Performance of Multi-Process and Multi-Thread Processing on Multi-core SMT Processors

Hiroshi Inoue and Toshio Nakatani
IBM Research – Tokyo

An Old Question on New Platforms

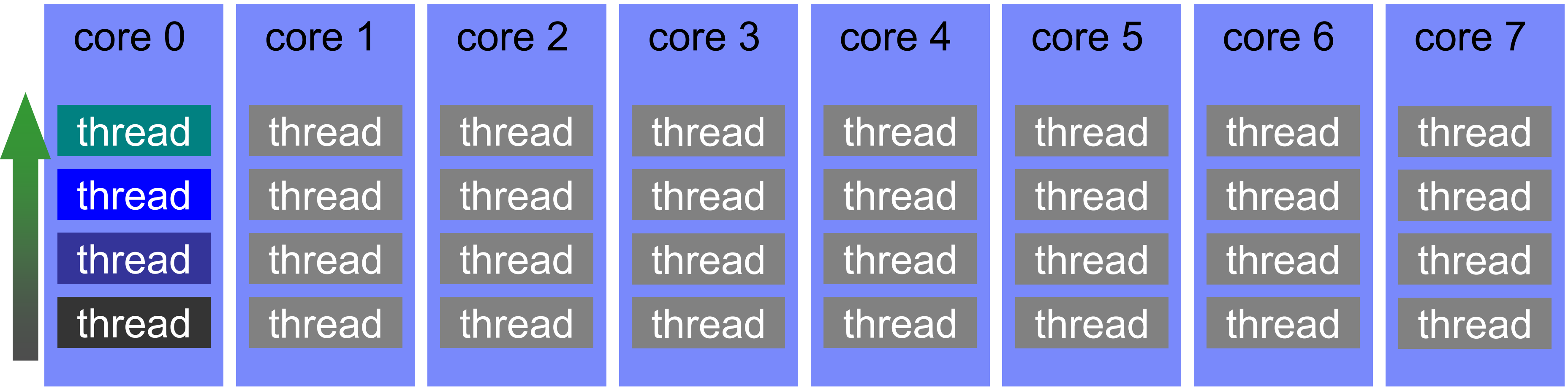
- **Threads** vs. **Processes**: Which is better to achieve higher performance?
  - Each process has own virtual memory space
  - → Using processes provides better inter-process isolation
  - Threads in one process shares a virtual memory space
  - → *Multi-thread processing is better for performance due to its memory efficiency (smaller footprint)*

- Is this answer still valid on today's processors with multiple cores and multiple SMT threads in a core?

Approach

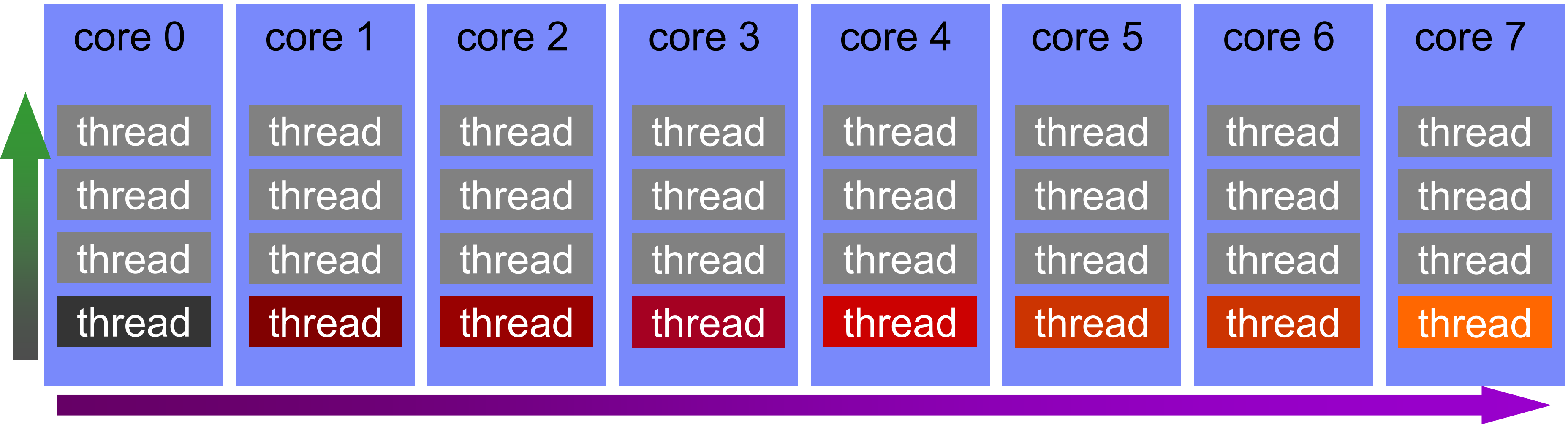
- Comparing multi-thread model and multi-process model on two types of hardware parallelism
  - SMT scalability
  - Core scalability

SMT Scalability and Core Scalability



SMT scalability: performance improvement
using increasing number of SMT threads in one core

SMT Scalability and Core Scalability



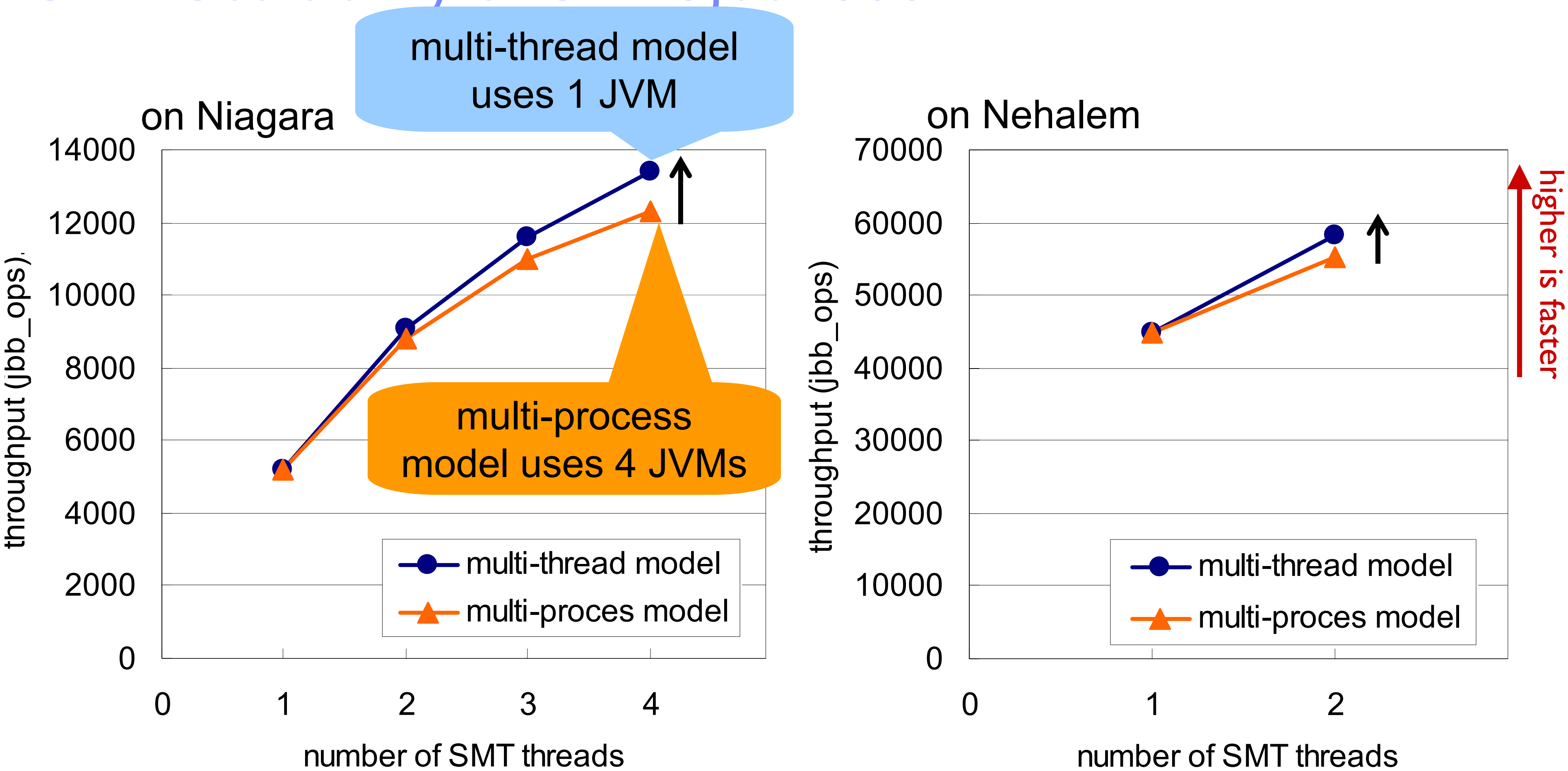
SMT scalability: performance improvement using increasing number of SMT threads in one core

Core scalability: performance improvement using increasing number of cores with one thread in each core

