Lecture 19c: Decoupled Access-Execute

Prof. Onur Mutlu

ETH Zürich
Spring 2021
7 May 2021
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Fine-Grained Multithreading
- Out-of-order Execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- SIMD Processing (Vector and array processors, GPUs)
Decoupled Access/Execute (DAE)
Decoupled Access/Execute (DAE)

- **Motivation:** Tomasulo’s algorithm too complex to implement
  - 1980s before Pentium Pro

- **Idea:** Decouple operand access and execution via two separate instruction streams that communicate via ISA-visible queues.

Decoupled Access/Execute (II)

- Compiler generates two instruction streams (A and E)
  - Synchronizes the two upon control flow instructions (using branch queues)

q = 0.0
Do 1 k = 1, 400
1 x(k) = q + y(k) * (r * z(k+10) + t * z(k+11))

Fig. 2a. Lawrence Livermore Loop 1 (HYDRO EXCERPT)

<table>
<thead>
<tr>
<th>Access</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7 + -400</td>
<td>. negative loop count</td>
</tr>
<tr>
<td>A2 + 0</td>
<td>. initialize index</td>
</tr>
<tr>
<td>A3 + 1</td>
<td>. index increment</td>
</tr>
<tr>
<td>X2 + r</td>
<td>. load loop invariants</td>
</tr>
<tr>
<td>X5 + t</td>
<td>. into registers</td>
</tr>
<tr>
<td>loop:</td>
<td>X7 + z + 11, A2</td>
</tr>
<tr>
<td>X4 + X2 *f X3</td>
<td>. r*z(k+10)-flt. mult.</td>
</tr>
<tr>
<td>X3 + X5 *f X7</td>
<td>. t * z(k+11)</td>
</tr>
<tr>
<td>X7 + y, A2</td>
<td>. load y(k)</td>
</tr>
<tr>
<td>X6 + X3 +f X4</td>
<td>. r<em>z(x+10)+t</em>z(k+11))</td>
</tr>
<tr>
<td>X4 + X7 *f X6</td>
<td>. y(k) * (above)</td>
</tr>
<tr>
<td>A7 + A7 + 1</td>
<td>. increment loop counter</td>
</tr>
<tr>
<td>x, A2 + X4</td>
<td>. store into x(k)</td>
</tr>
<tr>
<td>A2 + A2 + A3</td>
<td>. increment index</td>
</tr>
<tr>
<td>JAM loop</td>
<td>. Branch if A7 &lt; 0</td>
</tr>
</tbody>
</table>

AEQ + z + 10, A2  X4 + X2 *f AEQ
AEQ + z + 11, A2  X3 + X5 *f AEQ
AEQ + y, A2      X6 + X3 +f X4
A7 + A7 + 1      X, A2 + EAQ
A2 + A2+ A3      A2 + A2+ A3

Fig. 2b. Compilation onto CRAY-1-like architecture

Fig. 2c. Access and execute programs for straight-line section of loop
Decoupled Access/Execute (III)

- **Advantages:**
  + Execute stream can run ahead of the access stream and vice versa
  + If A is waiting for memory, E can perform useful work
  + If A hits in cache, it supplies data to lagging E
  + Queues reduce the number of required registers
  + Limited out-of-order execution without wakeup/select complexity

- **Disadvantages:**
  -- **Compiler support** to partition the program and manage queues
  -- Determines the amount of decoupling
  -- Branch instructions require synchronization between A and E
  -- Multiple instruction streams (can be done with a single one, though)
Astronautics ZS-1

- Single stream steered into A and X pipelines
- Each pipeline in-order


Loop Unrolling to Eliminate Branches

for (int i = 0; i < N; i++){
}

for (int i = 0; i < N; i+=4){
    A[i+1] = A[i+1] + B[i+1];
}

- **Idea:** Replicate loop body multiple times within an iteration

+ Reduces loop maintenance overhead
  - Induction variable increment or loop condition test

+ Enlarges basic block (and analysis scope)
  - Enables code optimization and scheduling opportunities

-- What if iteration count not a multiple of unroll factor? (need extra code to detect this)

-- Increases code size
A Modern DAE Example: Pentium 4

Intel Pentium 4 Simplified

Mutlu+，“Runahead Execution,”
HPCA 2003.
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Fine-Grained Multithreading
- Out-of-order Execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- **SIMD Processing (Vector and array processors, GPUs)**