Speculative Data Driven Multithreading

- **Goal:** Use spare thread contexts to target critical instructions
  - Branches that are likely to mispredict
  - Loads that are likely to miss
  - Many studies show that these are the instructions that count
  - Make the most of the work done by the helper thread (DDT)
Key Ideas

- DDT does not need to be fetched in program order
- Its instructions do not actually have to commit
- Using the DDT just for prefetching (as many other proposals do) wastes opportunities (especially for branches)
- Leverage the compiler/profiler to generate the threads.
Managing the Costs of DDTs

- DDTs consume fetch and execute bandwidth
  - Cost: Slows the main thread
  - Solution: Minimize the size of the DDTs
- DDTs consume physical registers
  - Cost: Can stall the main thread
  - Solution: “integrate” results of DDT into the main threads execution.
Creating DDTs

- DDTs are created by the compiler
- How can we predict when a DDT is “good”?  
  - A DDT is “good” if it allows execution of an instruction in the DDT far before it would execute in the control-driven thread.
Measuring DDT quality

- Data flow height -- The length of the longest dependence chain from the DDT’s inputs to an instruction
  - Shorter is better.
  - Ignores architectural limitations
- Fetch constraints
  - The processor can only fetch a few (2, they say) DDT instructions per cycle.
  - Use this constraint to “narrow” or “squeeze” the dataflow graph of the DDT
Measuring DDT quality

- The goal is to maximize DDT advantage
- Mostly, this means maximize DDT FCDH relative to main thread FCDH
  - Limit window to 1024 instructions
  - Try a variety of DDT lengths, see which is best.
Implementing DDT

- They implement DDT on an SMT-based system
  - Easy thread creation
  - Fast thread-to-thread communication
  - Minimizes the cost of the DDT (it can use “leftover” fetch bandwidth)
Integration

- Integration allows values created by one thread to be used by another.
  - Inter-thread bypassing.
  - The integration buffer is essentially a register file.
- The Integration table precisely identifies computed values
  - Input physical registers
  - Program counter
  - If these are the same, the result will always(?) be the same.
Integration

- If a match for an instruction is found in the integration table, the instruction can complete immediately.
  - Simply update the register map to label the register as the result of the instruction.
- Allows for some impressive tricks
  - Load times faster than a cache hit
  - Branches that resolve very early in the pipeline.
- Significant gain over previous helper thread techniques that just “warmed” BP and caches.
Integration

- Integration seems to add significant complexity
  - Previous work describes in it more detail, but it’s still not wholly convincing.
- The machine model they use corresponds to no real machine.
  - This is an unfortunate “feature” of the simulator they use.
- It’s hard to tell what the real impact is.
Evaluation

- Simulation-based study with SPEC2000 workloads + olden benchmarks (pointer-intensive).
- They present data to demonstrate the value of integration and their superiority to simple priority-based schemes
  - Integration helps branches a great deal
  - Prioritizing critical instructions doesn’t give them enough of an advantage.
Advantages of Integration

<table>
<thead>
<tr>
<th></th>
<th>Cache Misses</th>
<th>Branch Mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mcf</td>
<td>vpr</td>
</tr>
<tr>
<td><strong>Base</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>529.39</td>
<td>1304.95</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>12.43</td>
<td>3.86</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>24.57</td>
<td>47.48</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td>9.2</td>
<td>12.0</td>
</tr>
<tr>
<td><strong>Base + DDMT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>574.96</td>
<td>1400.95</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>7.45</td>
<td>3.36</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>19.44</td>
<td>39.02</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td>9.2</td>
<td>12.0</td>
</tr>
<tr>
<td><strong>Base + critical scheduling</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>529.42</td>
<td>1305.15</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>12.41</td>
<td>3.80</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>24.64</td>
<td>39.16</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td>0.0</td>
<td>-0.1</td>
</tr>
<tr>
<td><strong>Base + DDMT - Integration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>605.93</td>
<td>1406.29</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>9.45</td>
<td>3.65</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>22.56</td>
<td>41.29</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td>1.7</td>
<td>8.5</td>
</tr>
</tbody>
</table>

*TABLE 5. Performance contributions of data-driven sequencing and integration.*
Other efforts

- Many, many approaches have been tried for “helper” threads
  - One thread executes or partially executes the program to precompute values or warm on-chip structures
  - In some cases, the instructions are intermingled in a single thread.
  - Variation in the complexity of hardware required and the division between hw and sw.