Experimental Setup

- Systems
  - Niagara system
    - UltraSPARC T1 (Niagara 1) 1.2 GHz
    - 8 cores with 4 SMT threads in each core
    - Solaris 10
  - Nehalem system
    - Xeon X5570 (Nehalem) 2.93 GHz
    - 4 cores with 2 SMT threads in each core
    - Red Hat Enterprise Linux 5.4
- Software
  - Benchmarks: SPECjbb2005, SPECjvm2008
  - 32-bit HotSpot Server VM for Java 6 Update 17
  - Java heap size: 256 MB per thread using large page

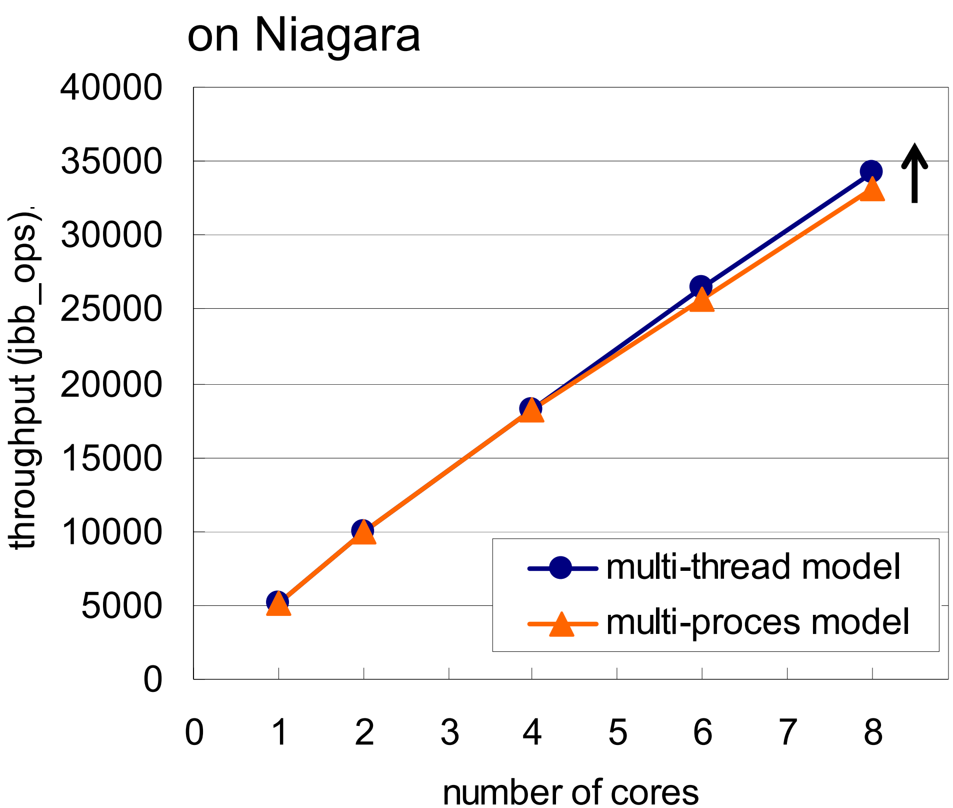
SMT Scalability of SPECjbb2005

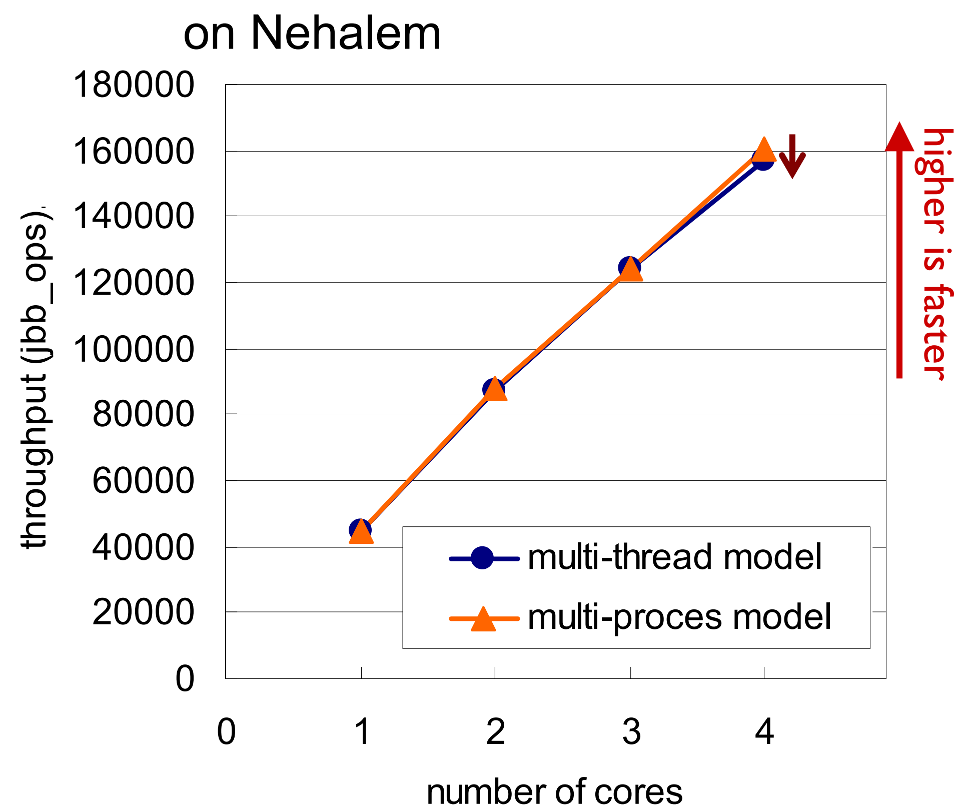


multi-thread model was 9.2% faster

multi-thread model was 5.5% faster

