Instruction Level Parallelism

Pipeline with data forwarding and accelerated branch

Loop Unrolling

Multiple Issue -- Multiple functional Units

Static vs Dynamic Optimizations
Optimization Example
C-code

```c
k = len;
do {
    k--;
} while(k > 0)
```
Register Usage

x19    len
x20    base address of A
x21    x

x10    address of A[k]
x9     value of A[k] (old and new)
Basic loop using pointer hopping

```
  lsl  x10, x19, 3
  add  x10, x10, x20

loop:

  sub  x10, x10, 8
  ldur x9, [x10, #0]
  add  x9, x9, x21
  stur x9, [x10, #0]
  sub  x11, x10, x20
  cbnz x11, loop

xxx
```
Time for 1000 Iterations for Single- and Multi-cycle

• Single Cycle
  – Every instruction takes 800 picoseconds (ps)
  – Time = \(6 \times 800 \times 1000 + 2 \times 800 = 4,801,600 \text{ ps} = 4801.6 \text{ nanoseconds (ns)}\)

• Multicycle
  – 200 ps per cycle, variable number of CPI
  – Cycles = \((1 \times 3 + 4 \times 4 + 1 \times 5) \times 1000 + 2 \times 4 = 24,008\)
  – Time = \(24,008 \times 200 \text{ ps} = 4,801.6 \text{ ns}\)
Pipeline, nops for data hazards

```assembly
lsl   x10, x19, 3
add   x10, x10, x20

loop:

sub   x10, x10, 8
ldur  x9, [x10, #0]
nop
add   x9, x9, x21
stur  x9, [x10, #0]
sub   x11, x10, x20
nop

cbnz  x11, loop

stall if taken
```
Time for simple pipeline

- 200 ps per cycle, 1 CPI (including nops)

- Time = 9x200x1000 + 2x200 − 1x200 ps = 1,800.2 ns (no stall last iteration)
Reordering code

lsl      x10, x19, 3
add      x10, x10, x20

loop:

sub       x10, x10, 8
ldur      x9, [x10, #0]
sub       x11, x10, x20
add       x9, x9, x21
stur      x9, [x10, #0]
cbnz       x11, loop
stall if taken
Time for pipeline with reordered code

- 200 ps per cycle, 1 CPI (including nops)

- Time = $7 \times 200 \times 1000 + 2 \times 200 - 1 \times 200$ ps = 1,400.2 ns (no stall last iteration)

- with branch prediction, no stall, 1,200.4 ns
Loop unrolling – step 1

ls1 X10, X19, 3
add X10, X10, X20

loop:
sub X10, X10, 8
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]

cbnz X11, loop
stall if taken
sub X10, X10, 8
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]
cbnz X11, loop
stall if taken
sub X10, X10, 8
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]
cbnz X11, loop
stall if taken
sub X10, X10, 8
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]
cbnz X11, loop
stall if taken
sub X10, X10, 8
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]
cbnz X11, loop
stall if taken
sub X10, X10, 8
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]
cbnz X11, loop
stall if taken
Loop unrolling – step 2
remove extra loop control

```
lsr X10, X19, 3
add X10, X10, X20

loop:
sub X10, X10, 32
ldur X9, [X10, #24]

nop
add X9, X9, X21
stur X9, [X10, #24]
ldur X9, [X10, #16]

nop
add X9, X9, X21
stur X9, [X10, #16]
ldur X9, [X10, #8]

nop
add X9, X9, X21
```

```
stur X9, [X10, #8]
add X10, X10, X20
ldur X9, [X10, #0]
sub X11, X10, X20
add X9, X9, X21
stur X9, [X10, #0]
```

cbnz X11, loop

```
stall if taken
```

Loop unrolling – step 2
remove data hazards with reorder, register renaming

```assembly
lsr X10, X19, 3
add X10, X10, X20

loop:
sub X10, X10, 32
ldur X9, [X10, #24]
ldur X12, [X10, #16]
add X9, X9, X21
stur X9, [X10, #24]
add X12, X12, X21
stur X12, [X10, #16]
ldur X9, [X10, #8]
ldur X12, [X10, #0]
add X9, X9, X21
stur X9, [X10, #8]
sub X11, X10, X20
add X12, X12, X21
stur X12, [X10, #0]
cbnz X11, loop
stall if taken
```
Time for pipeline with loop unrolling

• 200 ps per cycle, 1 CPI (including nops)
• 4 iterations per loop means 250 times in loop
• Time = 16x200x250 + 2x200 -1x200 ps = 800.2 ns  
  (no stall last iteration)
• with branch prediction 750.4 ns
Dual Pipeline

Upper path for ALU ops, Lower path for lw/sw
Dual Pipeline Requirements

- Two instruction (64 bits) IR
- Two additional read ports in register file
- An additional write port in register file
- Appropriate forwarding hardware

