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 its own virtual memory space
  - Using processes provides better inter-process isolation
  - Threads in one process share 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

On Niagara:
- Multi-thread model: Uses 1 JVM
- Multi-process model: Uses 4 JVMs
- Multi-thread model was 9.2% faster

On Nehalem:
- Multi-thread model: Uses 1 JVM
- Multi-process model: Uses 4 JVMs
- Multi-thread model was 5.5% faster
Core Scalability of SPECjbb2005

**on Niagara**

- Multi-thread model was 3.4% faster

**on Nehalem**

- Multi-thread model was 2.1% slower
Core Scalability and SMT Scalability on Niagara

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

using increasing number of SMT threads (up to 4 threads)

multi-process model = 1.0

multi-thread model is better

relative number of events for multi-thread model over multi-process model

- L1I cache miss
- L2 instruction miss
- L1D cache miss
- L2 data miss
- DTLB miss
Micro Architectural Statistics for SPECjbb2005

- 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

- core 0
- core 1
- core 2
- core 3

JVM

Java heap

1 GB = 256 MB x 4

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

multi-process (multi-JVM) model

- core 0
- core 1
- core 2
- core 3

JVM 0

JVM 1

JVM 2

JVM 3

Java heap

256 MB

- 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

multi-process PHP runtime (default)

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

multi-threaded PHP runtime

- process sharing virtual memory space
Core Scalability and SMT Scalability of MediaWiki

multi-thread model was 5.5% faster

consistent with results for Java benchmarks
Micro Architectural Statistics for MediaWiki

Using increasing number of SMT threads (up to 4 threads)

Multi-process model = 1.0

Relative number of events for multi-thread model over multi-process model:
- L1I cache miss
- L2 instruction miss
- L1D cache miss
- L2 data miss
- DTLB miss

Multi-process model is better
Multi-thread model is better

1 thread, 2 threads, 3 threads, 4 threads
Micro Architectural Statistics for MediaWiki

using increasing number of cores (up to 8 cores)

relative number of events for multi-thread model over multi-process model

multi-process model = 1.0

- 1 core
- 2 cores
- 4 cores
- 6 cores
- 8 cores

multi-process model is better

multi-thread model is better
Performance of MediaWiki using All SMT Threads

<table>
<thead>
<tr>
<th>core 0</th>
<th>core 1</th>
<th>core 2</th>
<th>core 3</th>
<th>core 4</th>
<th>core 5</th>
<th>core 6</th>
<th>core 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
</tr>
<tr>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
</tr>
<tr>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
</tr>
<tr>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
<td>thread</td>
</tr>
</tbody>
</table>

😊 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

core 0  core 1  core 2  core 3

multi-threaded PHP runtime

heap
Our Technique: Core-aware Memory Allocation

core 0  core 1  core 2  core 3
multi-threaded PHP runtime

physical page size (4 MB)
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