ILP Ends TLP Begins

Today's topics:
Explore a perfect machine
  unlimited budget to see where ILP goes
  answer: not far enough
Look to TLP & multi-threading for help
  everything has it's issues
  we'll look at some of them
Apology
  a bit more data than usual
  try not to yawn LOUDLY

ILP Limits via an Oracle

- Suspend reality and think of a perfect machine
  - infinite number of rename registers
    » no Wax hazards
    » for window size of \( n^2 \) comparisons for each register field
  - perfect branch & jump prediction
    » unbounded buffer of instructions available for execution
  - perfect address alias analysis
    » independent loads can be moved ahead of stores
  - perfect L1's
    » hit in 1 cycle
  - as many XU's as will ever be needed
    » no structural stalls
- infinite cost unrealistic
  - simulate rather than build
    » allows exploration
    - just how far can we get with ILP on a perfect machine
    - and sequential code

IBM Power5

- Most advanced superscalar processor to date
  - 4 fetch
  - 6 issue
  - 88 integer rename regs, 88 float rename regs
  - pipeline has over 200 instructions in flight
    » including 32 loads and 32 stores
- Not quite the Oracle but on the way
  - consumes a lot of power
  - target is blade server segment

ILP w/ Infinite Window Size

- Looks great
  - when compared w/ today's IPC < 3
    » if you can ignore the infinite cost

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Instruction issues per cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>55</td>
</tr>
<tr>
<td>espresso</td>
<td>63</td>
</tr>
<tr>
<td>fppp</td>
<td>18</td>
</tr>
<tr>
<td>doduc</td>
<td>75</td>
</tr>
<tr>
<td>tomcaty</td>
<td>119</td>
</tr>
</tbody>
</table>

SPECBenchmarks

©2007 Steven M. S. Ghanbari
Limit Window Size

- ILP shrinks rapidly

<table>
<thead>
<tr>
<th>Window Size</th>
<th>2K</th>
<th>512</th>
<th>128</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-bit compares</td>
<td>8M</td>
<td>2M</td>
<td>0.5M</td>
<td>0.13M</td>
</tr>
</tbody>
</table>

2K = ~4M * 3 5-bit compares
512 = 785K compares
128 = 49K compares
32 = 3K compares

Plus compares happen every cycle

conclude 32 is doable but watty

can improve by maybe 3x at great cost (remember this is still a perfect machine – just wi a limited window)

Switch to Half-Infinity

- You do the math
- For the remaining data assume
  - 2K window size
    - ~12 M 5-bit compares every clock
    - >10x bigger than anything that’s been built
  - 64 issue
    - ~<10x more than anything real
- Why choose this
  - given other restrictions it won’t be a limit
    - can you say *easier to simulate*?
    - *I knew you could*

Look at Semi-Real Branch Prediction

Tmnt: 8K entry predictor
Jump predictor: 2K entries
48K bits and 3% mispredict rate which is very good – just expensive

Standard: 512 entry 2-bit predictor

Conclusion:
- have to predict
- integer codes are a problem yet highly important in modern data-center apps
- Nobody makes much money on floating point – sad reality

Limiting Rename Registers

Integer codes remain problematic

$ problem with FP remains but ILP looks good if you don’t care about $

conclusion – need around 64 renamed registers to make much of a difference
**Alias Analysis Influence**

GL/STK – heap ref’s conflict
but nothing else

Inspection – what can the compiler do?

---

**Ambitious but Possible?**

- **HAL**
  - 1 better than IBM in all letters
  - 64 issue no restrictions
    - this one is actually ridiculous
    - it does focus on ILP limits rather than structural stalls
  - 1K entry tournament predictor
    - this has been done
  - perfect disambiguation
    - note close to possible for small window sizes
    - impractical for large windows
  - 64 register rename pool
    - ~100 have been done
- **What do we get?**
  - Al the Harpy says “a really good heater”

---

**And VIOLA!**

- 3-4x Improvement
  - for a machine nobody will buy – too hot, too costly

Interesting study & clear conclusion:

ILP is already past the point of diminishing return

Programmer is going to need to help out w/ exposing parallelism

Need a different type of HW support for parallelism

---

**Enter TLP**

- Again not a new idea
  - been around for > 10 years
    - Tullsen – UW – 1995 publishes the SMT idea
    - TERA MTA & IBM Pulsar show up in late 90’s – both MT
- **Thread vs. Process confusion**
  - process runs in it’s own virtual memory space
    - no shared memory
    - lots of OS protection & overhead
    - communicate via “message like channels” – e.g. pipes in Unix
  - threads
    - share memory and therefore synchronization needed
    - both are independent entities
      - with their own sets of registers and process state
  - **TLP difference**
    - multiple threads can run concurrently or interleaved on the same processor
    - one at a time and context switch for processes
Multi-Threading

