Smelt:
Machine-aware Atomic Broadcast Trees for Multicores

Stefan Kaestle, Reto Achermann, Roni Haecki, Moritz Hoffmann, Sabela Ramos, Timothy Roscoe

Systems Group, Department of Computer Science, ETH Zurich
Large number of trees: topologies and send orders

- Tree Topology?
- Number of cross NUMA links?
- Root of the tree?
- Send order of messages?
- Maximum out degree?
Large number of trees: topologies and send orders

Number of cross NUMA links?

Root of the tree?

Tree Topology?

Large number of possible trees

$$n! \binom{n-1}{n-1} = \frac{(2n - 2)!}{(n - 1)!}$$

topology + message send order

Send order of messages?

Maximum out degree?
There is no globally optimal tree structure

AMD Interlagos (4 Socket x 4 Cores x 2 Threads)

Cluster topology wins

Execution Time [kCycles]

0 5 10 15 20 25 30 35 40

Barrier  Reduction  Broadcast  2PC

Binary Tree  Cluster  Sequential

Intel Xeon Phi (61 Cores x 1 Thread)

Binary tree or Fibonacci win

Execution Time [kCycles]

0 10 20 30 40 50 60 70 80 90 100 110 120

Barrier  Reduction  Broadcast  2PC

Binary Tree  Cluster  Sequential  MST  Fibonacci
Smelt: Automatic optimization of broadcast and reduction trees
Example: Building fast and simple barriers

Barrier Benchmark on Intel Sandy Bridge 4x8x2

Execution Time [kCycles]

<table>
<thead>
<tr>
<th>Barrier implementation</th>
<th>Dissemination</th>
<th>Parlib MCS</th>
<th>Smelt</th>
</tr>
</thead>
<tbody>
<tr>
<td>3x</td>
<td>7x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dramatic improvement through automatic optimization of communication patterns
Broadcasts and reductions are central building blocks for parallel programs

<table>
<thead>
<tr>
<th>Performance</th>
<th>Fault-Tolerance</th>
<th>Execution Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic broadcasts</td>
<td>Agreement protocols, atomic broadcasts</td>
<td>Reductions, broadcast, barriers</td>
</tr>
<tr>
<td>Replication for data locality</td>
<td>Replication for failure resilience</td>
<td>Thread synchronization, data gathering</td>
</tr>
<tr>
<td>e.g. Shoal, Carrefour, SMMP OS, FOS</td>
<td>e.g. 1Paxos</td>
<td>e.g. OpenMP</td>
</tr>
</tbody>
</table>
Multicore hardware is complex

Hierarchical:
- thread/cores
- caches/memory

Hardware parallelism

Interconnect / coherence
- protocol complexity

Distributed resources

Intel Sandy Bridge 4x8x2
Multicore hardware is complex

Hierarchical:
- thread/cores
- caches/memory

Generate a good tree topology and schedules automatically

Intel Sandy Bridge 4x8x2

Hardware parallelism

Interconnect / coherence protocol complexity

Distributed resources
Smelt is based on peer-to-peer message passing

- Works well for our approach.

- Clear concept: Enables **reasoning** about send and receive costs
Message-passing on multicores is different

Classical Network

- Machine 1
- Machine 2
- Machine 3
- Machine 4

Multicore interconnect

- Core 1
- Core 2
- Core 3
- Core 4

$t_{send}$, $t_{propagate}$, $t_{receive}$

Time [1us]

Time [10ns]
Message-passing on multicores is different

Classical Network

Machine 4
Machine 3
Machine 2

Multicore interconnect

Core 4
Core 3
Core 2

On multicores send and receive times dominate propagation time

Goal: Minimize total time of the broadcast
Minimizing the total time of a broadcast

- \( t_{\text{broadcast}} = t_{\text{last}} - t_{\text{start}} \)
- Minimize the longest path from the root to the leaves.

\[
t_{\text{path}} = \sum (t_{\text{send}} + t_{\text{receive}})
\]

We need to know the send and receive cost between any pair of cores
Information obtained from hardware discovery

```
$ lscpu
CPU(s):                64
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             4
NUMA node(s):          8
L1d cache:             16K
L1i cache:             64K
L2 cache:              2048K
L3 cache:              6144K
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28
NUMA node1 CPU(s):     32,36,40,44,48,52,56,60
NUMA node2 CPU(s):     2,6,10,14,18,22,26,30
NUMA node3 CPU(s):     34,38,42,46,50,54,58,62
NUMA node4 CPU(s):     3,7,11,15,19,23,27,31
NUMA node5 CPU(s):     35,39,43,47,51,55,59,63
NUMA node6 CPU(s):     1,5,9,13,17,21,25,29
NUMA node7 CPU(s):     33,37,41,45,49,53,57,61
```

```
$ numactl --hardware
node distances:
	node  0  1  2  3  4  5  6  7

0:  10 16 16 22 16 22 16 22
1:  16 10 22 16 16 22 16 22
2:  16 22 10 16 16 16 16 16
3:  22 16 16 10 16 16 22 22
4:  16 16 16 10 16 16 16 22
5:  22 22 16 16 16 10 22 16
6:  16 22 16 22 16 22 10 16
7:  22 16 16 22 22 16 16 10
```

AMD Interlagos 4x4x2
Information obtained from hardware discovery

$ lscpu

CPU(s):                64
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             4
NUMA node(s):          8
L1d cache:             16K
L1i cache:             64K
L2 cache:              2048K
L3 cache:              6144K
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28
NUMA node1 CPU(s):     32,36,40,44,48,52,56,60
NUMA node2 CPU(s):     2,6,10,14,18,22,26,30
NUMA node3 CPU(s):     34,38,42,46,50,54,58,62
NUMA node4 CPU(s):     3,7,11,15,19,23,27,31
NUMA node5 CPU(s):     35,39,43,47,51,55,59,63
NUMA node6 CPU(s):     1,5,9,13,17,21,25,29
NUMA node7 CPU(s):     33,37,41,45,49,53,57,61

$ numactl -hardware
node distances:

<table>
<thead>
<tr>
<th>node</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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>22</td>
<td>16</td>
<td>22</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>10</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>22</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>22</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>10</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>22</td>
<td>16</td>
<td>22</td>
<td>16</td>
<td>22</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>22</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>10</td>
</tr>
</tbody>
</table>

Symmetric: $A \rightarrow B = B \rightarrow A$

Doesn’t distinguish between $send()$ and $recv()$

AMD Interlagos 4x4x2

NUMA distance: abstract value
Complement with microbenchmarks: pairwise send and receive

<table>
<thead>
<tr>
<th>Sending Core</th>
<th>Receiving Core</th>
<th>Cost of Send Operation [Cycles]</th>
<th>Cost of Receive Operation [cycles]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>min = 43</td>
<td>min = 43</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>max = 166</td>
<td>max = 360</td>
</tr>
</tbody>
</table>

AMD Interlagos 4x4x2
Complement with microbenchmarks: pairwise send and receive

Cost of Send Operation [Cycles]

- Receiving core: min = 43, max = 166
- Sending core: not symmetric

Not symmetric!

Core 10 \(\rightarrow\) Core 22: \(137 + 282 = 419\) cycles
Core 22 \(\rightarrow\) Core 10: \(159 + 351 = 510\) cycles

AMD Interlagos 4x4x2
Smelt
Using Smelt for group communication

```c
#include <smelt/smelt.h>

void main()
{
    smelt_init();
    smelt_topology_create();
    smelt_broadcast(msg);
}
```
Smelt’s tree generator heuristics

- **Remote Cores First**: Prioritize tasks that involve remote cores.
- **Avoid Expensive Communication**: Minimize communication between nodes.
- **Maximize Parallelism**: Ensure tasks are evenly distributed for parallel processing.
- **No redundancy**: Avoid redundant computations to optimize resources.
Step 1: Pick the root Core with the smallest total send time to any other core.