Core Scalability of SPECjbb2005



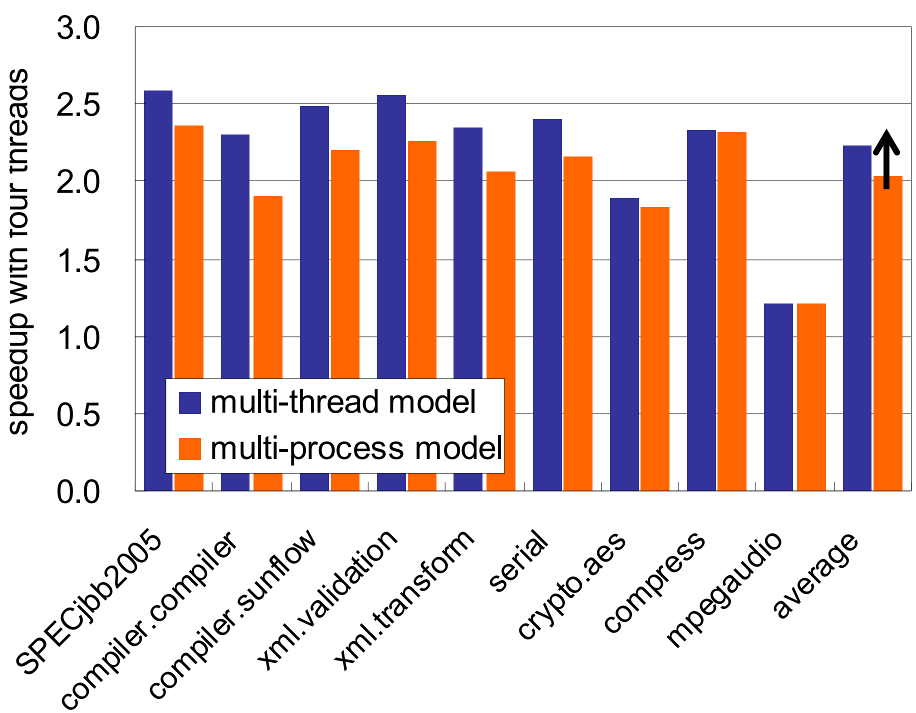


multi-thread model was 3.4% faster

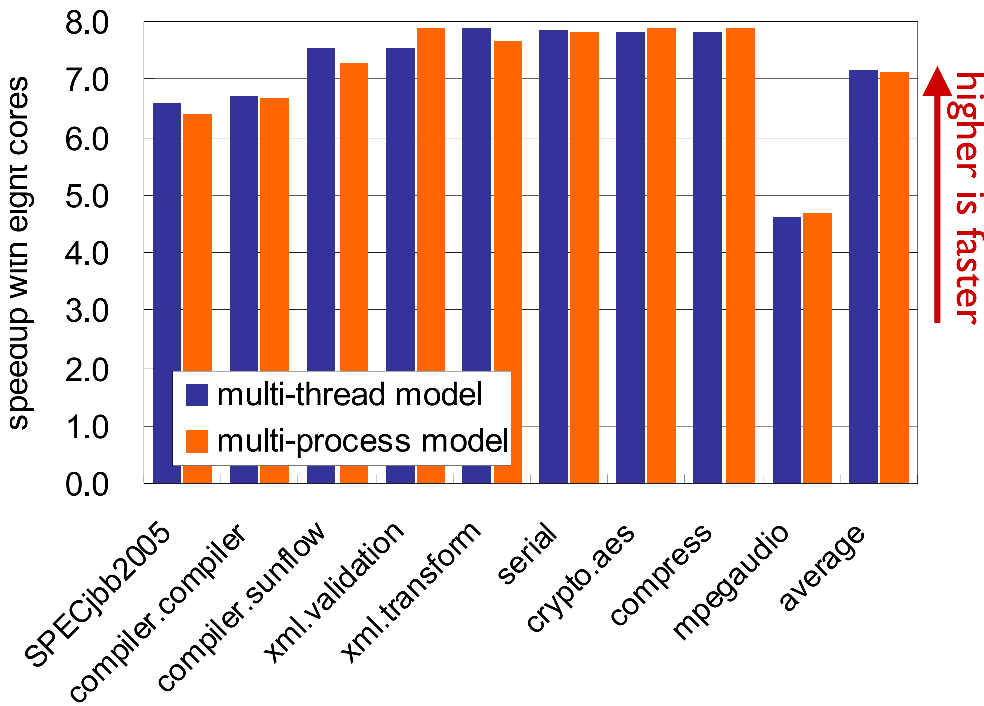
multi-thread model was 2.1% **slower**

Core Scalability and SMT Scalability on Niagara

SMT scalability



Core scalability

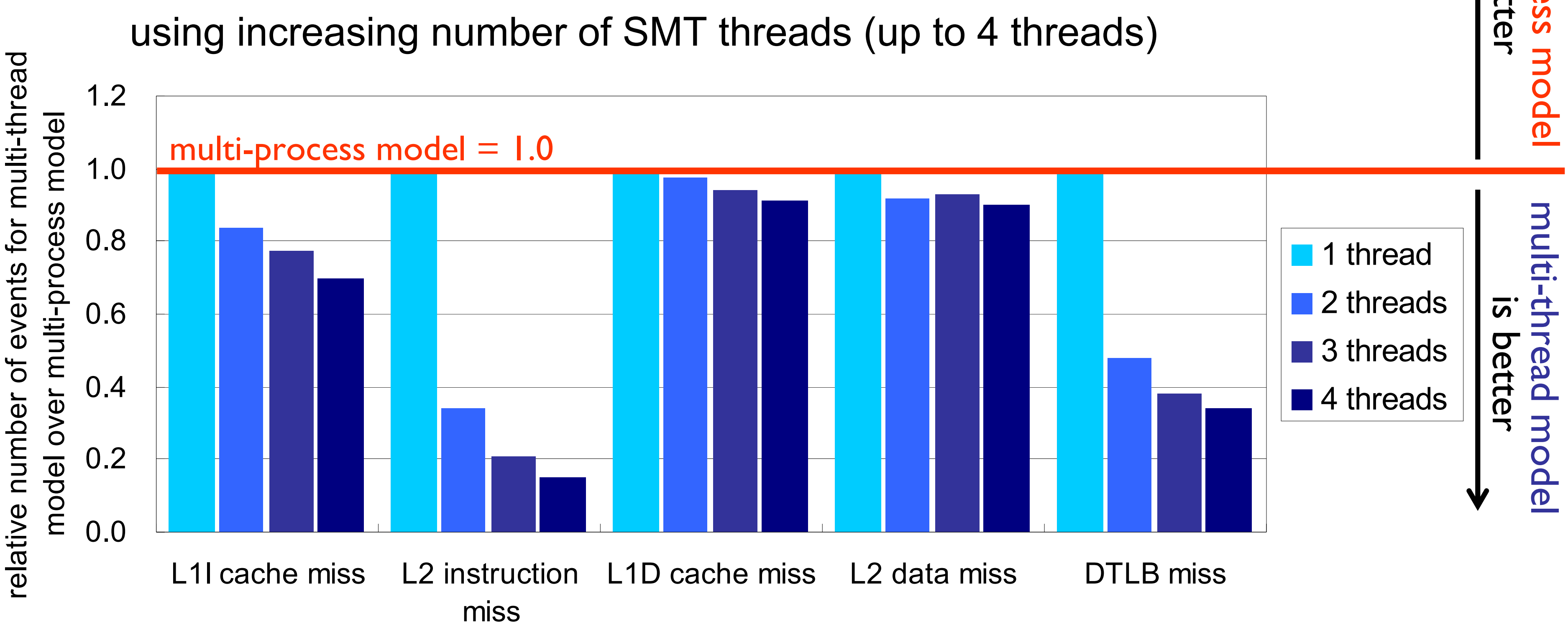


multi-thread model was 9.6% faster on average

No performance advantage for multi-thread model

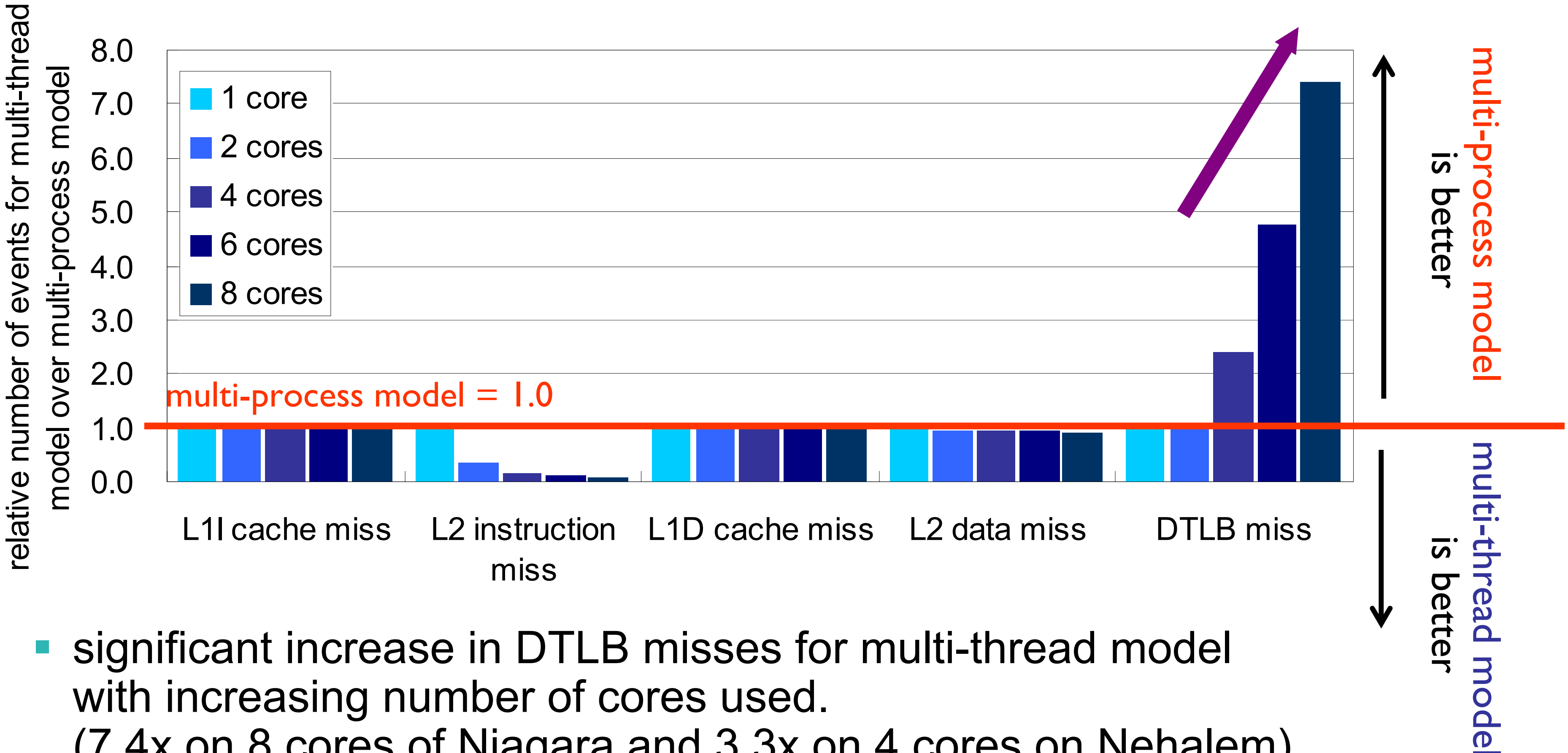
(please refer to the paper on results for Nehalem)

Micro Architectural Statistics for SPECjbb2005



Micro Architectural Statistics for SPECjbb2005

using increasing number of cores (up to 8 cores)

