## COURSE OUTLINE

<table>
<thead>
<tr>
<th>Trends</th>
<th>1. Introduction – Trend I</th>
<th>13/10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Trend II</td>
<td>20/10</td>
</tr>
<tr>
<td></td>
<td>Multi Core trend</td>
<td></td>
</tr>
<tr>
<td>Multiple engines</td>
<td>3. MC vs. MT</td>
<td>27/10</td>
</tr>
<tr>
<td></td>
<td>4. Hetero/ Accelerators/</td>
<td>3/11</td>
</tr>
<tr>
<td></td>
<td>5. MultiAmdhal (Tzhaee)</td>
<td>10/11</td>
</tr>
<tr>
<td></td>
<td>MT trend and Core</td>
<td></td>
</tr>
<tr>
<td>Core Topics</td>
<td>6. Mem Intensive Arch (Shahar)</td>
<td>17/11</td>
</tr>
<tr>
<td></td>
<td>7. Power management (Efi)</td>
<td>24/11</td>
</tr>
<tr>
<td></td>
<td>8. Front End – Trace $ and Decoded $</td>
<td>8/12</td>
</tr>
<tr>
<td></td>
<td>9. MT and Mem/Mem. Disambiguation</td>
<td>15/12</td>
</tr>
<tr>
<td></td>
<td>Global views</td>
<td></td>
</tr>
<tr>
<td>Implementation trends</td>
<td>10. Data Flow (Yoav)</td>
<td>22/12</td>
</tr>
<tr>
<td></td>
<td>11. Binary translations (Gadi)</td>
<td>29/12</td>
</tr>
<tr>
<td></td>
<td>Conclusions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12. Real world Core Example (Ronny)</td>
<td>5/1</td>
</tr>
<tr>
<td></td>
<td>13. Conclusions</td>
<td>12/1</td>
</tr>
<tr>
<td></td>
<td>19/1</td>
<td></td>
</tr>
</tbody>
</table>
Front End

Instruction Supply: Issues and solutions

– Fix length vs. variable length instructions
– Dynamic instruction stream
  – Decoded cache
  – Trace cache
  – Basic block caches
– Instruction Reuse
– SMC

Today's lecture: Comprehend Instruction supply concepts
References of the Day

- “Optimization of Instruction Fetch Mechanisms for high issue rates”
  T. Conte, K. Menezes, P. Mills, B. Patel, ISCA 22, June 1995
- “Enhancing the Trace cache Fetch Mechanism”
- “Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching”
  E. Rotenberg, S. Bennett, J. Smith, Univ of Wisconsin, Micro 29, Nov 1996
- “Fetch directed instruction prefetching”
  Reinman, G.; Calder, B.; Austin, T.; MICRO ’32 Nov 1999
- “eXtended block cache”
- “Filtering techniques to improve trace-cache efficiency”
- “Micro-operation cache: a power aware frontend for variable instruction length ISA”,
Instruction Supply

- BTB
- Instruction Cache
  - Instruction fetch/decode
  - Rename
  - Scheduler
- Execution Engines
- Data Supply
- Trace/Decoded cache
The Challenges

Provide adequate instruction supply despite of:

- Instruction Cache misses
- Branch miss-predictions
- Frequent branches
- Frequent cache line crossing
- Complex Instruction Decoding
Instruction Decode Bottleneck

• In Super scalar Machine we need: Decoding of several instructions each cycle:
  ● RISC : Fixed instruction length ➔ simple ➔ fixed width decoder
  ● CISC: variable instruction length ➔ Complex

☐ CISC solutions
  ● Wide dynamic decoding
  ● Annotated Instruction cache
  ● Decoded instruction cache
  ● …
“Brute force” Instruction Decode

Superscalar RISC Decoder

Superscalar CISC Decoder (Wide decoder)

Unknown instruction starting locations: Need to decode all possibilities!
Instruction Decode

Superscalar CISC Decoder (example: Annotated Cache)
The Generic Processor
Decoded cache

- Instruction Cache
- Instruction fetch/decode
- Rename
- Scheduler
- Trace/decoded cache
- Execution engine
- BTB
- Data supply
Issue: Fetch Bandwidth

- **Solutions:**
  - Block Structured ISA
  - Multiported Instruction cache
  - Fetch Ahead mechanism

- **Problems**
  - Compiler based schemes require ISA enhancements
  - HW schemes require Multiported Instruction Cache
  - Wrong fetch ahead penalty and latency
Instruction Prefetching

- **Tagged next line prefetching**
  - Tag each cache line with a prefetch bit
  - Line n’th prefetch bit indicates whether next line (n+1) need to be prefetched

- **Target & wrong path prefetching**
  Similar to branch prediction – but in cache line granularity
  - Prefetching lines based on the predicted path
  - Prefetching either fall-through or taken paths

- **Streaming buffers**
  - A fifo queue
  - For I cache miss: Prefetch sequential cache blocks starting with the one that missed into the streaming buffer until it filled
  - Lookup in parallel in stream buffers and the I cache
Decoded Instruction Cache

Captures explicit instruction info:

- Instruction boundaries
- engine explicit signals
  - Address field
  - instruction type
  - register control
  - Flags Control
  - Branch hints
  - 
- Fully decoded instructions (or uops)
  - Requires complex address mapping
Decoded Instruction Cache (DIC)

