential) initialization of \( A \), \( b \) and \( x_0 \)
  forbid write access to \( A \) and \( b \) thread binding
  while(!found_solution)
  parallel for over the iterative rule
  allow write access to \( A \) and \( b \)
Jacobi solver as Application Benchmark

- Solving of $A \cdot x = b$, $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $x \in \mathbb{R}^n$
- Iterative rule:
  \[
  x^{m+1}_i = \frac{1}{a_{i,i}} \left( b_i - \sum_{j \neq i} a_{i,j} x^m_j \right)
  \]
- Abstract code
  
  (sequential/ideal) initialization of $A$, $b$ and $x_0$
  
  forbid write access to $A$ and $b$
  
  thread binding
  
  while(!found_solution)
    
    parallel for over the iterative rule
  
  allow write access to $A$ and $b$
Jacobi solver (Westmere-EX)

usage of a page table per node
- pinned threads, ideal initialization
- pinned threads, seq. initialization
- no pinned threads, seq. initialization

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
<th>140</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>no pinned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>threads, seq.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initialization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pinned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>threads, seq.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initialization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pinned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>threads, ideal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initialization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>usage of a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page table per</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 80 threads: 144,609, 69,247, 33,543, 44,77
- 160 threads: 91,067, 62,864, 27,517, 27,746

matrix size: 5120 x 5120
iterations: 20000
Conclusions and Outlook

- Memory management can reflect the actual hardware
- First performance results are promising
- Reduction of overhead by
  - usage of virtual address-to-node mapping
  - bundling of NUMA nodes
- Introduce possibilities to detect performance issues
- Simple integration into existing programming models

```c
#pragma omp parallel for shared(A,B,C) readonly(A,B)
for (i=0; i<0; i++)
    C[i] = A[i] + B[i];
```
Thank you for your kind attention!

Stefan Lankes

Chair for Operating Systems
RWTH Aachen University
Kopernikusstr. 16
52056 Aachen, Germany

www.lfbs.rwth-aachen.de
contact@lfbs.rwth-aachen.de
Backup slides
Related Work

- Page placement strategies are extensively investigated
  - Page placement via hints
    » Affinity-On-Next-Touch
      - Proposals: Nordergraaf & van der Pas
      - Variations: Shermerhorn, Goglin et al., Bircsak at al.
    » Template library of locality management (Majo & Gross)
  - (Semi)automatic page placement
    » profile-guided automatic page placement (Mueller et al.)
    » dynamic page migration via counting remote memory accesses
      - Memory controller extensions: SGI Origin
      - Compiler extensions: Nikolopoulos et al.

- However, it exists room for optimizations.
Page Table per Node

Basic idea

- One page table per node
- Context switch: Load node-local page table
- Page fault
  - Page not mapped: allocate new page and map locally
  - Page mapped remotely:
    » RW page: duplicate mapping
    » RO page: duplicate page and map clone locally

- New system call to create a process, which uses our node-based memory management,
  - Per default, the processes use the traditional concept.
- Via `mprotect` the page replication could be implicitly en- or disabled for certain memory regions.
Overhead (Westmere-EX)

<table>
<thead>
<tr>
<th></th>
<th>unmodified Linux kernel (3.3.8)</th>
<th>page table per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>time to allocate a page</td>
<td>1.666µs</td>
<td>6.671µs</td>
</tr>
<tr>
<td>time to protect a page</td>
<td>0.00005µs</td>
<td>0.032µs</td>
</tr>
<tr>
<td>time to replicate a page</td>
<td></td>
<td>4.479µs</td>
</tr>
<tr>
<td>time to unprotect a page</td>
<td>0.0001µs</td>
<td>0.148µs</td>
</tr>
<tr>
<td>time to replicate a reference</td>
<td></td>
<td>1.445µs</td>
</tr>
</tbody>
</table>

Test platform
- 8 Intel Xeon CPU E7-8850 (Westmere-EX)
- 8 * 10 Cores / 8 * 20 Cores via HyperThreading
## Overhead (NumaScale-Cluster)

<table>
<thead>
<tr>
<th></th>
<th>unmodified Linux kernel (2.6.37)</th>
<th>page table per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>time to allocate a page</td>
<td>2.810µs</td>
<td>3.143µs</td>
</tr>
<tr>
<td>time to protect a page</td>
<td>0.034µs</td>
<td>0.110µs</td>
</tr>
<tr>
<td>time to replicate a page</td>
<td></td>
<td>26.956µs</td>
</tr>
<tr>
<td>time to unprotect a page</td>
<td>0.195µs</td>
<td>2.787µs</td>
</tr>
<tr>
<td>time to replicate a reference</td>
<td></td>
<td>6.044µs</td>
</tr>
</tbody>
</table>

### Test platform
- 2 systems with 2 AMD QuadCores of type 8378 combined via NumaConnect
Jacobi solver (NumaScale-Cluster)

- no pinned threads, seq. Initialization
- pinned threads, seq. initialization
- pinned threads, par. initialization
- usage of a page table per node

Time [s] vs. Number of threads

- Matrix size: 3072 x 3072
- Iterations: 20000