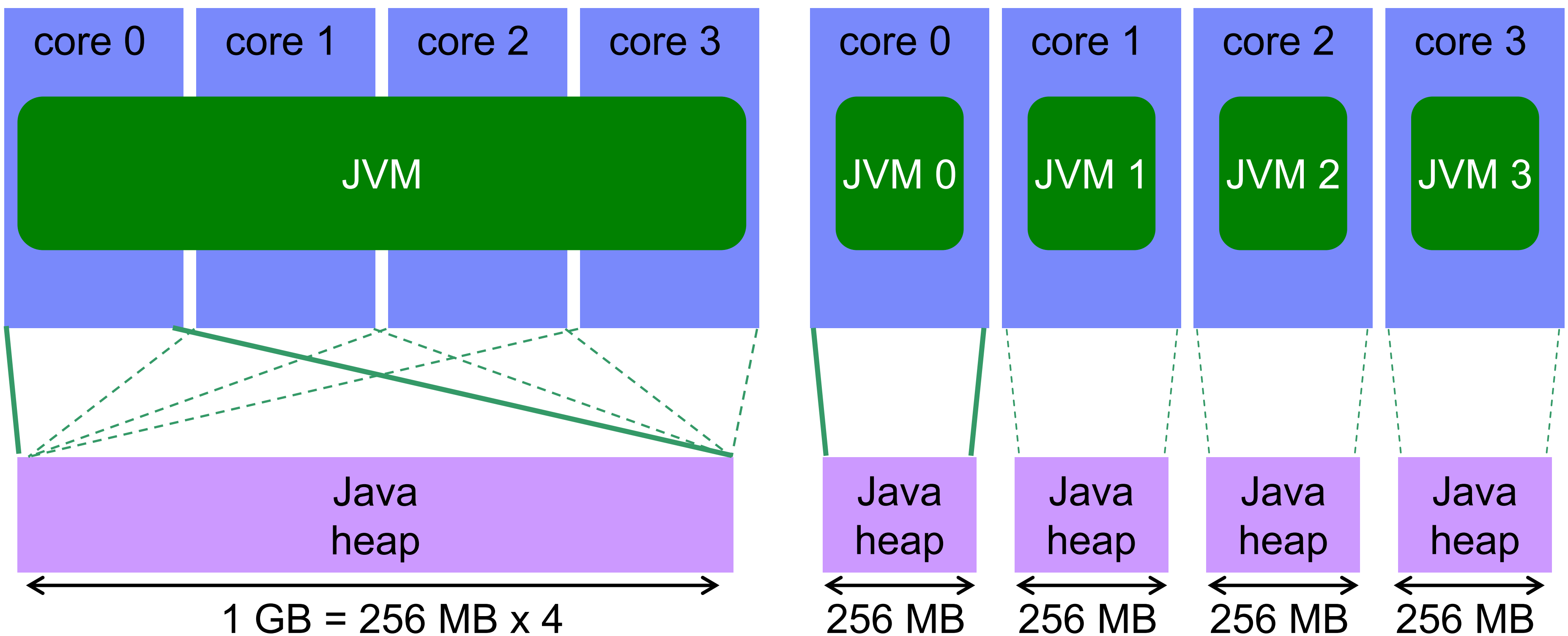


- significant increase in DTLB misses for multi-thread model with increasing number of cores used. (7.4x on 8 cores of Niagara and 3.3x on 4 cores on Nehalem)

Difference in Memory Access Patterns

multi-thread (one-JVM) model

multi-process (multi-JVM) model

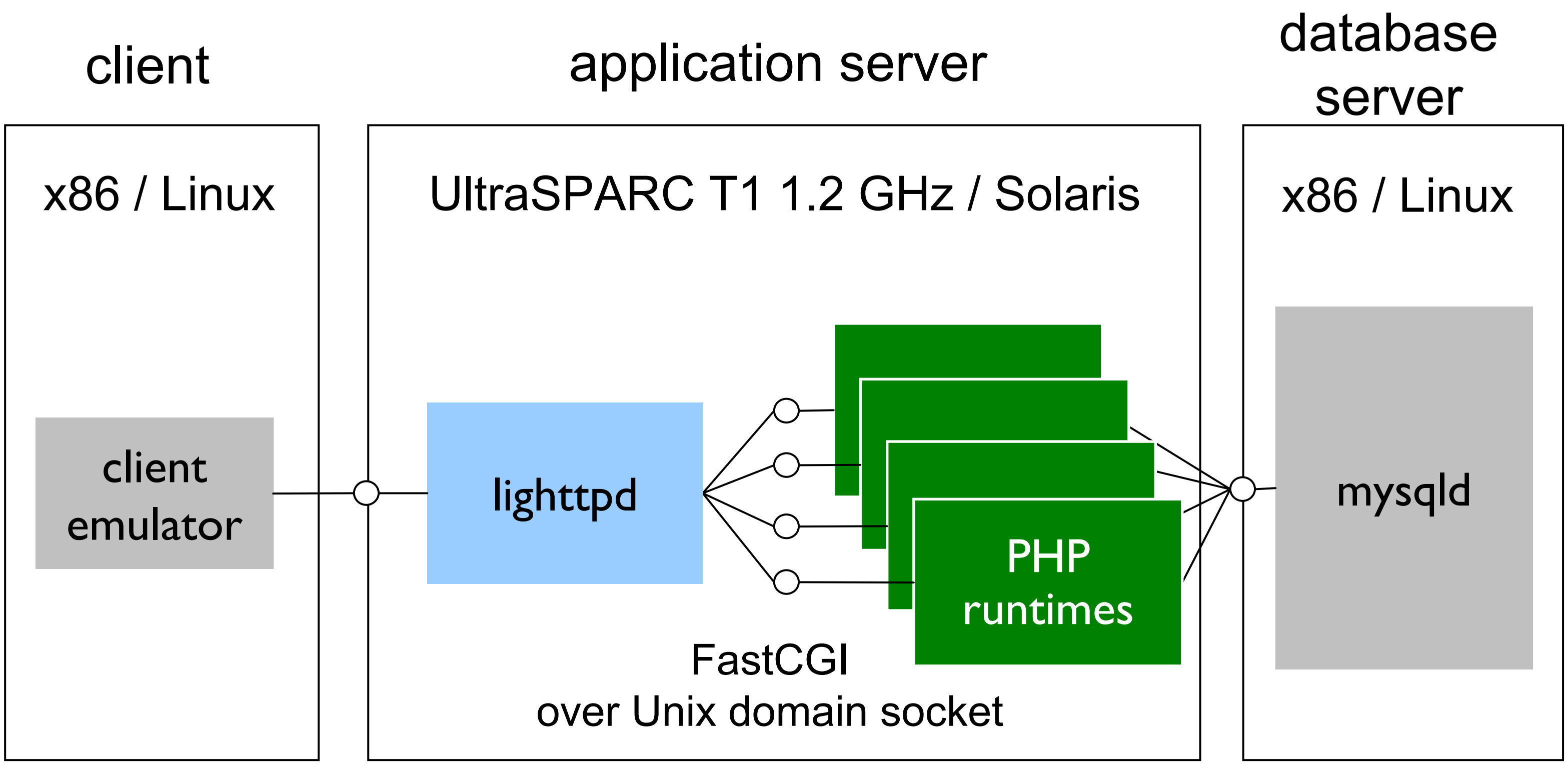


☹ each core accesses 1-GB memory space
☹ each memory page is accessed from 4 cores

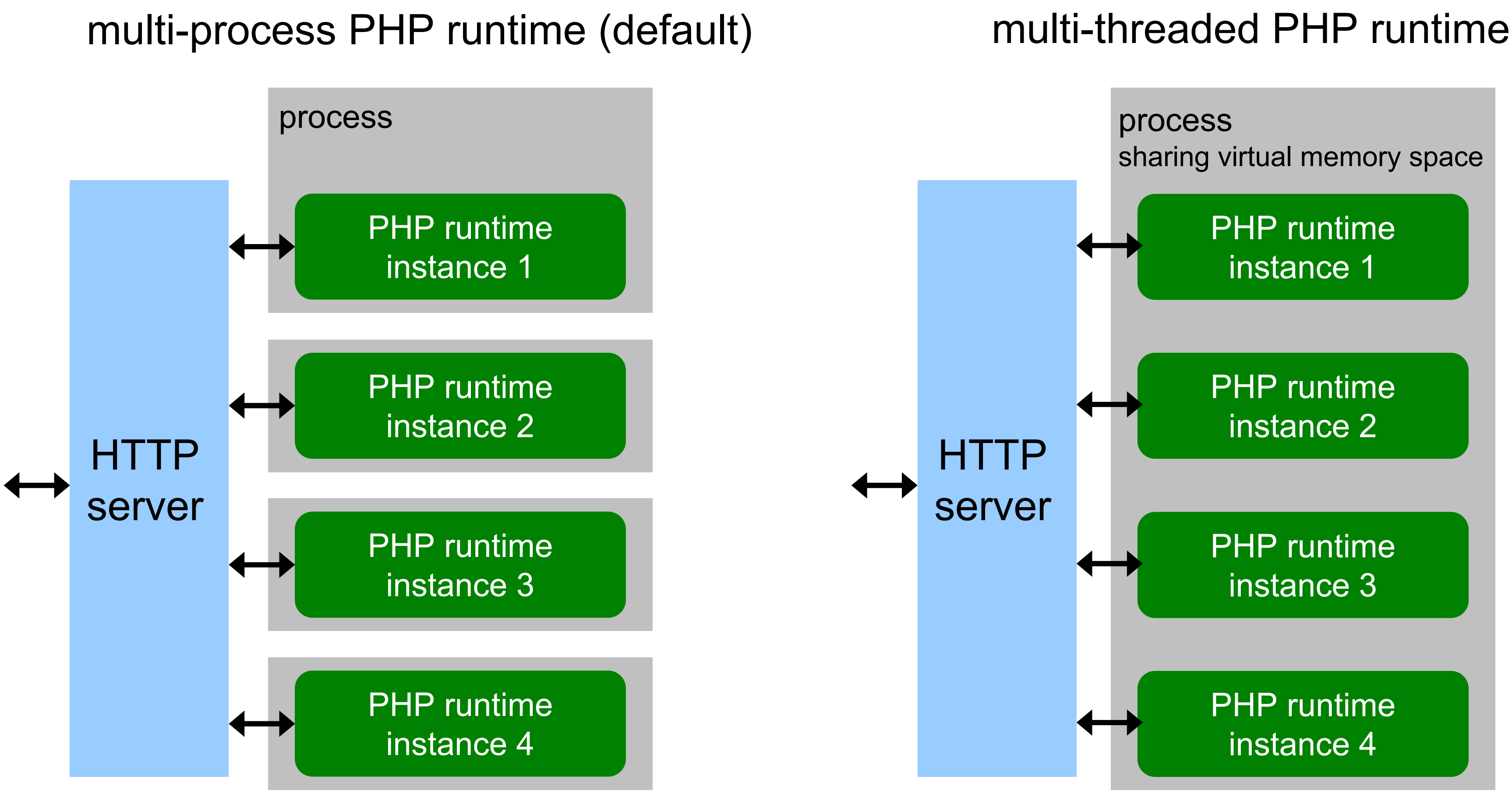
☺ each core accesses only 256-MB heap
☺ each memory page is accessed from only 1 core

