SMT Issues

- SMT CPU performance gain potential.
- Modifications to Superscalar CPU architecture necessary to support SMT.
- SMT performance evaluation vs. Fine-grain multithreading Superscalar, Chip Multiprocessors.
- Hardware techniques to improve SMT performance:
  - Optimal level one cache configuration for SMT.
  - SMT thread instruction fetch, issue policies.
  - Instruction recycling (reuse) of decoded instructions.
- Software techniques:
  - Compiler optimizations for SMT.
  - Software-directed register deallocation.
  - Operating system behavior and optimization.
- SMT support for fine-grain synchronization.
- SMT as a viable architecture for network processors.
- An SMT implementation: Intel’s Hyper-Threading (2-way SMT) Microarchitecture and performance in compute-intensive workloads.
Operating System Impact on SMT Architecture


• The SimOS environment adapted for SMT:
  – Alpha-based SMT CPU core added.
  – Digital Unix 4.0d modified to support SMT.

• Study goals:
  
  1. Compare SMT/OS performance results with previous SMT performance results that do not account for OS behavior and impact.
  2. Contrast OS impact between OS intensive and non OS intensive workloads.

• Two types of workloads selected for the study:
  – Non OS intensive workload: Multiprogrammed 8 SPECInt95 benchmarks.
  – OS intensive workload: Multi-threaded Apache web server (64 server processes), driven by the SPECWeb benchmark (128 clients).

• No SMT-specific OS optimizations were investigated in this study.
OS (Kernel) Code Vs. User Code

1. Operating systems usually have large working sets that can overwhelm the cache and TLB due to code and data size.
2. Operating systems may negatively impact branch prediction performance, because of frequent branches and infrequent loops.
3. OS execution is often brief and intermittent, invoked by interrupts, exceptions, or system calls, and can cause the replacement of useful cache, TLB and branch prediction state for little or no benefit.
4. The OS may perform spin-waiting, explicit cache/TLB invalidation, and other operations not common in user-mode code.
SimOS

- SimOS is a complete machine simulation environment developed at Stanford (http://simos.stanford.edu/).
- Designed for the efficient and accurate study of both uniprocessor and multiprocessor computer systems.
- Simulates computer hardware in enough detail to boot and run commercial operating systems.
- SimOS currently provides CPU models of the MIPS R4000 and R10000 and Digital Alpha processor families.
- In addition to the CPU, SimOs also models caches, multiprocessor memory busses, disk drives, ethernet, consoles, and other system devices.
- SimOs has been ported for IRIX versions 5.3 (32-bit) and 6.4 (64-bit) and Digital UNIX; a port of Linux for the Alpha and X86 is being developed.
SimOS System Diagram

Target Operating System and Workloads (ex. gcc running on IRIX 5.3)

SimOS

I/O Devices  CPU models  Memory Systems

Simulation Host

Alpha CPU model SMT-modified to support 8 threads
A Base SMT hardware Architecture.

### Simulated Alpha-based 8-way (thread) 8-issue SMT Processor Parameters

- Duplicate the register file, program counter, subroutine stack and internal processor registers of a superscalar CPU to hold the state of multiple threads.
- Add per-context mechanisms for pipeline flushing, instruction retirement, subroutine return prediction, and trapping.
- Fetch unit, Functional units, Data L1, L2, TLB shared among contexts.
- ~10% chip-area increase over superscalar. (compared to ~5% for 2-way SMT Intel’s hyper-threaded P4/Xeon)

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>9 stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch Policy</td>
<td>8 instructions per cycle from up to 2 contexts (the 2.8 ICOUNT scheme of [41])</td>
</tr>
<tr>
<td>Functional Units</td>
<td>6 integer (including 4 Load/Store and 2 Synchronization units)</td>
</tr>
<tr>
<td>Instruction Queues</td>
<td>32-entry integer and floating point queues</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>100 integer and 100 floating point</td>
</tr>
<tr>
<td>Retirement bandwidth</td>
<td>12 instructions/cycle</td>
</tr>
<tr>
<td>TLB</td>
<td>128-entry ITLB and DTLB</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>McFarling-style, hybrid predictor [26]</td>
</tr>
<tr>
<td>Local Predictor</td>
<td>4K-entry prediction table indexed by 2K-entry history table</td>
</tr>
<tr>
<td>Global Predictor</td>
<td>8K entries, 8K-entry selection table</td>
</tr>
<tr>
<td>Branch Target Buffer</td>
<td>1K entries, 4-way set associative</td>
</tr>
</tbody>
</table>