- Can “resolve” CISC instruction length problem
- Reduces engine flush penalty by eliminating/reducing decoding stage
- **Power aspect** - avoid repeated costly decoding activity
- **Die area impact** - depends on the decoding level
- Big **performance** delta when miss Decoded cache (=> glass jaws programs)
- Different implementations: PPC620, Pentium, AMD K6, ...
Decoded Instruction Cache (DIC)

Decoded Instruction Cache implications - size

• Partial decoding - add several bits for each Instruction cache byte
  • Start/end marks + Decoding hints
• Full decoding – provide “complete” decoded information
  • Start/end information, full micro-operation cache, partial (?) microcode information
Fetch bandwidth

Instruction Fetch is an issue:

- Empty fetch (Cache misses)
- Partial fetch (branches)
- Fetch thrown away (mispredictions)
Micro-Operation Cache

• Lines of sequential instructions block are stored
• Already pre-decoded into uops
• Degenerated Trace Cache

• Goal
  • Reduce fetch bandwidth
  • Lower power – save decode time
  • Minimal redundancy ➔ area, power

• Challenges
  • Simpler prediction structure
  • Fast address translation
  • Lower switch latency
Micro-Operation Cache (UC) +fusion

IC-to-UC mapping example (4 Uops in UC line, 16 bytes IC line)

IC line (FF00):  
Offset: 0 1 2 3 4 5 6 7 8 9 A B C D E F

I1 I2 I3 I4 - JE RET

Uops:

UC lines (same set):
UCL1 - way 0
Offset: 3
Byte length: 5

UCL2 - way 7
Offset: 8
Byte length: 3

UCL3 - way 4
Offset: B
Byte length: 1

BB_1 : I1, ..., I4
BB_2:  I5

ICache to UCache mapping example

Front-end pipeline with Uop cache

ICache to UCache mapping example
Fetch bandwidth example

Control flow graph
A, B, C are instruction blocks

Dynamic instruction stream

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>...</th>
<th>...</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

time
Trace Cache Concept

• A different “Instruction” cache contains the dynamic stream of the executed instructions

 => Trace cache acts as “branch predictor” + wide instructions supplier
Trace Cache Overview

I Cache

Decoder

Trace Cache

Fill Unit

address

Stream Mode

Build Mode

To Execution Core
Trace Cache Overview

- **L2**
- **I Cache**
- **Decoder**
- **Branch Predictor**
- **Trace Cache**
- **Fill Unit**
- **IP**
- **selection logic**

To Execution Core
Trace cache line

- **Tag**: identifies starting address of trace
- **N instructions** (potentially decoded)
- **Next address**: next fetch address
- **path info**: branch flags (T, NT), number of branches, trace ends w/ branch?,...

<table>
<thead>
<tr>
<th>PC</th>
<th>Tag</th>
<th>N Instructions</th>
<th>Next address</th>
<th>path info?</th>
</tr>
</thead>
</table>

Uri Weiser/VLSI_06_2011.ppt/Nov 2014
Trace cache issues

- **Trace cache hit rate**
  - Cost of miss in the TC = moving from “stream node to build mode”
  - Do we force lines’ unique prefix (same address)?

- **Trace cache inefficiency ➔ power, area**
  - Trace redundancy
    - Multiple entry Trace Cache
    - Multiple exit Trace cache ....
  - Decoded information
  - Tuning
  - ...

Trace cache example
The Intel Pentium® 4 processor

• Single entry, multiple exits
• Unique prefix (no two traces can start at same address)
• Each trace may occupy several lines
• 6 micro-operations in a line (single fetch)
• Works in ½ frequency
• Requires a special branch predictor for the trace cache
• Several practical limitations
  ➢ Number of branches in line
  ➢ Several restrictions on micro-operations mix in line
• Switching from “stream” to “build” is costly
Trace Cache: Conclusions

• New approach to improve Instruction Fetch bandwidth
  • Potential performance improvement

• Implications
  • impact on die area, power!
  • Major performance impact when miss in Trace Cache

• Future potential enhancements
  • Optimized traces
  • Filter caches
Block-Based Trace Cache

• Can be viewed as a 2 level trace cache
• Lines of sequential instruction block are stored
• Traces are stored as sequence of pointers to blocks
  + Less redundancy than trace cache
  - Indirect access – increased latency
  - Simultaneous access to several blocks requires multiporting
• Overall benefit is not a given
Block Based Trace Cache

Figure 1 - The conventional trace cache.

Figure 2 - The block-based trace cache.
Extended Block Cache

- Consists of block connected with unconditional jumps only
- Blocks are indexed by end instruction!
- Allow merging of suffix blocks
- Allow extending traces dynamically
- Requires branch predictor to locate block start
Block Based Trace Cache

Figure 5. XBC Structure: the Whole Picture.
Instruction Reuse

Instruction Re-execution with the same input data will yield the same results (output data)

- If input data (A) for a repeated instruction "BLOCK" (E) is identical to the previous occurrence - replace (collapse) the execution BLOCK with the result="output" (O) of the block
- This is NOT a speculative process!
Self Modified Code

- The concept
  Architecture based uArchitecture

- The issues

- The solutions