- 2 variants
  - fine-grained MT – e.g. TERA
    - round robin walk through threads
    - next cycle – next thread
    - TERA – 128 threads
      - built in 128 cycle load-use delay
      - basic idea was to cover main memory latency and do away w/ caches
      - great if you put every app into a 128 thread mold
    - TERA – 128 threads
      - it failed and Burton goes to the dark side (a.k.a. Microsoft)
  - coarse grained MT – e.g. IBM Pulsar
    - sometimes called “switch on miss”
    - basic idea
      - anytime something bad happens
        - TLB or L2 miss
        - switch to next runnable thread
        - some sort of fairness policy is required
        - usually just round-robin suffices
        - similar goal – hide performance effect of long stalls

Symmetric Multithreading

- Idea
  - multiple independent threads
  - increase number of parallel instructions to issue

SMT Resource Perspective

- Each thread has its own
  - PC, next PC
    - next is needed for exceptions
  - private logical registers
    - and mapping to renamed physical registers
  - ROB
    - if shared a stall in one thread will stall the others
- Shared
  - branch predictor
    - larger size will be needed
  - main memory ports, TLB, page table
    - artifact of shared memory
  - more threads does increase memory pressure
    - biggest problem is single ported L1P's

SMT Pipeline Structure

- Shared Front-end
- Private Front-end
- What about RAS, LSQ?
SMT Issues

- Single thread performance goes down
  - competition with other threads for resources
  - resource utilization goes up
    - hence throughput goes up
- Fetch who?
  - which thread has priority?
    - unless set by user dynamic critical path can't be known in a small window
    - setting LSQ and ROB partition sizes is one way of implementing a priority in later stages
    - not so simple in Fetch
- ICOUNT
  - widely accepted heuristic
  - fetch each thread to roughly equalize processor resources
- better methods possible
  - BUT beware of creeping complexity
  - power and validation costs can fall off a cliff

---

4 Modernish Processors

<table>
<thead>
<tr>
<th>CPU</th>
<th>uArch</th>
<th>Fetch/Issue/Ex</th>
<th>XU's</th>
<th>Clock (GHz)</th>
<th>T's &amp; area</th>
<th>Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pent 4 Extreme</td>
<td>Spec. Dyn. Issue, deep pipe, 2way SMT</td>
<td>3/3/4</td>
<td>7 int 1 FP</td>
<td>3.8</td>
<td>125M 122 mm²</td>
<td>115</td>
</tr>
<tr>
<td>Athlon 64 FX-57</td>
<td>Spec. Dyn Issue</td>
<td>3/3/4</td>
<td>6 int 3 FP</td>
<td>2.8</td>
<td>114M 115 mm²</td>
<td>104</td>
</tr>
<tr>
<td>1 Core of Power5</td>
<td>Spec. Dyn Issue, SMT</td>
<td>6/4/6</td>
<td>6 int 3 FP</td>
<td>1.9</td>
<td>200M 300 mm²</td>
<td>80</td>
</tr>
<tr>
<td>Itanium 2 EPIC, mostly static sched</td>
<td>6/5/11</td>
<td>9 int 2 FP</td>
<td>1.6</td>
<td>592M 423 mm²</td>
<td>130</td>
<td></td>
</tr>
</tbody>
</table>

Power5 is dual core – area, T's, power estimated for single core; large die size is due to 9 MB L3 cache on chip.
Note: as we move into multi-core perf/watt becomes the critical efficiency metric
   Even better: energy-delay product since that is architecture and workload specific

---

**Efficiency**

<table>
<thead>
<tr>
<th>SPECcint/M</th>
<th>SPECfp/M</th>
<th>SPECint/mm^2</th>
<th>SPECfp/mm^2</th>
<th>SPECint/Watt</th>
<th>SPECfp/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistor</td>
<td>Transistor</td>
<td>Transistor</td>
<td>Transistor</td>
<td>Transistor</td>
<td>Transistor</td>
</tr>
</tbody>
</table>

**Concluding Remarks**

- **SMT is a big boost to the ILP game**
  - uses previous skills in dynamic superscalar architecture
  - rules of thumb
    - double threads
      - 1.6x performance gain
      - ~10-15% power gain
  - where does the curve saturate
    - depends on workload
      - ~4 cores seems to be a sweet spot
      - time will tell
    - Sun Niagara Falls
      - 8 cores, 8 threads/core
      - simpler cores however
        - performs well in the data-center
- **Next**
  - leave processor side and examine the memory side
    - both need to be balanced and done right to win