Experimental Setup for a larger PHP workload

- Benchmark
  – MediaWiki (wiki server used in Wikipedia)

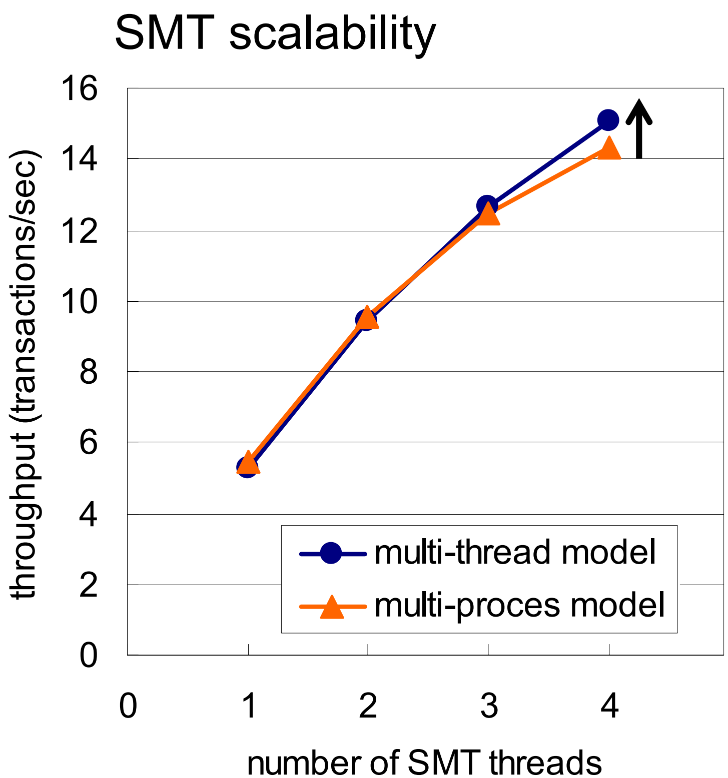


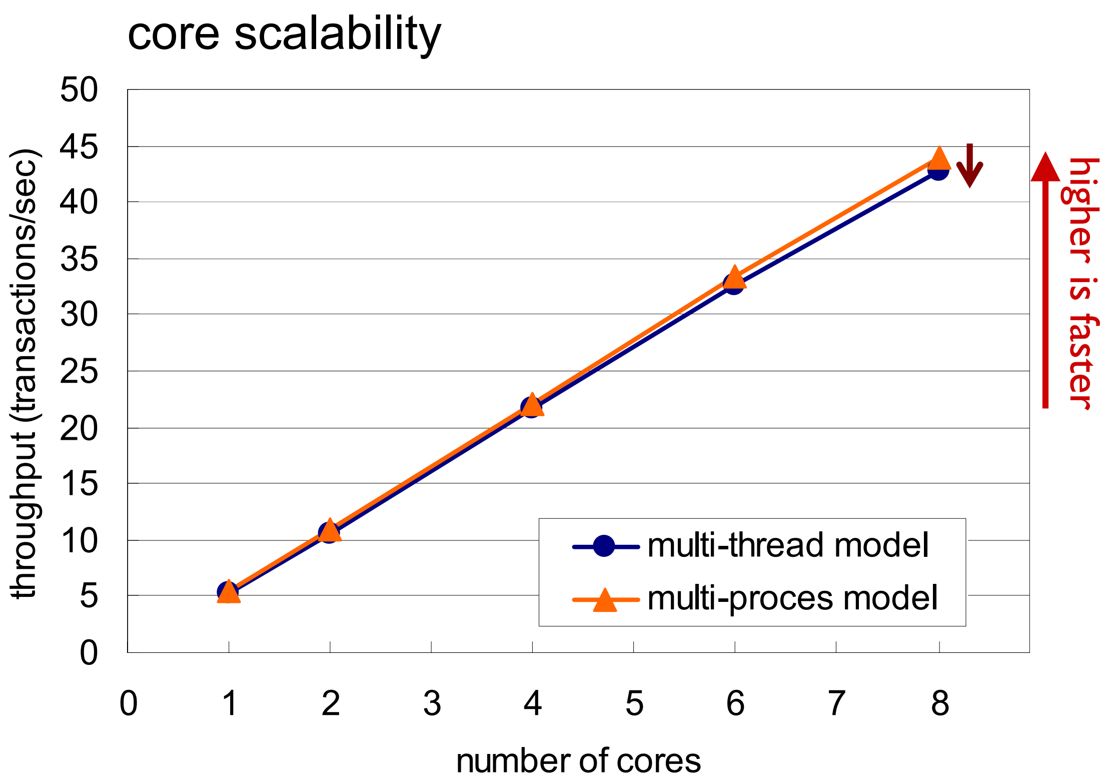
PHP runtime configuration



- each runtime instance handles independent requests
- no communication among PHP runtime instances