### Cache Hierarchy

<table>
<thead>
<tr>
<th>Cache Line Size</th>
<th>64 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icache</td>
<td>128KB, 2-way set associative, single port, 2 cycle fill penalty</td>
</tr>
<tr>
<td>Dcache</td>
<td>128KB, 2-way set associative, dual ported (only from CPU, r/w). Only 1 request at a time supported from the L2, 2 cycle fill penalty</td>
</tr>
<tr>
<td>L2 cache</td>
<td>16MB, direct mapped, 20 cycle latency, fully pipelined (1 access per cycle)</td>
</tr>
<tr>
<td>MSHR</td>
<td>32 entries for the L1 caches, 32 entries for the L2 cache</td>
</tr>
<tr>
<td>Store Buffer</td>
<td>32 entries</td>
</tr>
<tr>
<td>L1-L2 bus</td>
<td>256 bits wide, 2 cycle latency</td>
</tr>
<tr>
<td>Memory bus</td>
<td>128 bits wide, 4 cycle latency</td>
</tr>
<tr>
<td>Physical Memory</td>
<td>128MB, 90 cycle latency, fully pipelined</td>
</tr>
</tbody>
</table>
OS Modifications for SMT

Only minimal required OS modifications to support SMT considered (no OS optimizations for SMT considered here):

- OS task scheduler must support multiple threads in running status:
  - Shared-memory multiprocessor (SMP) aware OS (including Digital Unix) has this ability but each thread runs on a different physical CPU in SMP systems.
  - An SMT processor reports to such an OS as multiple shared memory CPUs (logical processors).

- TLB-related code must be modified:
  - Mutual exclusion support to access to address space number (ASN) tags of the TLB by multiple threads simultaneously.
  - Modified ASN assignment to account for the presence of multiple threads.
  - Internal CPU registers used to modify TLB entries replicated per context.

- No OS changes required to account for the shared L1 cache of SMT vs. the non shared L1 for SMP.
SPECInt95 Workload Execution Cycle Breakdown

- Percentage of execution cycles for OS Kernel instructions:
  - During program startup: 18%, mostly due to data TLB misses and to a lesser extent system calls.
  - Steady state: 5% still dominated by TLB misses.

Selected to represent non OS intensive workloads

SMT-7

"An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture", Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Breakdown of Kernel Time for SPECInt95

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
SPEC System Calls Percentage

System calls as a percentage of total execution cycles.

SMT-7

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
### SPECInt95 Dynamic Instruction Mix

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Program Start-up</th>
<th>Steady State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User</td>
<td>Kernel</td>
</tr>
<tr>
<td>Load</td>
<td>19.5</td>
<td>16.5(51%)</td>
</tr>
<tr>
<td>Store</td>
<td>12.3</td>
<td>19.0(57%)</td>
</tr>
<tr>
<td>Branch</td>
<td>15.1</td>
<td>15.9</td>
</tr>
<tr>
<td>Conditional</td>
<td>(64%)</td>
<td>65.9</td>
</tr>
<tr>
<td>Unconditional</td>
<td>19.5</td>
<td>14.1</td>
</tr>
<tr>
<td>Indirect Jump</td>
<td>14.7</td>
<td>11.7</td>
</tr>
<tr>
<td>PAL call/return</td>
<td>.01</td>
<td>8.9</td>
</tr>
<tr>
<td>Total</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Remaining Integer</td>
<td>50.0</td>
<td>48.6</td>
</tr>
<tr>
<td>Floating Point</td>
<td>3.1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

- Percentage of dynamic instructions in the SPECInt workload by instruction type.
- The percentages in parenthesis for memory operations represent the proportion of loads and stores that are to physical addresses.
- A percentage breakdown of branch instructions is also included.
- For conditional branches, the number in parenthesis represents the percentage of conditional branches that are taken.

**SMT-7**

"An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture", Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
**SPECInt95 Total Miss rates & Distribution of Misses**

<table>
<thead>
<tr>
<th>Cause of misses</th>
<th>Percentage of Misses Due to Conflicts (sums to 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrathread conflicts</td>
<td>51.0   4.9</td>
</tr>
<tr>
<td>Interthread conflicts</td>
<td>39.5   1.1</td>
</tr>
<tr>
<td>User-kernel conflicts</td>
<td>1.8    1.7</td>
</tr>
<tr>
<td>Invalidation by the OS</td>
<td>.40    .9</td>
</tr>
<tr>
<td>Compulsory</td>
<td>.01    .01</td>
</tr>
</tbody>
</table>

- The miss categories are percentages of all user and kernel misses.
- Bold entries signify kernel-induced interference.
- User-kernel conflicts are misses in which the user thread conflicted with some type of kernel activity (the kernel executing on behalf of this user thread, some other user thread, a kernel thread, or an interrupt).

