MLP yes!
ILP no!

Work on memory level parallelism.
Stop worrying about IPC.

Andy “Krazy” Glew
glew@cs.wisc.edu, glew@hf.intel.com

Thought Experiment

Assume:
- Memory latency (cache miss latency) = 1000x ALU compute latency
- Memory bandwidth easy to obtain

Definitions

MLP
= Memory Level Parallelism
= Number cache misses simultaneously outstanding
esp. for linked lists!

ILP = Instruction Level Parallelism
IPC metric misleading (Inst. per Clock)

Why IPC is misleading
MLP ≠ IPC

- MLP ≈ 4 cache misses outstanding
- IPC = 4 / 1004 ≈ 0.004
- Narrow machine ≈ Wide superscalar
  - e.g. AXPY (trivial MLP)
**Mindset**

**Microarchitecture Impact**

- CPU idle waiting for memory
- Low IPC <1
- Present OOO CPUs
  - I-cache miss critical path
  - D-cache miss non-critical
- Branch Prediction
- Data Value Prediction
- Prefetching
- Multithreading

**Main Points**

- Need changed *mindset* to seek MLP optimizations
- IPC a bad metric for ILP

**Processors with low IPC can have high MLP**
- deep instruction windows
- highly non-blocking caches
- MLP enablers
  - implicit multithreading
  - hardware skiplists
  - large microarchitecture data structures
    - e.g. main memory compression

**Pointer Chasing is critical**

Large linked data structures
- e.g. 3D graphics “world”
- memory in size
- >> cache
- Brute force doesn’t help
  - increasing window
  - increasing frequency

**Solution**

HW Skip Lists

# skips \(\propto\) memory
⇒ store in main memory w. compression

**MLP processor sketch**
Conclusion

Inevitable forward march of ILP will continue
- the MT generations
  - explicit
  - implicit
- the MLP generation

No shortage of ideas to make uniprocessors faster.

Backup Slides

- Slides not included in the short (8 minute) presentation,
- answer likely questions
- useful if slides photocopied

Processors with low IPC can have high MLP

Examples

- Simplifying assumptions
  - AXPY:
    - simplest case unoptimized, other cases the usual: SW pipelining, accumulators, etc.
  - Linked Data Structures
    - randomized list / tree layout
    - independent visit functions

<table>
<thead>
<tr>
<th>EUs</th>
<th>Misses</th>
<th>Uarch</th>
<th>TimeN</th>
<th>Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>InO</td>
<td>N*1000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>OOO</td>
<td>N/64*1000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>OOO</td>
<td>N/64<em>1000 + 64</em>4</td>
<td></td>
</tr>
</tbody>
</table>
Tree Traversal

\[ V(p) \{ p \rightarrow l \&\& V(p \rightarrow l); \ p \rightarrow r \&\& V(p \rightarrow r); \ VV(p) ; \} \]

<table>
<thead>
<tr>
<th>EUs</th>
<th>Misses</th>
<th>Uarch</th>
<th>Time ( N )</th>
<th>Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>InO</td>
<td>( N \times (1000 + V) )</td>
<td>—— ——</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>OOO</td>
<td>( N \times 1000 )</td>
<td>—— ——</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>OOO</td>
<td>( N \times 1000 + 64 \times 4 )</td>
<td>—— ——</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>OOO + skiplists</td>
<td>( N/m \times 1000 + 64 \times 4 )</td>
<td>—— ——</td>
</tr>
</tbody>
</table>

Tree Algorithms

- Combinatoric explosion N-ary trees
- Traversal order skiplists \( \approx \) threading
  - works if similar traversals repeated
- Searches
  - key equal searches \( \rightarrow \) hash tables
  - proximity and range searches \( \rightarrow \) traversals

Caveats - Do I believe this s**t?

- Is there a memory wall? **YES**
- Do I care about branch prediction? **YES, but in the MLP world it is a secondary effect.**
- Do I care about IPC? **In inner loops. Not when latency cache missing.**
- Aren’t circuits getting slower? **NO. Wires are getting slower. Gates are still getting faster. Tricks make ALUs faster still.**
- Isn’t MLP a form of ILP? **YES. IPC is not a metric of ILP.**

Cache Hierarchies

- Multilevel cache hierarchies
  - latency = \( \sqrt{\text{size}} \)
  - reportedly of diminishing effectiveness
  - large working set applications? **(not SPEC95)**
Alternate MLP architecture

- SW skip lists
  - library data structures (STL)
  - or, compiler…
- Prefetch instructions
  - Eager prefetch of linked data structures
    ⇒ less speedup than hardware MLP
  - if tree nodes big
  - Traversal order threading + skip lists
    ⇒ same speedup as hardware MLP

Unused Slides

- The following slides are not used in the current MLP presentation, and contain new information. E.g. they are skipped just because of time.