Core Scalability and SMT Scalability of MediaWiki



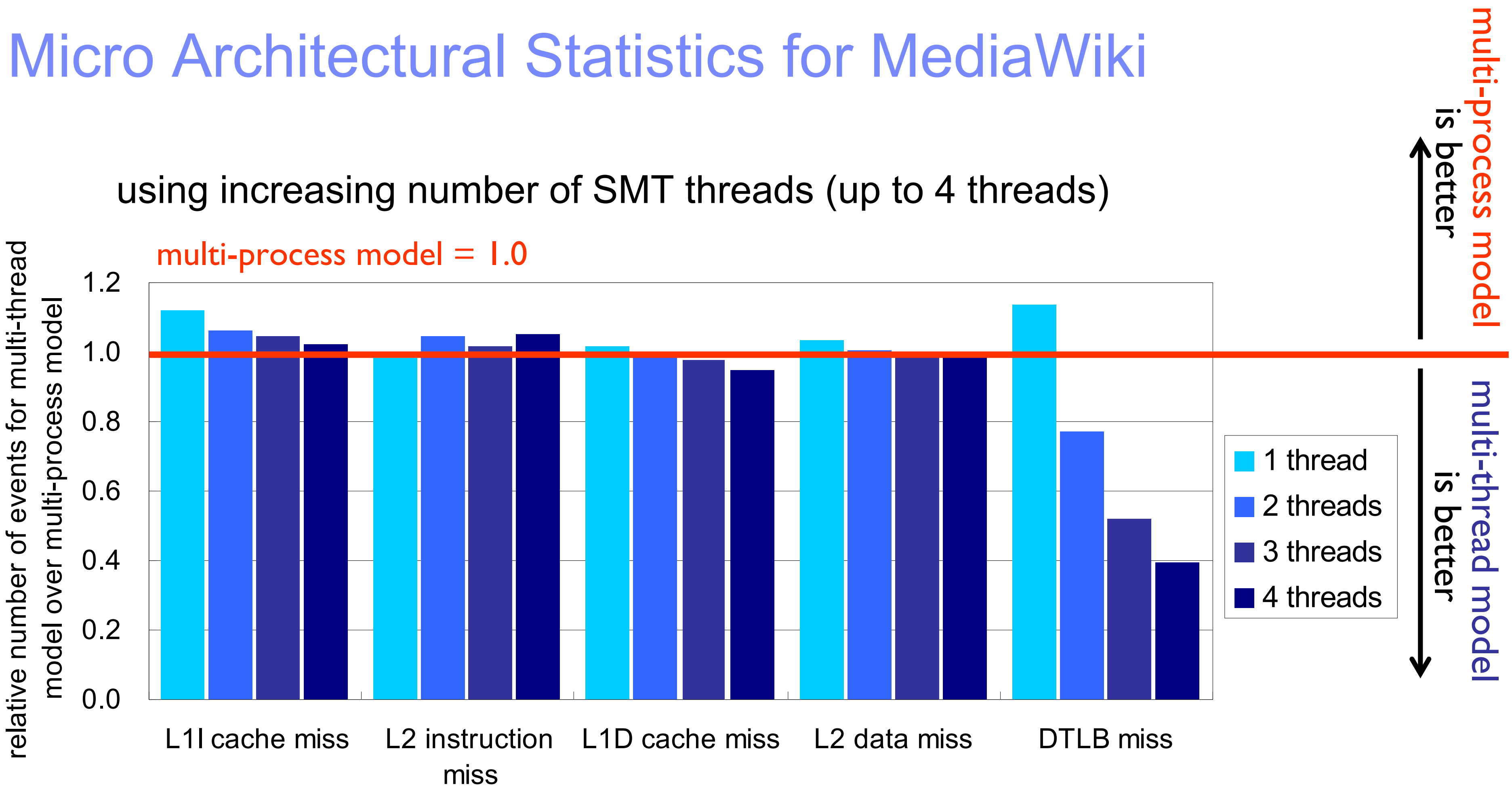


multi-thread model was 5.5% faster

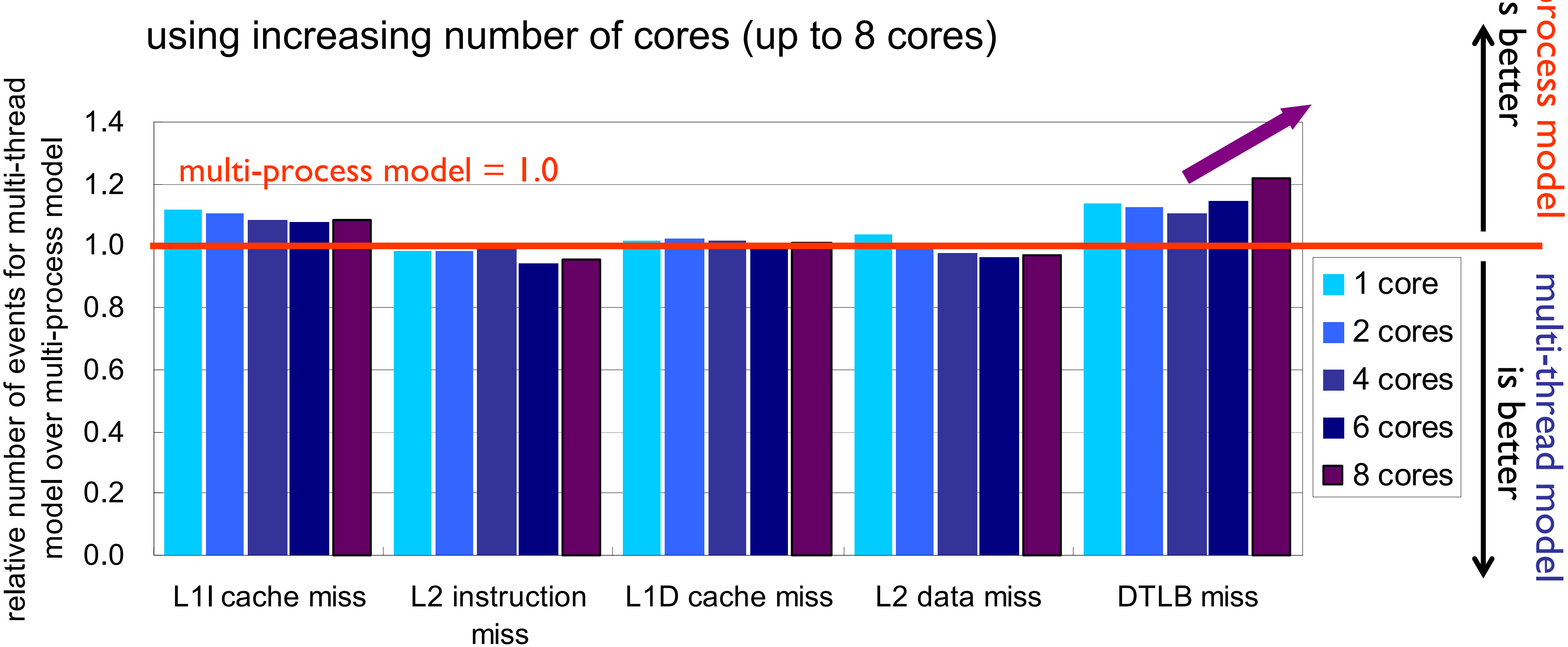
multi-thread model was 2.5% **slower**

- consistent with results for Java benchmarks

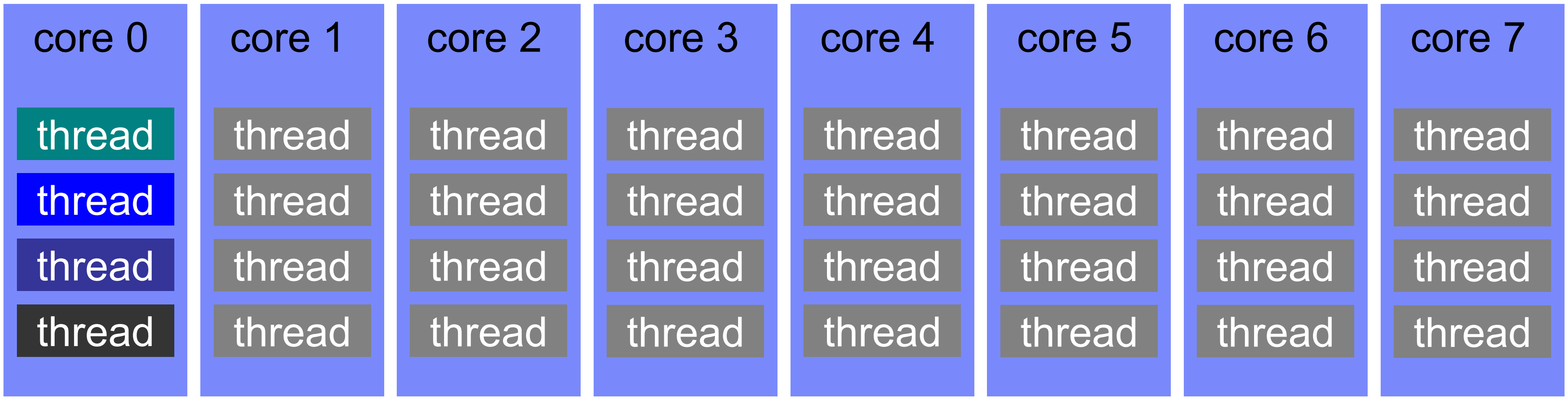
Micro Architectural Statistics for MediaWiki



Micro Architectural Statistics for MediaWiki



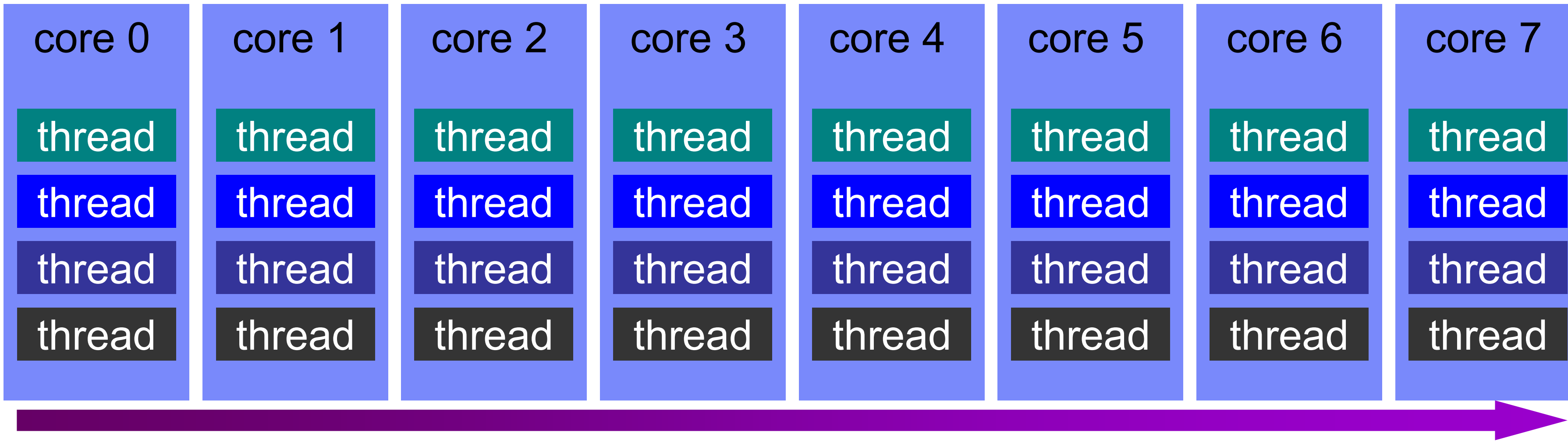
Performance of MediaWiki using All SMT Threads



