Application-Level Optimizations on NUMA Multicore Architectures: the Apache Case Study

Fabien Gaud\textsuperscript{1}, Renaud Lachaize\textsuperscript{2}, Baptiste Lepers\textsuperscript{3}, Gilles Muller\textsuperscript{1}, Vivien Quéma\textsuperscript{3}

\footnotesize{\textsuperscript{1}INRIA, \textsuperscript{2}Université de Grenoble, \textsuperscript{3}CNRS}

May 12, 2010
Objectives

- NUMA multicore architectures are becoming commonplace

- **Application domain:** Data servers, *a.k.a.* networked services

- **Goal:** Improve the performance of data servers on multicore architectures

- **Contribution:** Scaling the Apache Web server on NUMA multicore systems using application-level optimizations
The Apache web server do not scale on NUMA architectures
What can we do?

- Address scalability issues at the OS level
  - Corey (OSDI 08)
  - Barrelfish (SOSP 09)
  - PK (OSDI 10)
Apache on PK

The diagram shows the scalability of Apache on PK with respect to the number of dies and the number of clients per die. The line labeled "Ideal scalability" represents an ideal scenario, while the lines labeled "Apache on PK" and "Apache" show the actual performance, which is lower than the ideal case by 22%. This indicates that Apache on PK does not solve scalability issues.

Application-Level Optimizations on NUMA Multicore Architectures: the Apache Case Study
What do we propose?

- Addressing scalability issues at the OS level is not sufficient
  - Application-level issues
  - Some issues are difficult to handle (e.g. scheduling)

- **Approach:** address scalability issues at the application level
Methodology

- Consider both hardware and software bottlenecks

- Hardware bottlenecks:
  - Processor interconnect
  - Distant memory accesses

- Software bottlenecks:
  - Synchronization primitives
Hardware testbed

- 4 processors / 16 cores
Hardware testbed

- 4 processors / 16 cores
Hardware bottlenecks

- Memory efficiency (IPC)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 die</td>
<td>0.38</td>
</tr>
<tr>
<td>4 dies</td>
<td>0.30</td>
</tr>
</tbody>
</table>

21% IPC decrease
Hardware bottlenecks (2)

- IPC decrease:
  - Reduced cache efficiency

<table>
<thead>
<tr>
<th>Configuration</th>
<th>L3 cache miss ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 die</td>
<td>14</td>
</tr>
<tr>
<td>4 dies</td>
<td>14</td>
</tr>
</tbody>
</table>
Hardware bottlenecks (2)

- IPC decrease:
  - Reduced cache efficiency
  - HyperTransport link saturation

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Max HT usage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 die</td>
<td>25</td>
</tr>
<tr>
<td>4 dies</td>
<td>75</td>
</tr>
</tbody>
</table>
Hardware bottlenecks (2)

- IPC decrease:
  - Reduced cache efficiency
  - HyperTransport link saturation
  - Increased number of distant memory accesses

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Distant accesses/kB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 die</td>
<td>4</td>
</tr>
<tr>
<td>4 dies</td>
<td>14</td>
</tr>
</tbody>
</table>
Request processing

Receiving a TCP request
HTTP request processing
Request processing

PHP processing
Request processing

Sending the response (1)
Request processing

Sending the response (2)
Proposal #1

- **Solution:** co-localizing TCP, Apache and PHP processing

- **Implementation:** use one instance of the Apache/PHP stack per die (*N-Copy*)
  - One node manages 5 network interfaces
N-Copy: request processing

Receiving a TCP request
N-Copy: request processing

HTTP request processing
N-Copy: request processing

PHP processing
N-Copy: request processing

Sending the response (1)
N-Copy: request processing

Sending the response (2)
N-Copy: performance

![Graph showing performance improvement of N-Copy compared to Apache.]

**9.1% performance improvement compared to stock Apache**
N-Copy: performance (2)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average IPC</th>
<th>Distant accesses/kB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 die</td>
<td>0.38</td>
<td>4</td>
</tr>
<tr>
<td>4 dies (Stock Apache)</td>
<td>0.30</td>
<td>14</td>
</tr>
<tr>
<td>4 dies (N-Copy)</td>
<td>0.36</td>
<td>5</td>
</tr>
</tbody>
</table>

Memory efficiency improved by 20%
N-Copy: can we do better?

<table>
<thead>
<tr>
<th>Die</th>
<th>Average CPU usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die 0</td>
<td>100</td>
</tr>
<tr>
<td>Die 1</td>
<td>85</td>
</tr>
<tr>
<td>Die 2</td>
<td>85</td>
</tr>
<tr>
<td>Die 3</td>
<td>100</td>
</tr>
</tbody>
</table>

Problem:
- Dies are not equally efficient
- Load is not *properly* balanced on dies
N-Copy: load balancing

- **Solution:** balance load on dies proportionally to their efficiency

- **Implementation:** use an external load balancing mechanism
  - Currently implemented at client-side
  - Could be integrated in a more global solution
N-Copy: final performance

21.2% performance improvement compared to stock Apache
Software bottlenecks

- **Goal:** find functions that
  - Do not scale
  - Represent a significant execution time

- **Example:**
  - Function $f$ accounts for
    - 1 cycle/byte at 1 die
    - 10 cycles/byte at 4 dies
    - 20% of the total execution time
  - 18% *potential performance gain*
- **Problem:** the VFS layer does not scale
  - Aggregated potential performance gain: 6 %
  - Most of the calls are issued by the `stat` function
Proposal #2

- **Solution:** use an application-level cache to reduce the number of calls to `stat`.

- **Implementation:**
  - Modified the Apache `ap_directory_walk` function
  - Using `inotify` for file updates
Stat cache: performance

33% performance improvement compared to stock Apache
Summary

- **Problem:** Apache does not scale on NUMA architectures

- **Contribution:** application-level optimizations considering NUMA aspects and Linux scalability issues

- **Results:** +33% performance improvement
Conclusion
Conclusion

- **Goal:** Improve the performance of data servers on multicore architectures

- **Contributions:**
  - Detailed analysis of the Apache bottlenecks
  - Application-level optimisations that improve Apache scalability

- **Future work:**
  - Scheduling applications on NUMA multicore platforms with an asymmetric interconnect topology
  - Profiling tools for multicore architectures
  - Descriptive language that allows manipulating both flows and components
Questions?