Data structures + MLP

- Arrays: trivial; easy MLP
- Linear linked lists: skip lists work
- N-ary Trees
  - combinatoric explosion
  - traversals easier to parallelize than searches esp. proximity searches
- Hash tables
  - already minimally cache missing
  - conflicts: hash probing > chaining

Large Instruction Windows

- Brute Force
  - large windows spill to RAM
  - cache frequently used parts
- Expandable, Split, Instruction Windows
  - Sohi / Multiscalar

Forget / Recompute
- 2+ windows
- retirement (=OOO)
- non-blocking
  - Oldest instruction blocked ⇒ advance window marking result unknown.
  - Mispredict ⇒ set non-blocking window = retirement window.
Datascalar

- Limited MLP help
- E.g. 4-way datascalor (planar)
  \[ \Rightarrow M_{size}/4, M_{latency}/2 \text{ - faster} \]
  \[ \Rightarrow \text{Interconnect delay} = M_{latency}/2 \]
- Linked list, randomized:
  \[ \frac{1}{4} M_{latency}/2 + \frac{3}{4} M_{latency}/2 = M_{latency}/2 \]
- N-way:
  \[ \frac{1}{N} \sum_{i=1}^{N-1} \left( \frac{1}{N} + \frac{L_{interconn}}{M_{latency}} \right) \]

Processors with low IPC can have high MLP

Examples

Here:
- AXPY
- Linear Linked List

Backup:
- Tree Traversal
  ** 3D graphics
  large linked structures
  ≈ memory
  >> cache
  randomized

Processors with low IPC can have high MLP

AXPY

\[ A \ast X_i + Y_i \]

Wide Superscalar CPU  Narrow CPU
4 misses deep  6 misses deep

Many cache misses ⇒ Narrow - Wide = small startup cost
Why not spend HW on cache misses, not superscalar EUs?

<table>
<thead>
<tr>
<th>EUs</th>
<th>Misses</th>
<th>Uarch</th>
<th>Time_N</th>
<th>Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>InO</td>
<td>N*(1000+V)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>OOO</td>
<td>N*1000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>OOO</td>
<td>N<em>1000 + 64</em>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>OOO + skiplists_m</td>
<td>N/m<em>1000 + 64</em>4</td>
<td></td>
</tr>
</tbody>
</table>
MLP enablers

Skip Lists

- Convert list to tree
  ⇒ eager prefetch now helps ⇒ MLP=m
- Can be done by
  - software (library, compiler?)
  - hardware
    - store skip pointers in (compressed) main memory

Large μarch Data Structures

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW skip lists</td>
<td>Store in main memory</td>
</tr>
<tr>
<td>≥ 1 pointer per node</td>
<td>compress reserve</td>
</tr>
<tr>
<td>≪ memory size</td>
<td></td>
</tr>
<tr>
<td>&gt;&gt; any “cache”</td>
<td></td>
</tr>
<tr>
<td>Large Instruction Window</td>
<td>Spill to reserved RAM</td>
</tr>
<tr>
<td></td>
<td>Cache physical registers</td>
</tr>
<tr>
<td></td>
<td>Forget / recompute</td>
</tr>
</tbody>
</table>

MLP processor

super non-blocking

- Narrow EUs
  - wide inner loops?
- Deep instruction window
  - cached / spilled to RAM
  - forget / recompute
- Deeply non-blocking cache
- HW skiplists
  - stored in main memory with compression
- Smart Sequential Algorithms
  - Belady lookahead in instruction window
- Dynamic MT

More Backups

Non-obsolete slides added to end because of paper’s updateable qualities
Caveats - Do I believe this s**t?

Speed, Voltage, Power

- Brainiacs vs. Speed
  - déjà vu all over again
- Power
  - Speed $\propto$ Voltage
  - Power $\propto$ Voltage^2
  - Superscalar parallelism to save power may be a good thing ≠ performance

- Speed is fungible with superscalarness
  - 8GHz 1-way
  - 1 GHz 8-way
  - if circuit speeds trade off
  - sequential always easier than parallel

Workloads

- Unabashedly single-user
  - servers can use MT
- Q: next killer app.?
  - Is there one?
  - Does it fit in cache? Glew: no.
  - Probably something like 3D graphics virtual worlds.