☺ multi-thread model was 5.5% faster

☺ TLB misses were reduced by 60%

Performance of MediaWiki using All SMT Threads



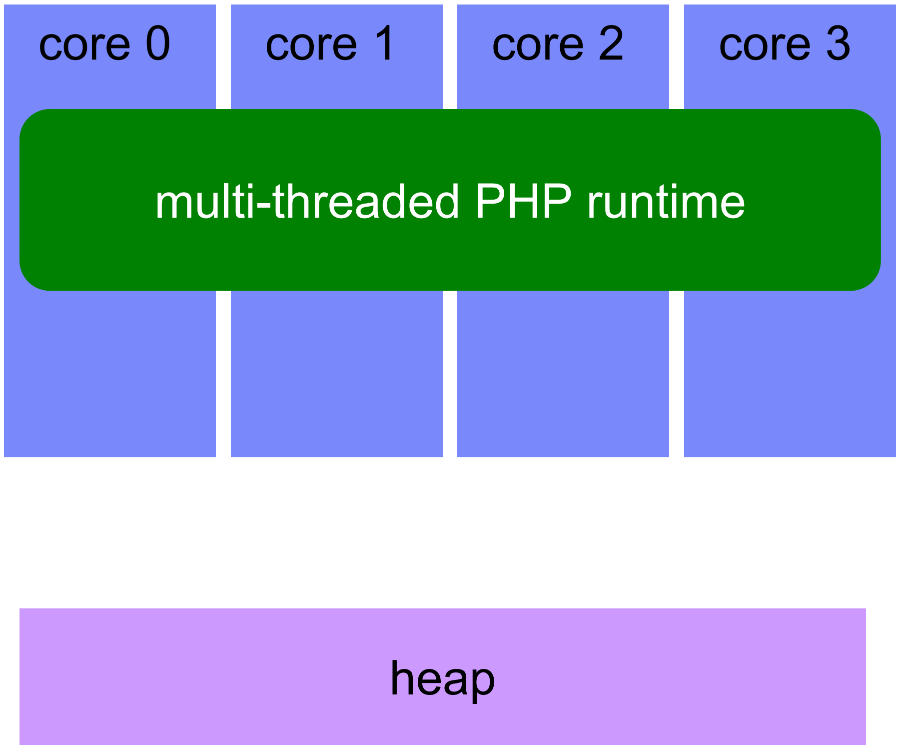
☺ multi-thread model was 5.5% faster

☺ TLB misses were reduced by 60%

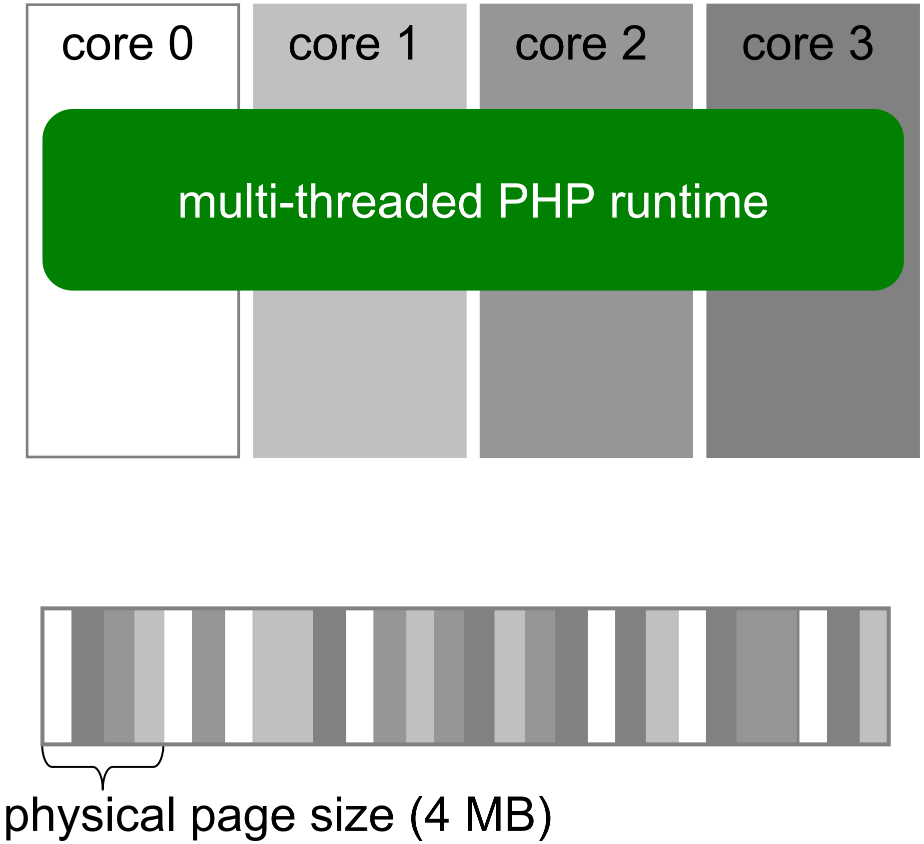
☹ multi-thread model was only 1.7% faster

☹ TLB misses were reduced by only 19%

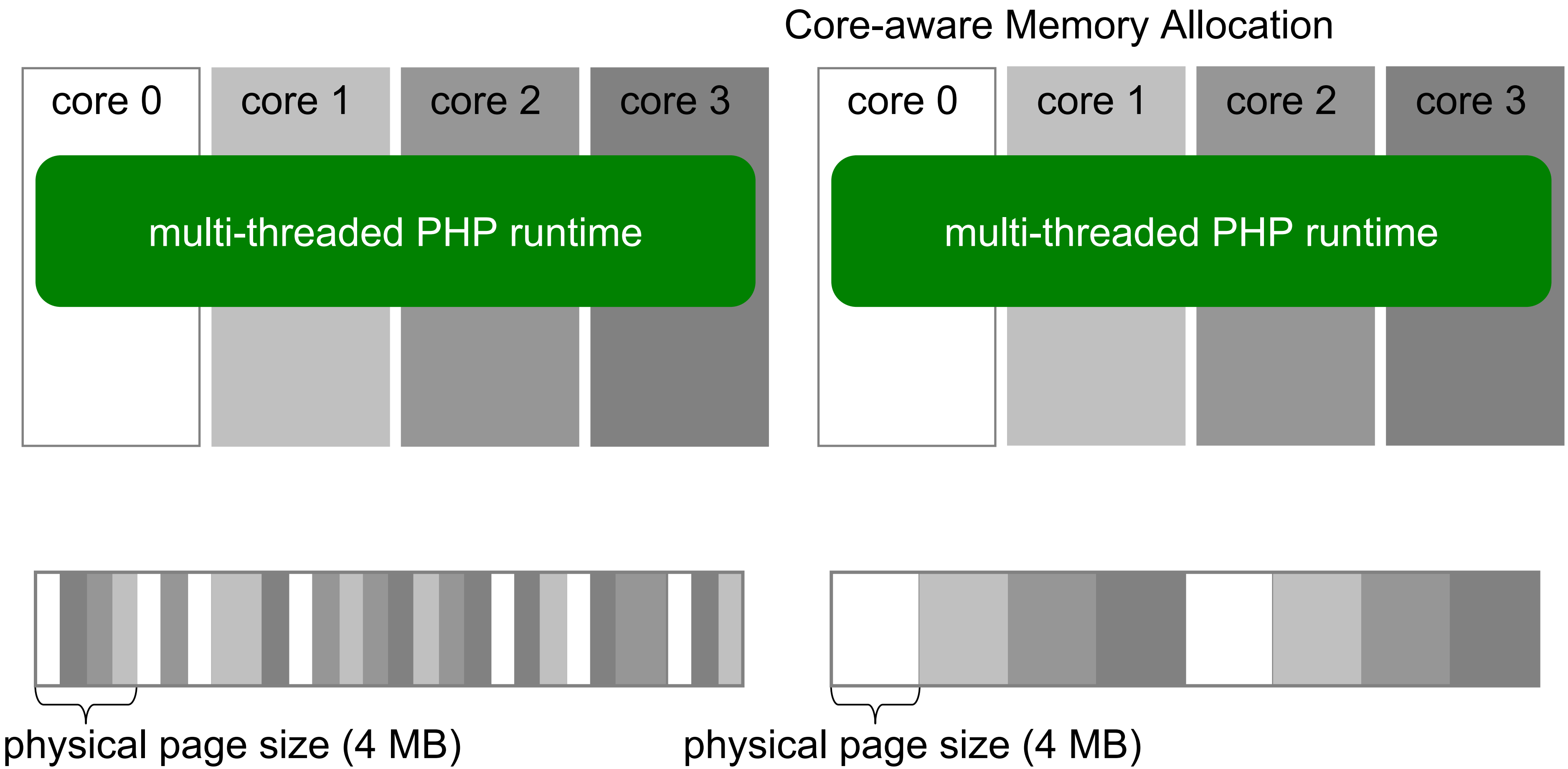
Our Technique: Core-aware Memory Allocation



