Example

for(int i=0; i<1000; i++)
    a[i] = b[i] + c;

Loop:
LDR R5, R2, #0 ; element of b
ADD R6, R3, R5 ; make next a
R1 – base addr of a
R2 – base addr of b
R3 – contains value of c
R4 contains value 1000
SUB R4, R4, #1 ; decr loop var
BRp R4, Loop

Assume only load delay first

Rescheduling

Loop:
LDR R5, R2, #0
ADD R6, R3, R5
ADD R1, R1, #1
ADD R2, R2, #1
SUB R4, R4, #1
BRp R4, Loop
Loop Unrolling

- We still have the control hazard when we hit the branch.
- In this case 1000 times. We can fix this by resheduling and unrolling the loop.

```c
for(int i=0;i<1000;i = i + 2) {
    a[i] = b[i] + c;
    a[i+1] = b[i+1]; c;
}
```

Unrolling Loop

Loop:
- LDR R5, R2, #0 ; element of b
- ADD R6, R3, R5 ; make next a
- LDR R0, R2, #1 ; next element of b
- ADD R7, R3, R0 ; make next a + 1
- STR R6, R1, #0 ;
- STR R7, R1, #1 ;
- ADD R1, R1, #2 ;
- ADD R2, R2, #2 ;
- SUB R4, R4, #2 ; decrement loop var
- BRp R4, Loop

Performance Improvement

- CPI_{loop} with rescheduling
  - 13 cycles per loop
  - 3 cycles for branch penalty (assume no prediction & no useful instructions to fill slot)
  - 7 total instructions per loop
  - CPI = (13 + 3)/7 = 2.2

- CPI_{loop} with rescheduling & loop unrolling
  - 16 cycles per loop
  - 3 cycles for branch
  - 10 total instructions per loop
  - CPI = 19/10 = 1.9
  - Note: that branches occur half as many times as the original
Limitations of Loop Unrolling

- Need Lots of Registers
  - To hold sums/products
  - When not enough registers, must spill temporaries onto stack
    - Wipes out any performance gains

- Adds to instruction count
  - Problem if memory is limited. E.g. Embedded System
  - Can add to cache performance

- By default GCC compiler does not enable loop unrolling when code is compiled

Loop Unrolling gcc options

- funroll-loops
  - Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop.

- funroll-all-loops
  - Unroll all loops, even if their number of iterations is uncertain when the loop is entered. This usually makes programs run more slowly.

Other Compiler Optimization Techniques

- Loop Fusion: one loop out of two or more neighboring loops
  - Advantage: reduce loop overhead

- Loop Fission: to break a loop into multiple loops over the same index range but each taking only a part of the loop's body
  - Advantage: to achieve better utilization of locality of reference
    - Phenomenon of the same value or related storage locations being frequently accessed

Fission vs. Fusion Example

Fission

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
   a[i] = 1;
}
for (i = 0; i < 100; i++) {
   b[i] = 2;
}
```

Fusion

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
   a[i] = 1;
   b[i] = 2;
}
```
Getting CPI < 1: Issuing Multiple Instructions (Ops)/Cycle

- **Vector Processing**
  - Explicit coding of independent loops as operations on large vectors of numbers. E.g. Cray machines
  - Multimedia instructions being added to many processors e.g. MMX, SSE

- **(Very) Long Instruction Words (V)LIW:**
  - Fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates
  - Intel Architecture-64 (IA-64) 64-bit address

- **Superscalar**
  - varying no. instructions/cycle (1 to 8),
  - Multiple functional units
  - scheduled by compiler(static) or by HW (dynamic)
  - E.g. IBM PowerPC, DEC Alpha, Pentium 4

- **Chip Multiprocessors**
  - Simple individual processors and high clock rate
  - 2 or more cores per chip
  - E.g. Intel Core 2 Duo, IBM PS3

- **Anticipated success of multiple instructions lead to Instructions Per Clock cycle (IPC) vs. CPI**

---

**Superscalar Schematic Example**

- Issue packet: group of instructions from fetch unit that could potentially issue in 1 clock
  - If instruction causes structural hazard/data hazard either due to earlier instruction in execution or to earlier instruction in issue packet, then instruction should not issue
  - 0 to N instruction issues per clock cycle, for N-issue

- Performing issue checks in 1 cycle could increase clock cycle time: O(N²) comparisons
  - => issue stage usually split and pipelined
  - 1st stage decides how many instructions from within this packet can issue(available functional units), 2nd stage examines hazards among selected
  - instructions and those already been issued

