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

Fabien Gaud[1], Renaud Lachaize[2], Baptiste Lepers[3], Gilles Muller[1], Vivien Quéma[3]

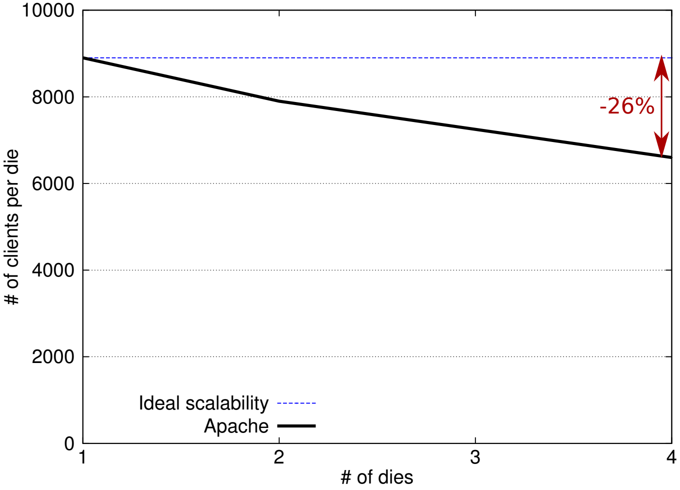
[1]INRIA, [2]Université de Grenoble, [3]CNRS

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

Problem

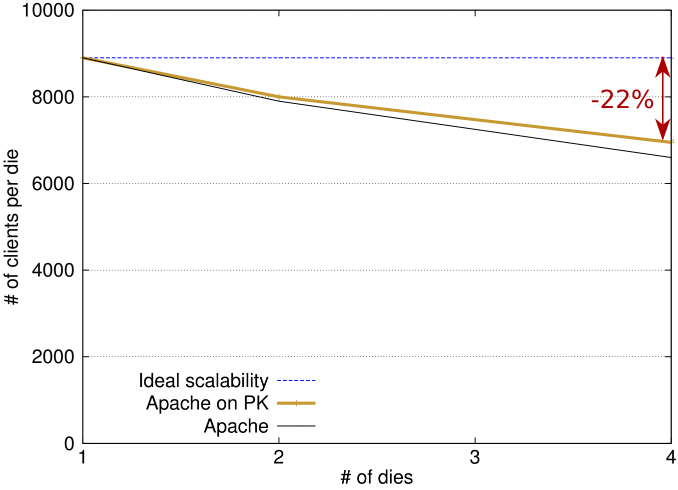


The Apache web server do not scale on NUMA architectures

What can we do?

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

Apache on PK



Does not solve scalability issues

What do we propose?

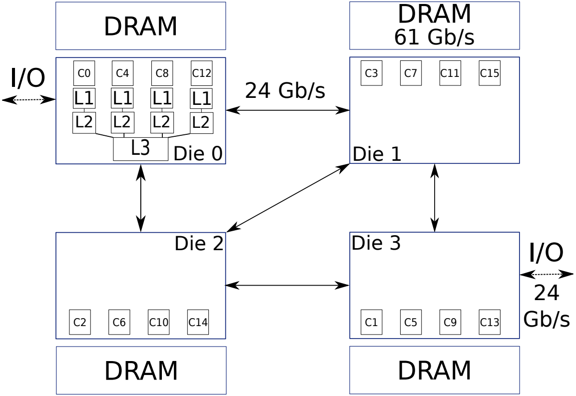
- Addressing scalability issues at the OS level is not sufficient
  - Application-level issues
  - Some issues are difficult to handle (e.g. scheduling)

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

Methodology

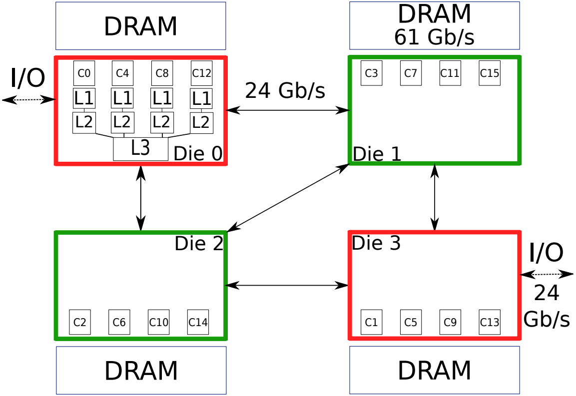
- Consider both hardware and software bottlenecks
- Hardware bottlenecks:
  - Processor interconnect
  - Distant memory accesses
- Software bottlenecks:
  - Synchronization primitives