- Compiler must insure no conflicts between the two simultaneous instructions

- Instruction pairing and order can be done statically by compiler
Dual Pipeline

- Two instruction pipe
  - one for arithmetic or branch
  - one for load or store
- Instructions can be issued at same time
  - if no data dependencies
  - following instructions follow same hazard rules
- Loop unrolling for more overlap
- Register renaming to avoid data dependency
Dual Pipeline, pairings

```
lsr X10, X19, 3
add X10, X10, X20

loop:
  sub X10, X10, 8  ldur X9, [X10, #-8]
  sub X11, X10, X20  nop
  add X9, X9, X21  nop
  cbnz X11, loop  stur X9, [X10, #0]
  stall if taken
```
Time for dual pipeline (no loop unrolling)

- 200 ps per cycle, 1 or 2 instr per cycle

- Time = $5 \times 200 \times 1000 + 2 \times 200 - 1 \times 200$ ps = 1,000.2 ns

- 800.4 ns with branch prediction
Loop unrolling – step 2
remove data hazards with reorder, register renaming

```assembly
ls1  X10, X19, 3
add  X10, X10, X20

loop:
sub  X10, X10, 32
ldur X9, [X10, #24]
ldur X12, [X10, #16]
add  X9, X9, X21
stur X9, [X10, #24]
add  X12, X12, X21
stur X12, [X10, #16]
add  X12, X12, X21
stur X12, [X10, #16]
ldur X9, [X10, #8]
ldur X12, [X10, #0]
add  X9, X9, X21
stur X9, [X10, #8]
add  X12, X12, X21
stur X12, [X10, #0]
sub  X11, X10, X20
add  X12, X12, X21
sub  X11, X10, X20
add  X12, X12, X21
add  X12, X12, X21
stur X12, [X10, #0]
cbnz X11, loop
stall if taken
```
Dual Pipeline, Loop unrolling
more register renaming

```
lsr  X10, X19, 3  
add  X10, X10, X20

loop:

sub  X10, X10, 32  
ldur X9, [X10, #24]  
ldur X12, [X10, #16]  
add  X9, X9, X21  
stur X9, [X10, #24]  
add  X12, X12, X21  
stur X12, [X10, #16]  
ldur X13, [X10, #8]  
ldur X14, [X10, #0]  
add  X13, X13, X21  
stur X13, [X10, #8]  
sub  X11, X10, X20  
add  X14, X14, X21  
stur X14, [X10, #0]  
cbnz X11, loop  
```

stall if taken
step 2, reorder/pair instructions

\begin{verbatim}
ls1    X10, X19, 3
add    X10, X10, X20

loop:
sub    X10, X10, 32
ldur   X9, [X10, #-8]
ldur   X12, [X10, #16]
add    X9, X9, X21
ldur   X13, [X10, #8]
add    X12, X12, X21
stur   X9, [X10, #24]
add    X13, X13, X21
ldur   X14, [X10, #0]
sub    X11, X10, X20
stur   X12, [X10, #16]
add    X14, X14, X21
stur   X13, [X10, #8]
cbnz   X11, loop
stall if taken
\end{verbatim}
Time for dual pipeline

• 200 ps per cycle, 1 or 2 instr per cycle
• 4 iterations per loop, 250 times through loop

\[
\text{Time} = 9 \times 200 \times 250 + 2 \times 200 - 1 \times 200 \text{ ps} = 450.2 \text{ ns}
\]
• 400 ns with branch prediction
• 10 times faster than single cycle or multi-cycle
• 3 times faster than simple pipeline
• 1 \(\frac{3}{4}\) times faster than single pipeline, loop unrolled
More Parallelism?

• Suppose loop has more operations
  – Multiplication (takes longer than adds)
  – Floating point (takes much longer than integer)

• More parallel pipelines for different operations – all the above techniques could result in better performance

• Static (as above) vs dynamic reordering

• Speculation

• Out-of-order execution
Multiple Issue – Multiple Functional Units

Instruction fetch and decode unit

Reservation station  Reservation station  ...  Reservation station  Reservation station

Functional units

Integer  Integer  ...  Floating point  Load-store

Out-of-order execute

Commit unit

In-order commit
Dynamic Scheduling

• Processor determines which instructions to issue
• Reservation unit use copies of operand values from registers (equivalent to renaming)
• When operation is completed, results are buffered in the commit unit
• When can be done in order, results are written to registers or memory
• Speculation
  – guessing which way a branch goes
  – Guessing that a load does not conflict with a store
The extensive queues allow up to 106 RISC operations to be outstanding, including 24 integer operations, 36 floating point/SSE operations, and 44 loads and stores. The load and store units are actually separated into two parts, with the first part handling address calculation in the Integer ALU units and the second part responsible for the actual memory reference. There is an extensive bypass network among the functional units; since the pipeline is dynamic rather than static, bypassing is done by tagging results and tracking source operands, so as to allow a match when a result is produced for an instruction in one of the queues that needs the result. Copyright © 2009 Elsevier, Inc. All rights reserved.
FIGURE 4.75 The Opteron X4 pipeline showing the pipeline flow for a typical instruction and the number of clock cycles for the major steps in the 12-stage pipeline for integer RISC-operations. The floating point execution queue is 17 stages long. The major buffers where RISC-operations wait are also shown. Copyright © 2009 Elsevier, Inc. All rights reserved.