**SMT-7**

"An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture", Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Metrics for SPECInt95 with and without the Operating System for both SMT and Superscalar (Steady State)

<table>
<thead>
<tr>
<th>Metric</th>
<th>SMT SPEC only</th>
<th>SPEC+OS</th>
<th>Change</th>
<th>Superscalar SPEC only</th>
<th>SPEC+OS</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC</td>
<td>5.9</td>
<td>5.6</td>
<td>-5%</td>
<td>3.0</td>
<td>2.6</td>
<td>-15%</td>
</tr>
<tr>
<td>Average # fetchable contexts</td>
<td>7.7</td>
<td>7.1</td>
<td>-8%</td>
<td>1.0</td>
<td>0.8</td>
<td>-20%</td>
</tr>
<tr>
<td>Branch misprediction rate (%)</td>
<td>8.1</td>
<td>9.3</td>
<td>15%</td>
<td>5.1</td>
<td>5.0</td>
<td>-2%</td>
</tr>
<tr>
<td>Instructions squashed (% of instructions fetched)</td>
<td>15.1</td>
<td>18.2</td>
<td>21%</td>
<td>31.8</td>
<td>32.3</td>
<td>2%</td>
</tr>
<tr>
<td>L1 icache miss rate (%)</td>
<td>1.0</td>
<td>2.0</td>
<td>190%</td>
<td>0.1</td>
<td>1.3</td>
<td>1300%</td>
</tr>
<tr>
<td>L1 Dcache miss rate (%)</td>
<td>3.2</td>
<td>3.6</td>
<td>15%</td>
<td>0.6</td>
<td>0.5</td>
<td>-15%</td>
</tr>
<tr>
<td>L2 miss rate (%)</td>
<td>1.1</td>
<td>1.4</td>
<td>27%</td>
<td>1.0</td>
<td>1.8</td>
<td>72%</td>
</tr>
<tr>
<td>ITLB miss rate (%)</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>DTLB miss rate (%)</td>
<td>0.4</td>
<td>0.6</td>
<td>36%</td>
<td>0.04</td>
<td>0.05</td>
<td>25%</td>
</tr>
</tbody>
</table>

- The maximum issue for integer programs is 6 instructions on the 8-wide SMT, because there are only 6 integer units.

SMT-7

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Apache experiences little start-up period since Apache’s ‘start-up’ consists simply of receiving the first incoming requests and waking up the server threads.

- Once requests arrive, Apache spends over 75% of its time in the OS kernel.
Breakdown of kernel time for Apache vs. SPECInt95 on SMT

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000

SMT-7

Majority TLB miss handing
Apache System Calls By Name

Apache System Calls by Name

% of All Execution Cycles

0 5 10 15 20 25 30 35 40 45 50

Apache

Other
smmap
munmap
stat
read
write
writev
close
accept
select
open
Kernel preamble
PAL code

Memory Mapped Files

I/O

I/O Control

Dominate System Calls

SMT-7

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Apache System Calls By Function

SMT-7

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Apache Dynamic Instruction Mix

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>User</th>
<th>Kernel (in %)</th>
<th>Overall (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>21.8</td>
<td>19.9 (54.2)</td>
<td>20.3 (42.0)</td>
</tr>
<tr>
<td>Store</td>
<td>10.1</td>
<td>11.5 (40.3)</td>
<td>11.2 (32.7)</td>
</tr>
<tr>
<td>Branch</td>
<td>16.7</td>
<td>17.8</td>
<td>17.6</td>
</tr>
<tr>
<td>Conditional</td>
<td></td>
<td>(54%) 70.6</td>
<td>(52%) 66.2</td>
</tr>
<tr>
<td>Unconditional</td>
<td></td>
<td>12.9</td>
<td>15.4</td>
</tr>
<tr>
<td>Indirect Jump</td>
<td></td>
<td>16.3</td>
<td>14.2</td>
</tr>
<tr>
<td>PAL call/return</td>
<td></td>
<td>0.2</td>
<td>4.2</td>
</tr>
<tr>
<td>Total</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Remaining Int.</td>
<td>51.4</td>
<td>50.8</td>
<td>50.9</td>
</tr>
<tr>
<td>Floating Point</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

- The percentages in parenthesis for memory operations represent the proportion of loads and stores that are to physical addresses.
- A percentage breakdown of branch instructions is also included.
- For conditional branches, the number in parenthesis represents the percentage of conditional branches that are taken.