Hardware testbed



- 4 processors / 16 cores

Hardware testbed



- 4 processors / 16 cores

Hardware bottlenecks

- Memory efficiency (IPC)

Configuration	Average IPC
1 die	0.38
4 dies	0.30

21% IPC decrease

Hardware bottlenecks (2)

- IPC decrease:
  - Reduced cache efficiency

Configuration	L3 cache miss ratio (%)
1 die	14
4 dies	14

Hardware bottlenecks (2)

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

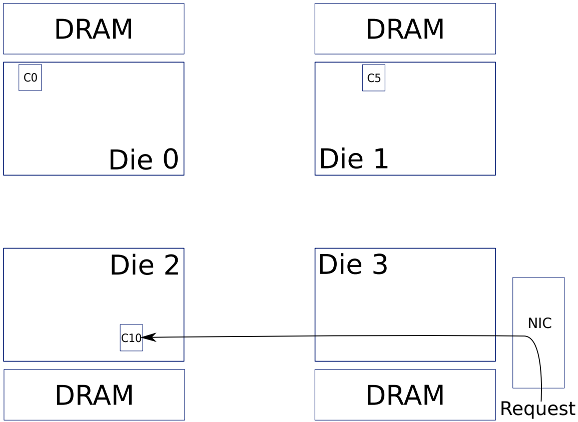
Configuration	Max HT usage (%)
1 die	25
4 dies	75

Hardware bottlenecks (2)

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

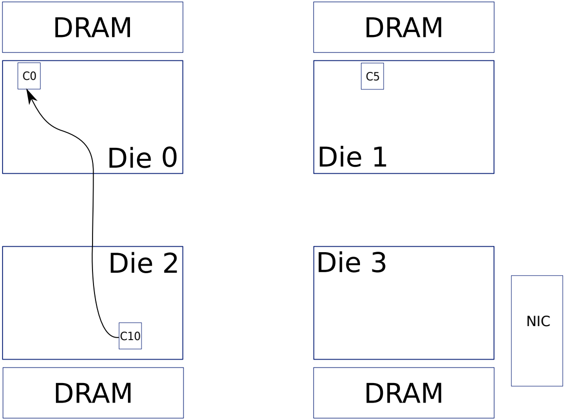
Configuration	Distant accesses/kB
1 die	4
4 dies	14

Request processing



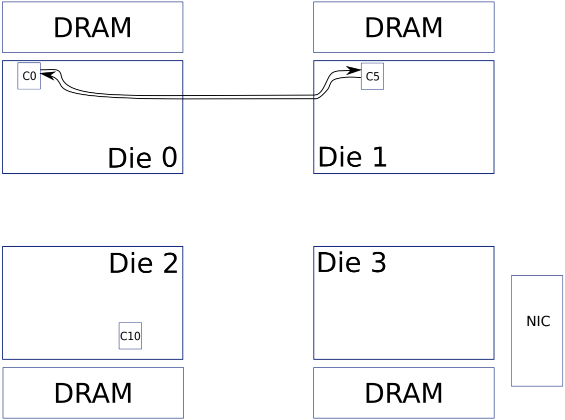
Receiving a TCP request

Request processing



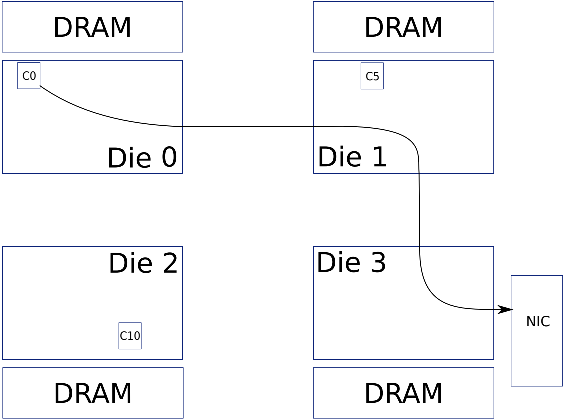
HTTP request processing

Request processing



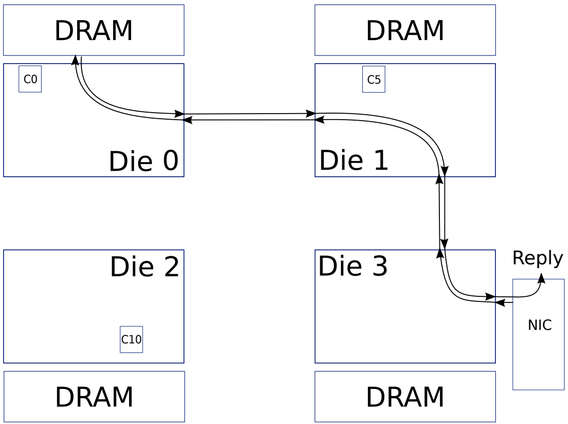
PHP processing

Request processing



Sending the response (1)

Request processing

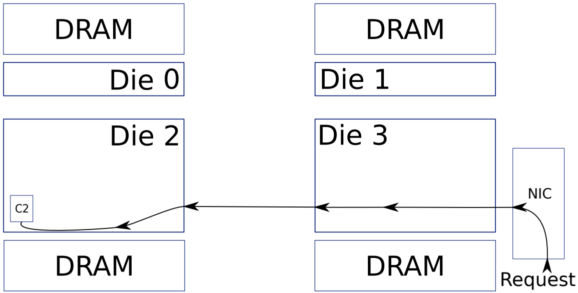


Sending the response (2)

Proposal #1

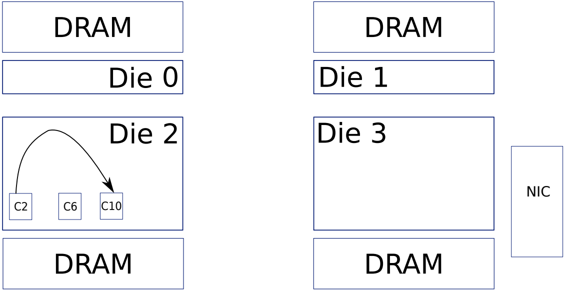
- **Solution:** co-localizing TCP, Apache and PHP processing
- **Implementation:** use one instance of the Apache/PHP stack per die *(N-Copy)*
  - One node manages 5 network interfaces

