Hyperthreading Technology

Aleksandar Milenkovic
Electrical and Computer Engineering Department
University of Alabama in Huntsville
milenka@ece.uah.edu
www.ece.uah.edu/~milenka/

Outline

- What is hyperthreading?
- Trends in microarchitecture
- Exploiting thread-level parallelism
- Hyperthreading architecture
- Microarchitecture choices and trade-offs
What is hyperthreading?

- **SMT - Simultaneous multithreading**
  - Make one physical processor appear as multiple logical processors to the OS and software
- **Intel Xeon for the server market, early 2002**
- **Pentium4 for the consumer market, November 2002**
- **Motivation:** boost performance for up to 25% at the cost of 5% increase in additional die area

Trends in microarchitecture

- **Higher clock speeds**
  - To achieve high clock frequency make pipeline deeper (superpipelining)
  - Events that disrupt pipeline (branch mispredictions, cache misses, etc) become very expensive in terms of lost clock cycles
- **ILP: Instruction Level Parallelism**
  - Extract parallelism in a single program
  - Superscalar processors have multiple execution units working in parallel
  - Challenge to find enough instructions that can be executed concurrently
  - Out-of-order execution => instructions are sent to execution units based on instruction dependencies rather than program order
Trends in microarchitecture

- Cache hierarchies
  - Processor-memory speed gap
  - Use caches to reduce memory latency
  - Multiple levels of caches: smaller and faster closer to the processor core

- Thread-level Parallelism
  - Multiple programs execute concurrently
    - Web-servers have an abundance of software threads
    - Users: surfing the web, listening to music, encoding/decoding video streams, etc.

Exploiting thread-level parallelism

- CMP – Chip Multiprocessing
  - Multiple processors, each with a full set of architectural resources, reside on the same die
  - Processors may share an on-chip cache or each can have its own cache
  - Examples: HP Mako, IBM Power4
  - Challenges: Power, Die area (cost)

- Time-slice multithreading
  - Processor switches between software threads after a predefined time slice
  - Can minimize the effects of long lasting events
  - Still, some execution slots are wasted
Exploiting thread-level parallelism

- **Switch-on-event multithreading**
  - Processor switches between software threads after an event (e.g., cache miss)
  - Works well, but still coarse-grained parallelism (e.g., data dependences and branch mispredictions are still wasting cycles)

- **SMT – Simultaneous multithreading**
  - Multiple software threads execute on a single processor without switching
  - Have potential to maximize performance relative to the transistor count and power

---

Hyperthreading architecture

- One physical processor appears as multiple logical processors
- HT implementation on NetBurst microarchitecture has 2 logical processors

<table>
<thead>
<tr>
<th>Architectural State</th>
<th>Architectural State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor execution resources</td>
<td>Processor execution resources</td>
</tr>
</tbody>
</table>

Architectural state:
- general purpose registers
- control registers
- APIC: advanced programmable interrupt controller
Hyperthreading architecture

- Main processor resources are shared
  - caches, branch predictors, execution units, buses, control logic

- Duplicated resources
  - register alias tables (map the architectural registers to physical rename registers)
  - next instruction pointer and associated control logic
  - return stack pointer
  - instruction streaming buffer and trace cache fill buffers

Die Size and Complexity

Figure 2. Intel Pentium 4 and the visible processor resources duplicated to support hyperthreading technology. Hyperthreading requires duplication of additional miscellaneous pointers and control logic, but these are too small to point out.
Resources sharing schemes

- **Partition** - dedicate equal resources to each logical processors
  - Good when expect high utilization and somewhat unpredicatable
- **Threshold** - flexible resource sharing with a limit on maximum resource usage
  - Good for small resources with bursty utilization and when the micro-ops stay in the structure for short predictable periods
- **Full sharing** - flexible with no limits
  - Good for large structures, with variable working-set sizes

---

Shared vs. partitioned queues

Figure 3. Comparison of a shared and a partitioned queue. The slower (dark) thread has a downstream stall, such as a data-cache miss. In this situation, the queues will not send any slower micro-ops to the next pipeline stage. The figure shows how the queues will progress through cycles 0 (a), 1 (b), 2 (c), and 3 (d). Eventually, in cycle 4 (e) the shared queue lets a slower thread block the progress of the faster (light) thread.
NetBurst Pipeline

Shared vs. partitioned resources

- Partitioned
  - E.g., major pipeline queues

- Threshold
  - Puts a threshold on the number of resource entries a logical processor can have
  - E.g., scheduler

- Fully shared resources
  - E.g., caches
    - Modest interference
    - Benefit if we have shared code and/or data

Figure 4. In this view of a Netburst microarchitecture’s execution pipeline, the light and dark areas indicate the resource utilization of the two software threads running on the two logical processors.
Scheduler occupancy

Figure 5. Snapshot of scheduler occupancy on a transaction processing workload over a short period of time. Each data point is the instantaneous scheduler occupancy for its respective logical processor, measured by the number of entries occupied by each thread.

Shared vs. partitioned cache

Figure 6. Cache hit rate and overall performance impact for a fully shared cache normalized against values for a partitioned cache. On average, the shared cache had a 40 percent better cache hit rate and 12 percent better performance. Notice that no single application workload lost performance because of the shared cache.
Performance Improvements

<table>
<thead>
<tr>
<th>Software</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMPEG with DivX Video</td>
<td>26%</td>
</tr>
<tr>
<td>Adobe Photoshop Imaging</td>
<td>25%</td>
</tr>
<tr>
<td>Magix MP3 Maker Audio</td>
<td>24%</td>
</tr>
<tr>
<td>Adobe After Effects Video</td>
<td>15%</td>
</tr>
</tbody>
</table>

Figure 7. Hyperthreading technology performance gains on several popular multitthreaded software packages.

<table>
<thead>
<tr>
<th>Software</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word, Acrobat, and Virus Scan</td>
<td>27%</td>
</tr>
<tr>
<td>PowerPoint, Acrobat, and Virus Scan</td>
<td>20%</td>
</tr>
<tr>
<td>Windows Movie Maker and Magix MP3 Maker</td>
<td>18%</td>
</tr>
<tr>
<td>Magix MP3 Maker and Virus Scan</td>
<td>15%</td>
</tr>
</tbody>
</table>

Figure 8. Hyperthreading technology performance boost on multitasking workloads.