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al., in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
### Metrics for SMT SPEC, Apache & Superscalar Apache

<table>
<thead>
<tr>
<th>Metric</th>
<th>SMT Apache</th>
<th>SMT SPEC steady-state</th>
<th>Superscalar Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC</td>
<td>4.6</td>
<td>5.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Instructions squashed (% of instructions fetched)</td>
<td>26.9</td>
<td>18.2</td>
<td>45.9</td>
</tr>
<tr>
<td>Avg. # of fetchable contexts</td>
<td>5.7</td>
<td>7.1</td>
<td>.4</td>
</tr>
<tr>
<td>Branch mispredict. rate (%)</td>
<td>9.1</td>
<td>9.3</td>
<td>7.4</td>
</tr>
<tr>
<td>ITLB miss rate (%)</td>
<td>.8</td>
<td>.0</td>
<td>.7</td>
</tr>
<tr>
<td>DTLB miss rate (%)</td>
<td>0.6</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>L1 Icache miss rate (%)</td>
<td>5.0</td>
<td>2.0</td>
<td>6.5</td>
</tr>
<tr>
<td>L1 Dcache miss rate (%)</td>
<td>8.4</td>
<td>3.6</td>
<td>3.4</td>
</tr>
<tr>
<td>L2 miss rate (%)</td>
<td>2.1</td>
<td>1.4</td>
<td>1.5</td>
</tr>
<tr>
<td>0-fetch cycles (%)</td>
<td>13.8</td>
<td>6.6</td>
<td>65.0</td>
</tr>
<tr>
<td>0-issue cycles (%)</td>
<td>3.1</td>
<td>0.6</td>
<td>62.4</td>
</tr>
<tr>
<td>Max. (6) issue cycles (%)</td>
<td>58.2</td>
<td>87.1</td>
<td>6.3</td>
</tr>
<tr>
<td>Avg. # of outstanding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I$ misses</td>
<td>1.9</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>D$ misses</td>
<td>2.7</td>
<td>1.2</td>
<td>0.3</td>
</tr>
<tr>
<td>L2$ misses</td>
<td>1.3</td>
<td>1.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

All applications are executing with the operating system.

---

"An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture", Josh Redstone et al., in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Apache+OS Total Miss rates & Distribution of Misses

<table>
<thead>
<tr>
<th>Cause of the misses</th>
<th>Percentage of Misses Due to Conflicts (sums to 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Branch Target Buffer</td>
</tr>
<tr>
<td></td>
<td>User</td>
</tr>
<tr>
<td>Intrathread conflicts</td>
<td>12.2</td>
</tr>
<tr>
<td>Intertthread conflicts</td>
<td>.4</td>
</tr>
<tr>
<td>User-kernel conflicts</td>
<td>2.4</td>
</tr>
<tr>
<td>Invalidation by the OS</td>
<td>.5</td>
</tr>
<tr>
<td>Compulsory</td>
<td>.1</td>
</tr>
</tbody>
</table>

- The miss categories are percentages of all user and kernel misses.
- Bold entries signify kernel-induced interference.
- User-kernel conflicts are misses in which the user thread conflicted with some type of kernel activity (the kernel executing on behalf of this user thread, some other user thread, a kernel thread, or an interrupt).

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
Percentage of Misses Avoided Due to Interthread Cooperation on Apache

<table>
<thead>
<tr>
<th>Mode that would have missed</th>
<th>Misses avoided due to interthread prefetching as a percentage of total misses</th>
<th>SMT-7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Branch Target Buffer</td>
<td>L1 Instruction Cache</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Kernel</td>
</tr>
<tr>
<td>Apache - SMT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kernel</td>
<td>0</td>
<td>19.5</td>
</tr>
<tr>
<td>Apache - Superscalar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kernel</td>
<td>0</td>
<td>1.9</td>
</tr>
</tbody>
</table>

- Percentage of misses avoided due to interthread cooperation on Apache, shown by execution mode.
- The number in a table entry shows the percentage of overall misses for the given resource that threads executing in the mode indicated on the leftmost column would have encountered, if not for prefetching by other threads executing in the mode shown at the top of the column.