- Higher branch penalties => prediction accuracy important
Multi-Issue Architecture: Fetch Stages
- Must fetch multiple instructions per cycle from cache memory to keep a steady feed of instructions going to the other stages
- Reduce misaligned by diving memory into banks
- Conditional branch instructions are usually predicted in advance to ensure uninterrupted stream of instructions
- Speculative Execution

Multi-Issue Architecture: Issue Stages
- Instructions are decoded and examined for control and data dependencies
  - RAW, WAW and WAR
  - WAW and WAR are handled by Register Renaming
- Instructions are dispatched to buffers associated with hardware functional units for later issuing and execution
- Instructions can be dispatched in or out of order
  - Out of order for increasing ILP
  - Requires re-ordering to maintain program logic

Issue Stage: Data Dependencies & Register Renaming
- ADD R3, R7, R0 RAW
- LDRWAW R2, R3, #0
- ADD R3, R3, #4 WAR

Register Renaming
- A technique used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations
- Use a table to map architectural registers and physical registers
  - Physical registers are assigned from a “free list”
  - Ports required: operands per instruction x no. of decoded instructions

Issue Stage: Reservation Stations
- Instruction issuing is defined as the run-time checking for availability of data and resources
- Reservation stations hold information about source operands for an operation
- When all operands are present, the instruction may issue
- They may be partitioned according to instruction type or pooled into a single large block
Multi-Issue Architecture: Reorder & Commit

- The state of the execution is noted in the reorder buffer
  - As instructions are executed out of program order
- Purpose of the commit phase is to maintain the illusion of a sequential execution model
  - When an instruction is "committed", its result is allowed to modify the logical state of the machine
  - E.g. The state of the machine is saved in a history buffer
    - Instruction update the state of the machine as they execute and when there is a problem, the state of the machine can be recovered from the history buffer.
    - Commit phase gets rid of the history state that's no longer needed

Chip Multiprocessors

- Advances in the field of integrated chip processing
  - Gate density
    - More transistors per chip
- An integrated circuit with two or more independent processors
  - Also known as multi-core processors
- Cores may be coupled together tightly or loosely
  - May or may not share caches
  - May implement message passing or shared memory inter-core communication methods

Shared memory architectures

- All processors access all memory as global address space
- Changes made by one processor are visible by other processors
- Two types based on the differences in memory access speed:
  - Uniform memory access (UMA)
  - Non-uniform memory access (NUMA)

Shared Memory Architecture: UMA vs. NUMA

UMA

- Access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data
- Bus Based
  - Also becomes bottleneck

NUMA

- Identical processors, processors have different time for accessing different part of the memory
  - A processor can access its own local memory faster than non-local memory
- Network Based
Cache Coherence

- Cache coherence
  - There are multiple versions of data (memory copy, and cache copies).
  - How to maintain a consistent system view?

- Need some mechanism to make the memory system appear coherent.
  - Mechanisms
    - Directory based
    - Snooping
  - Cache coherence protocols
    - E.g. MSI Protocol: Possible States a cache line can be in is Modified, Shared or Invalid

Shared Memory Architecture

- Advantages
  - Globally shared memory provides user-friendly programming perspective to programmers

- Disadvantage
  - Lack of scalability
    - No hope for UMA
    - What about NUMA
      - A lot of small traffic through the interconnect
      - adding processors changes the traffic requirement of the interconnect.
  - Writing correct shared memory parallel programs is not straightforward
    - E.g. Intel’s TBB (Open MP)

Distributed Memory Architectures

- Processors have their own local memory.
- Memory addresses in one processor do not map to another processor
- No concept of global address space
- No concept of cache coherency.
- To access data in another processor, use explicit communication

Distributed Memory Architectures contd..

- The networks can be very different for distributed memory architectures:
  - Massively parallel processors (MPP) usually use a specially designed network (and node).
    - IBM Bluegene, IBM SP series
  - Clusters usually use commodity system/local area networks: Infiniband, Quadrics, Myrinet, 10 Gbps Ethernet
    - Ranger (NO. 2 top supercomputer) at TACC(Texas Advanced Computing center) uses Infiniband
  - Grid computers use the Internet as the networks.
Distributed Memory Architectures contd..

- MPP, clusters and grid computers targets different types of applications
  - MPP and clusters support tightly coupled applications (large amount of interactions among processes)
    - