System techniques to mitigate the NUMA effect

Gaël Thomas, professor at Telecom SudParis
We need computing power

- To analyze large datasets
- To perform large computation
- To handle many clients
The computing power is in the CPU
(Old) computing power trends

Moore’s low: #transistors x2 each 1.5 year

# transistors (thousands)
Single step Perf (SpecINT)
Frequency (MHz) ⇒ processing power
Typical (electrical) power (Watts)

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore
But \( \uparrow \) frequency \( \Rightarrow \) \( \uparrow \) electrical power

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore
Fortunately, the Moore’s law still hold
Today: ↗ power by increasing #cores

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore
But programming a multicore is hard

#include <stdlib.h>

#define N 100000000

int main(int argc, char **argv) {
    int* a = malloc(sizeof(int) * N);
    for(int i=1; i<N; i++) {
        a[i] = a[i] * a[i-1];
    }
}

On my laptop at 2k€
(2 cores at 2.2GHz)
$ time ./bip
real 0m0.474s

On my server at 15k€
(48 cores at 2.2GHz)
$ time ./bip
real 0m1.142s

Not really what we can expect
Multicores radically change the way we design applications

- We have to parallelize our applications
Multicores radically change the way we design applications

- We have to parallelize our applications
- And our parallel algorithms have to scale
Multicores radically change the way we design applications

- We have to parallelize our applications
- And our parallel algorithms have to scale

But that’s not enough…

We have to handle complex memory architectures
A multicore has a complex architecture

Because a single memory bus becomes a bottleneck when we increase the number of cores
A multicore has a complex architecture

Memory access is non uniform
A multicore has a complex architecture

Transparent for the software, because the hardware transparently routes a memory request to the appropriate node.
A node has also a complex cache hierarchy
The machine used for our experiment

Total: 48 cores and 256GiB of RAM

- 8 nodes, max distance 2 hops
  - 6 cores/nodes
  - 16 GB/nodes

- Caches
  - Each node has a shared L3 cache of 5MiB
  - Each core has 2*64KiB of L1 cache and 512KiB of L2 cache

- Bandwidths
  - Memory controller: 13 GB/s
  - Interconnect link: up to 6 GB/s (asymmetric architecture)
The machine used for our experiment

Latencies

- ~ 5 cycles
- ~ 15 cycles
- ~ 50 cycles
- ~ 150 to 380 cycles

Local memory: 155 cycles
Remote memory (1 hop): 275 cycles
Remote memory (2 hops): 380 cycles
The machine used for our experiment

Worst case: all the cores access the same memory node

Up to 870 cycles (~6 times a local access)
NUMA matters

- Abstract cache-unfriendly application
  - 50% of the instructions access memory
  - 30% of the accesses in L1 cache
  - 30% of the accesses in L2 cache
  - 30% of the accesses in L3 cache

- Comparison between best and worst NUMA placements
  - Best: all accesses to local node ⇒ ~ 32 cycles/insn
  - Worst: all accesses to an overloaded node ⇒ ~ 156 cycles/insn
  ⇒ overhead of 385% in the worst case
NUMA matters

- Abstract cache-friendly application
  - 50% of the instructions access memory
  - 70% of the accesses in L1 cache
  - 70% of the accesses in L2 cache
  - 70% of the accesses in L3 cache

- Comparison between best and worst NUMA placements
  - Best: all accesses to local node ⇒ ~ 7 cycles/insn
  - Worst: all accesses to an overloaded node ⇒ ~ 17 cycles/insn
    ⇒ overhead of 137% in the worst case
First study

- **Goal:**
  - Understand how Linux manages NUMA
  - Understand how applications react to NUMA

- **How:**
  - Study a panel of 29 applications from 5 benchmarks (NPB, Parsec, Mosbench, X-stream, YCSB)
  - Evaluate various NUMA management policies
NUMA placement policies

- Defines from which NUMA node memory is allocated
NUMA placement policies

- Defines from which NUMA node memory is allocated

Leverage the page table:
maps a virtual address to a specific NUMA node by
mapping the virtual address to a page that belongs to the node
The hand-tuned policy

- Manually place the memory address ranges on the nodes
The hand-tuned policy

- Manually place the memory address ranges on the nodes
  + Tune the memory placement for an application
    - A lot of engineering effort for only a single application/hardware
The hand-tuned policy on Linux

- Hand-tuned thread placement
  - setaffinity(set of cores): for all the threads of a process
  - pthread_setaffinity(set of cores): for a single thread

- Hand-tuned memory placement
  - mbind(virtual address range, set of nodes)
    (granularity of a 4k-page)
The interleaved policy