"An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture", Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000
## OS Impact on Hardware Structures

### Performance for Apache

“An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture”, Josh Redstone et al. in Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000

<table>
<thead>
<tr>
<th>Metric</th>
<th>SMT</th>
<th>Superscalar</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Apache only</td>
<td>Apache+OS</td>
</tr>
<tr>
<td>Branch misprediction rate (%)</td>
<td>4.4</td>
<td>9.1</td>
</tr>
<tr>
<td>BTB misprediction rate (%)</td>
<td>36.7</td>
<td>59.6</td>
</tr>
<tr>
<td>L1 Icache miss rate (%)</td>
<td>0.9</td>
<td>5.0</td>
</tr>
<tr>
<td>L1 Dcache miss rate (%)</td>
<td>6.2</td>
<td>8.4</td>
</tr>
<tr>
<td>L2 miss rate (%)</td>
<td>0.6</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Thus must account for OS in SMT processor simulation for server applications (Apache here)

Or other types of OS intensive workloads

---

"EECC722 - Shaaban"
OS Impact on SMT Study Summary

- Results show that for SMT, omission of the operating system did not lead to a serious misprediction of performance for SPECInt95 (selected here to represent non OS intensive workloads), although the effects were more significant for a superscalar executing the same workload.

- On the Apache workload (selected here to represent OS intensive workloads), however, the operating system is responsible for the majority of instructions executed:
  1. Apache spends a significant amount of time responding to system service calls in the file system and kernel networking code.
  2. The result of the heavy execution of OS code is an increase of pressure on various low-level resources, including the caches and the BTB.
  3. Kernel threads also cause more conflicts in those resources, both with other kernel threads and with user threads; on the other hand, there is a positive interthread sharing effect as well.

Thus Must account for OS in SMT processor simulations
Possible SMT-specific OS Optimizations

- Smart SMT-optimized OS task scheduler for better SMT-core performance:
  - Schedule cooperating threads that benefit from SMT’s resource and data sharing to run simultaneously.
  - To aid SMT’s latency-hiding, avoid scheduling too many threads that have conflicts over same specific CPU resource (TLB, cache, FP etc.)
  - For SMP-SMT system tightly-coupled threads should be scheduled to logical processors in the same physical SMT CPU (processor affinity).

- Introduce a lightweight dedicated kernel context to cached in the SMT-core to handle process management and speedup system calls.

- Prevent the “idle loop” thread from consuming execution resources:
  - Intel Hyper-threading solution: use HALT instruction.

- Allow thread caching in the CPU to further reduce context-switching overheads.
Overview of Intel’s Hyper-Threading (HT) Microarchitecture & Performance

- Introduced in 2002 by adding 2 thread SMT (Hyper-Threading, HT) support to the Intel Xeon/P4 Northwood core (NetBurst microarchitecture).

- **Major Design Goals:**
  1. Minimize increase of chip area and complexity. Implementation increased relative chip area/power by 5%
  2. Prevent a stalling thread from affecting the progress of the other thread in the CPU.
  3. Maintain a good single-thread performance by switching to single thread mode.

- **To limit increase in chip area and complexity:**
  1. Each thread is only allocated a maximum of 50% of:
     - Fetch cycles, micro-ops queues, instruction scheduling queues, load/store buffers, re-order buffer entries, instruction retirement cycles.
  2. No resources (TLB, cache, queues, buffers, etc.) were resized to account for the additional demands SMT poses on shared CPU resources.

- These design constraints prevented this implementation from achieving the full performance potential of a 2 thread full SMT processor.
  - Performance gains of this implementation typically range from 5% to 28% vs. Potentially 15% - 60% gain for a 2 thread SMT without restrictions.
Intel Xeon/P4 HT Processor Pipeline

- Visible to OS as two logical processors when HT is enabled
- Duplicated for each thread:
  - Architectural thread state
  - Advanced programmable Interrupt Controller (APIC)

SMT-8
Front-End Detailed Pipeline

(a) Instruction Trace Cache Hit:
Pre-decoded instruction micro-ops fetched from trace cache
Fetch is round robin from each thread (e.g. RR1.xx)
Each thread limited to 50% of micro-ops queue ready for renaming/scheduling

(b) Instruction Trace Cache Miss:
Instructions fetched from L2 cache in round robin RR1.X
Instruction decoding into micro-ops alternates between threads every cycle
Instruction micro-ops filled in trace cache and micro-ops queue ready for renaming/scheduling
Out-Of-Order Execution Engine Pipeline (Core)