Heuristic #1: Expensive first

Step 2: Start Scheduler Schedule cores to send and receive messages

Apply Heuristics Decide where to send messages to

The tree is good, But not optimal!
Smelt Tree for Intel Xeon Phi using 61 cores

Smelt Tree for Intel Sandy Bridge 4 Sockets x 8 Cores x 2 Threads
# Evaluation Testbed

## Intel

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Sockets</th>
<th>Cores / Socket</th>
<th>Threads / Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ivy Bridge</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Nehalem</td>
<td>4</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Knights Corner</td>
<td>1</td>
<td>61</td>
<td>4</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>4</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Bloomfield</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

## AMD

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Sockets</th>
<th>Cores / Socket</th>
<th>Threads / Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magny Cours</td>
<td>4</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Barcelona</td>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Shanghai</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Interlagos</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Istanbul</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Full set of results online.

[http://machinedb.systems.ethz.ch](http://machinedb.systems.ethz.ch)
Smelt produces good trees across architectures

**AMD Interlagos (4 Socket x 4 Threads)**

- Execution Time [kCycles]
- Best other = Cluster

**Intel Xeon Phi (61 Threads)**

- Execution Time [kCycles]
- Best other = Fibonacci / Binary Tree
**Smelt produces good trees across architectures**

<table>
<thead>
<tr>
<th>Operation</th>
<th>KNC</th>
<th>BF</th>
<th>SB</th>
<th>IB</th>
<th>SH</th>
<th>IS</th>
<th>IL</th>
<th>MC</th>
<th>NL</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>broadcast</strong></td>
<td>1.24</td>
<td>1.06</td>
<td>1.11</td>
<td>1.37</td>
<td>1.13</td>
<td>1.10</td>
<td>1.16</td>
<td>1.15</td>
<td>1.07</td>
<td>1.22</td>
</tr>
<tr>
<td><strong>barrier</strong></td>
<td>1.12</td>
<td>1.07</td>
<td>1.30</td>
<td>1.41</td>
<td>1.09</td>
<td>1.08</td>
<td>1.13</td>
<td>1.03</td>
<td>1.09</td>
<td>1.38</td>
</tr>
<tr>
<td><strong>2PC</strong></td>
<td>1.17</td>
<td>1.09</td>
<td>1.22</td>
<td>1.35</td>
<td>1.10</td>
<td>1.13</td>
<td>1.11</td>
<td>1.07</td>
<td>1.17</td>
<td>1.33</td>
</tr>
<tr>
<td><strong>reduction</strong></td>
<td>1.18</td>
<td>1.08</td>
<td>1.27</td>
<td>1.24</td>
<td>1.01</td>
<td>1.24</td>
<td>1.09</td>
<td>1.18</td>
<td>1.53</td>
<td>1.21</td>
</tr>
</tbody>
</table>
Fast broadcast trees are good for reductions in most cases

Smelt Tree on Intel Bloomfield 2x4x2

Cluster Topology on Intel Bloomfield 2x4x2

Additional Cross-NUMA link
Smelt provides simple and fast barriers

Barrier Benchmark on Intel Sandy Bridge 4x8x2

Execution Time [kCycles]

- **Dissemination**
- **Parlib MSC**
- **Smelt**

Barriers based on reduction and broadcast

```c
void smelt_barrier(void) {
    smelt_reduce();
    smelt_broadcast();
}
```

Simple barrier implementation
OpenMP: EPCC OpenMP Benchmark Collection

```c
/* epcc openmp barrier benchmark */
void testbar() {
    int j;
    #pragma omp parallel private(j)
    {
        for (j = 0; j < innerreps; j++) {
            delay(delaylength);
            #pragma omp barrier
        }
    }
    #pragma omp barrier
}
```

**Implicit barrier at the end of parallel block**

**Explicit barrier**

Replaced GOMP barrier with Smelt

Remaining results on the website
Agreement Protocols: 1Paxos

4 clients to generate load
N replicas executing 1Paxos

1Paxos Benchmark on AMD Interlagos 4x4x2

Execution Time [kCycles]
Summary

- Broadcasts and reductions are **central building blocks**
- **No globally optimal** tree topology
- Information from hardware discovery is **not sufficient**
- Smelt’s produces **good trees**

machinedb.systems.ethz.ch  github.com/libsmelt