N-Copy: request processing



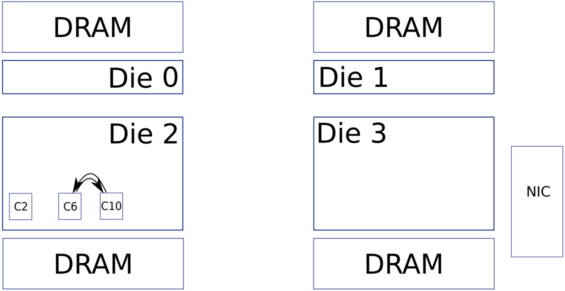
Receiving a TCP request

N-Copy: request processing



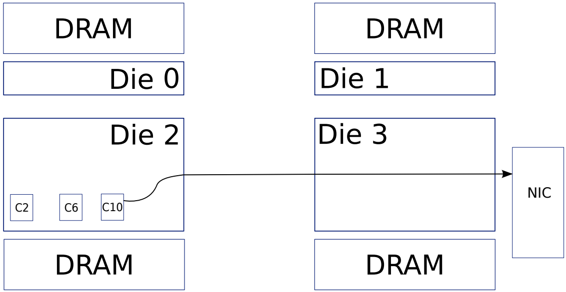
HTTP request processing

N-Copy: request processing



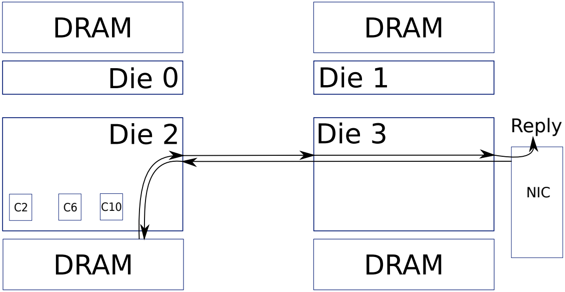
PHP processing

N-Copy: request processing



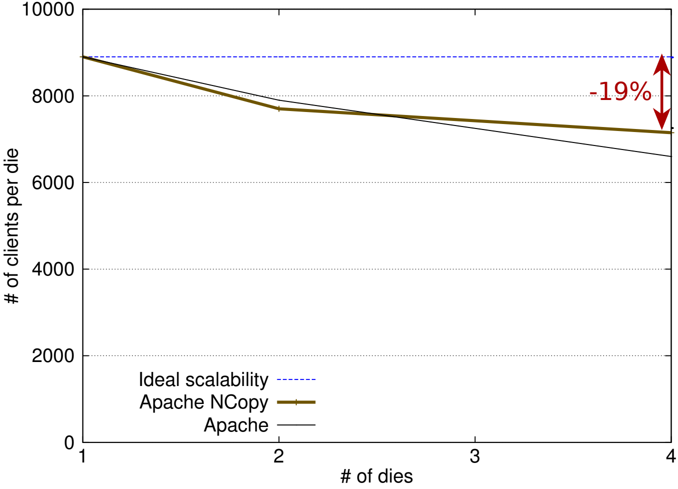
Sending the response (1)

N-Copy: request processing



Sending the response (2)

N-Copy: performance



9.1% performance improvement compared to stock Apache

N-Copy: performance (2)

Configuration	Average IPC	Distant accesses/kB
1 die	0.38	4
4 dies (Stock Apache)	0.30	14
4 dies (N-Copy)	0.36	5

Memory efficiency improved by 20%

N-Copy: can we do better?

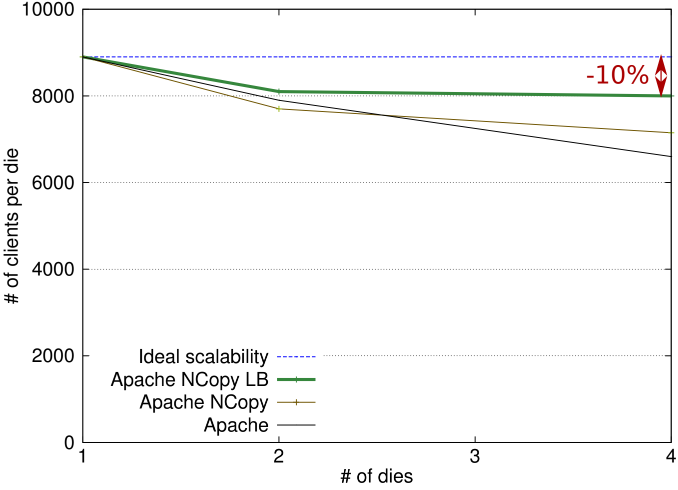
Die	Average CPU usage
Die 0	100
Die 1	85
Die 2	85
Die 3	100

- **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*

Software bottlenecks (2)

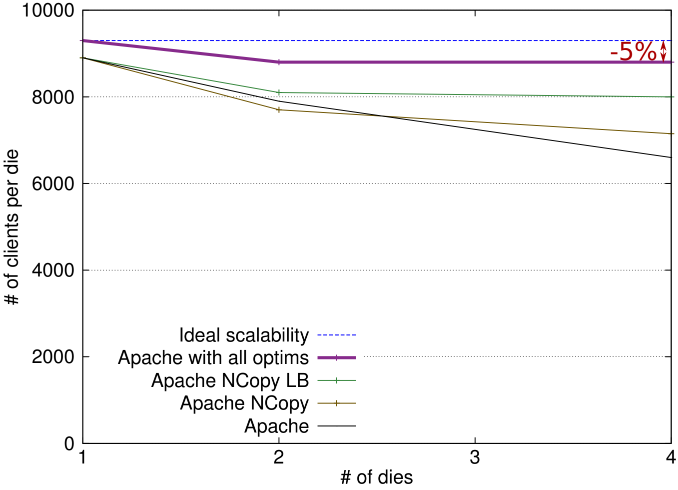
Function	Potential performance gain (%)
__d_lookup	2.49%
_atomic_dec_and_lock	2.32%
lookup_mnt	1.41%
copy_user_generic_string	0.83%
memcpy	0.76%

- **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?