In-order Rename Resource Allocation

Out-of-order Schedule/Execute

In-order Retirement

Uop Queue

Rename

Queue

Sched

Register Read

Execute

L1 Cache

Register Write

Retire

50% of entries per thread

128 rename (physical registers)

Memory micro-ops

Other micro-ops

No thread allocation constraints

Retire up to 3 micro-ops/cycle alternating between threads

SMT-8

EECC722 - Shaaban
HT Transaction Processing Performance
(Server application)

Figure 8: Performance increases from Hyper-Threading Technology on an OLTP workload (Transaction processing workload)
HT Web Server Benchmark Performance

16-21% performance gain for HT

SMT-8
Summary of HT Performance For Compute-Intensive Workloads

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Mechanical Design Analysis (finite element method) This application is used for metal-forming, drop testing, and crash simulation.</td>
</tr>
<tr>
<td>A2</td>
<td>Genetics A genetics application that correlates DNA samples from multiple animals to better understand congenital diseases.</td>
</tr>
<tr>
<td>A3</td>
<td>Computational Chemistry This application uses the self-consistent field method to compute chemical properties of molecules such as new pharmaceuticals.</td>
</tr>
<tr>
<td>A4</td>
<td>Mechanical Design Analysis This application simulates the metal-stamping process.</td>
</tr>
<tr>
<td>A5</td>
<td>Mesoscale Weather Modeling This application simulates and predicts mesoscale and regional-scale atmospheric circulation.</td>
</tr>
<tr>
<td>A6</td>
<td>Genetics This application is designed to generate Expressed Sequence Tags (EST) clusters, which are used to locate important genes.</td>
</tr>
<tr>
<td>A7</td>
<td>Computational Fluid Dynamics This application is used to model free-surface and confined flows.</td>
</tr>
<tr>
<td>A8</td>
<td>Finite Element Analysis This finite element application is specifically targeted toward geophysical engineering applications.</td>
</tr>
<tr>
<td>A9</td>
<td>Finite Element Analysis This explicit time-stepping application is used for crash test studies and computational fluid dynamics.</td>
</tr>
</tbody>
</table>

SMP speedup range 1.65 to 2.00
HT (SMT) speedup range 1.05 to 1.28

Why?
Summary of HT Performance For Compute-Intensive Workloads

<table>
<thead>
<tr>
<th>Application</th>
<th>CPI</th>
<th>Ideal 1/3 cycle/uop</th>
<th>FP%</th>
<th>One HT core SMT speedup</th>
<th>Two Physical cores SMP Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>2.04</td>
<td>1.47</td>
<td>29</td>
<td>1.05</td>
<td>1.65</td>
</tr>
<tr>
<td>A2</td>
<td>1.11</td>
<td>0.89</td>
<td>4.6</td>
<td>1.09</td>
<td>1.79</td>
</tr>
<tr>
<td>A3</td>
<td>1.69</td>
<td>0.91</td>
<td>16</td>
<td>1.09</td>
<td>1.77</td>
</tr>
<tr>
<td>A4</td>
<td>1.82</td>
<td>1.29</td>
<td>20</td>
<td>1.11</td>
<td>1.54</td>
</tr>
<tr>
<td>A5</td>
<td>2.48</td>
<td>1.45</td>
<td>36</td>
<td>1.17</td>
<td>1.68</td>
</tr>
<tr>
<td>A6</td>
<td>2.54</td>
<td>1.60</td>
<td>0.1</td>
<td>1.19</td>
<td>2.00</td>
</tr>
<tr>
<td>A7</td>
<td>2.80</td>
<td>2.05</td>
<td>10</td>
<td>1.23</td>
<td>1.75</td>
</tr>
<tr>
<td>A8</td>
<td>1.69</td>
<td>1.27</td>
<td>19</td>
<td>1.28</td>
<td>1.85</td>
</tr>
<tr>
<td>A9</td>
<td>2.26</td>
<td>1.76</td>
<td>20</td>
<td>1.28</td>
<td>1.89</td>
</tr>
</tbody>
</table>

SMP speedup range 1.65 to 2.00
SMT speedup range 1.05 to 1.28

Why?

This SMT implementation cannot match or exceed SMP performance due to:
1- Thread resource/cycle constraints imposed in Intel’s hyper-threading implementation
2- No low-level hardware resources resizing for additional thread demands.