- Round-robin from all the nodes
The interleaved policy

- Round-robin from all the nodes
  - Balance the load on all the nodes ⇒ no overloaded node
  - Many remote accesses ⇒ interconnect can saturate
The first-touch policy

- From the node that triggers the first access
  - Relies on the lazy mapping used in Linux

![Diagram showing the first-touch policy with a thread running on node 1 and a virtual address space of a process with page table and not-yet-mapped pages.]
The first-touch policy

- From the node that triggers the first access
  - Relies on the lazy mapping used in Linux
The first-touch policy

- From the node that triggers the first access
  - Relies on the lazy mapping used in Linux
The first-touch policy

- From the node that triggers the first access
  + Perfect locality and no saturation if a thread accesses its memory
  - Overloaded nodes if some threads allocate for the others
The Carrefour policy

- Proposed by Dashti et al. (ASPLOS’15)
  - Rebalance the load on all the nodes
  - Prevents the contention of the interconnect

- Dynamically migrate a page
  - From contended to uncontended nodes in case of contented node
  - On the node that uses the page in case of contended interconnect
The Carrefour policy

- Proposed by Dashti et al. (ASPLOS’15)
  - Rebalance the load on all the nodes
  - Prevents the contention of the interconnect

- Dynamically migrate a page
  - From contended to uncontended nodes in case of contended node
  - On the node that uses the page in case of contended interconnect

  + Improves locality and avoid contention in many cases
  - Can lead to inefficient placements for applications with different access patterns during the run
Evaluated policies

- Four combinations
  - First-touch (Linux FT)
  - First-touch with Carrefour (Linux FT/Carrefour)
  - Interleaved (Linux 4K)
  - Interleaved with Carrefour (Linux 4K/Carrefour)

- Only considers pages of 4KiB
Evaluation of the NUMA policies

Speedup relative to Linux FT

[presented at Eurosyst'17]
Evaluation of the NUMA policies

First conclusion

All the NUMA policies are important

Each application needs its own NUMA policy

Speedup relative to Linux FT
Second study

- Predict which NUMA policy is the best for an application

Goal:
- Select the most efficient NUMA policy
- Understand the memory access behavior
Predict the NUMA policy

- Measure the memory access imbalance with first-touch
  Relative standard deviation around the average #accesses per node

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Perfect balance

All the accesses go to a single node
Predict the NUMA policy

- Measure the memory access imbalance with first-touch
  Relative standard deviation around the average #accesses per node

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Low imbalance
High imbalance
Moderate imbalance
Predict the NUMA policy

Low imbalance with first-touch
Often because we already have a good locality

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Low imbalance  Moderate imbalance  High imbalance
Predict the NUMA policy

**Low imbalance with first-touch**

Often because we already have a good locality

=> keep first-touch

(1% slower than best in average)

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

First-touch

High imbalance

Moderate imbalance
Predict the NUMA policy

Moderate imbalance with first-touch
First-touch *roughly balances the load but locality is not perfect*

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

First-touch  
High imbalance  
Moderate imbalance
Predict the NUMA policy

Moderate imbalance with first-touch
First-touch roughly balances the load but locality is not perfect

⇒ use First-touch/Carrefour
(2% slower than best in average)

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

First-touch

High imbalance

First-touch/Carrefour
Predict the NUMA policy

High imbalance with first-touch
Interleaved balances the load and Carrefour improves locality

⇒ use Interleaved/Carrefour

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

First-touch  Interleaved/Carrefour  First-touch/Carrefour
Predict the NUMA policy

High imbalance with first-touch
Interleaved balances the load and Carrefour improves locality
⇒ use Interleaved/Carrefour
(2% slower than best in average)

<table>
<thead>
<tr>
<th>Imbalance</th>
<th>0%</th>
<th>40%</th>
<th>62%</th>
<th>83%</th>
<th>107%</th>
<th>138%</th>
<th>185%</th>
<th>283%</th>
</tr>
</thead>
<tbody>
<tr>
<td># of accessed nodes</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

First-touch          Interleaved/Carrefour
First-touch/Carrefour
Predict the NUMA policy

High imbalance with first-touch
Interleaved balances the load and Carrefour improves locality

Second conclusion

We can reasonably predict the best NUMA policy of an application
To take away

- NUMA placement has a huge impact on performance
- No miraculous NUMA policy for all the applications
- We can mitigate the NUMA effect
  - With first-touch, interleaved and Carrefour in many cases
  - With hand-tuned policies for specific applications (such as a GC)
To take away

- NUMA placement has a huge impact on performance
- No miraculous NUMA policy for all the applications
- We can mitigate the NUMA effect
  - With first-touch, interleaved and Carrefour in many cases
  - With hand-tuned policies for specific applications (such as a GC)

Thank You 😊