Our Technique: Core-aware Memory Allocation

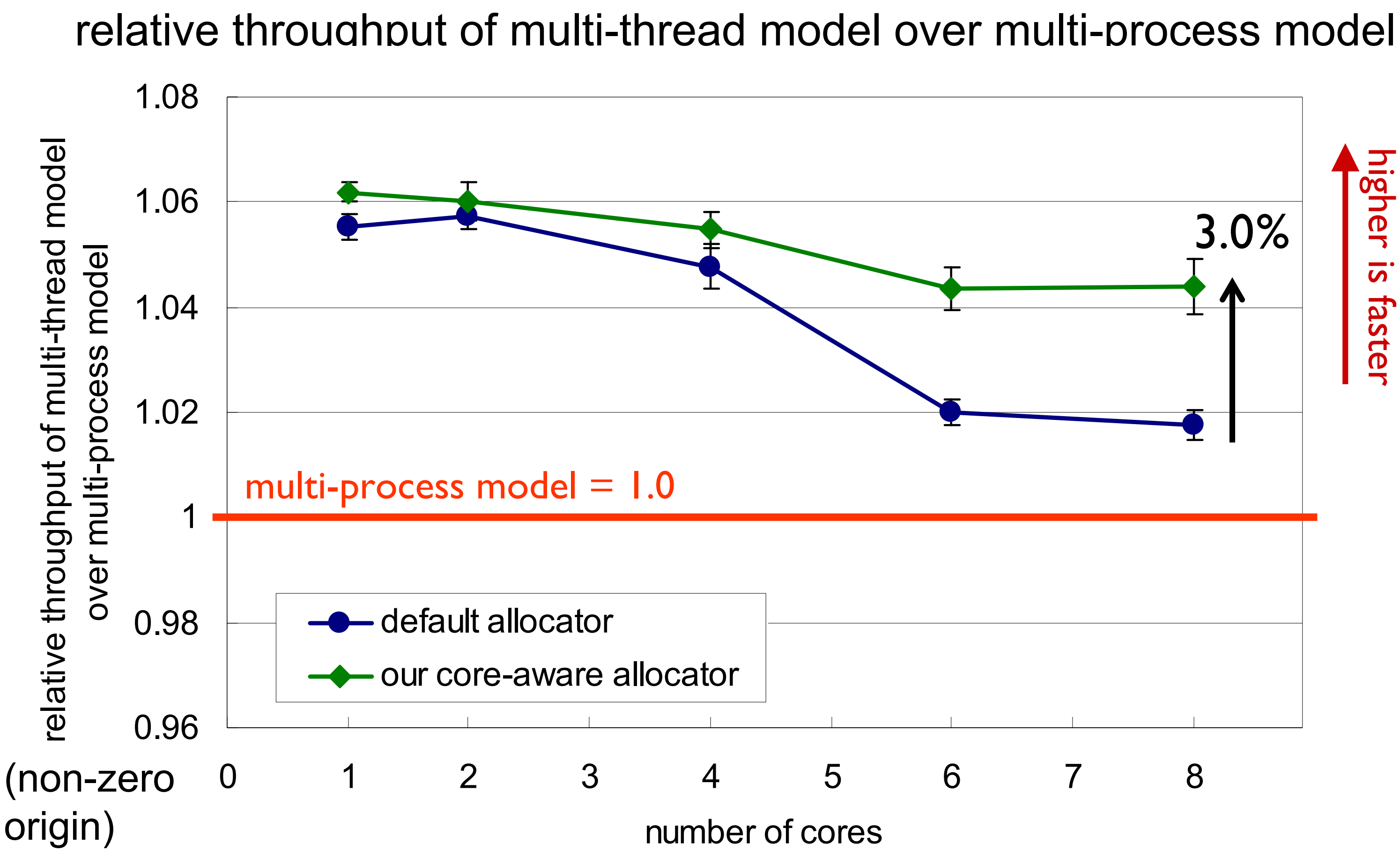


Our Technique: Core-aware Memory Allocation



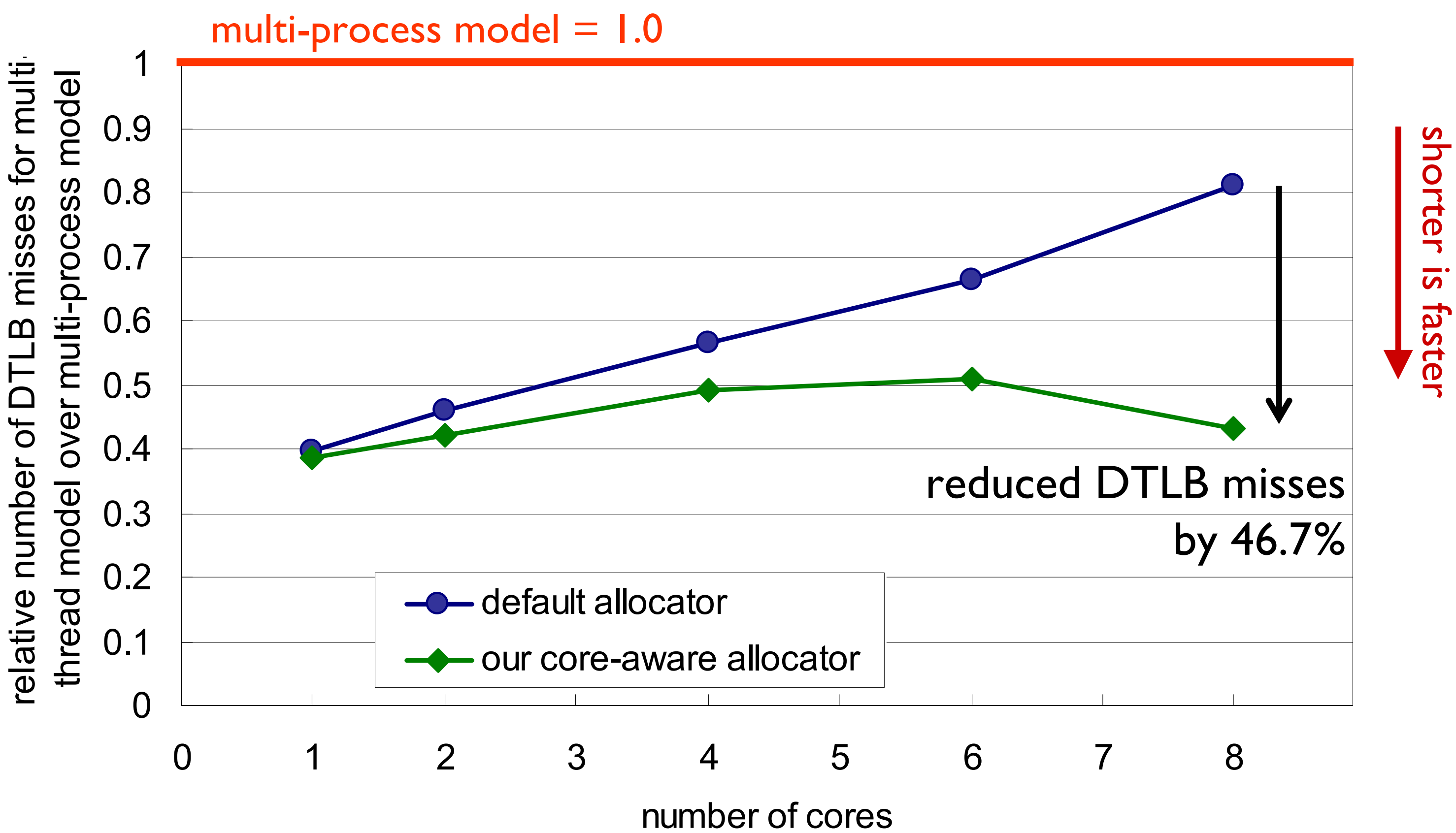
- avoid sharing the memory space among cores within a physical page

Performance of MediaWiki with Our Core-aware Malloc



- Our core-aware allocator improved the performance of multi-thread model by 3.0% over the default allocator in libc

DTLB misses with Our Core-aware Malloc



- Our core-aware allocator reduced the DTLB misses for the multi-thread model by 46.7%

Summary

- The multi-thread model tends to generate fewer cache misses but more DTLB misses on multi-core processors
- The increase in DTLB misses becomes more significant with increasing number of cores
- Core-aware memory allocation can maximize the benefit of multi-thread processing by reducing DTLB misses

Our Answer to the Question

- **Threads** vs. **Processes**: Which is better to achieve higher performance?

→ Multi-thread model has advantage over multi-process model, but memory allocator need to be enhanced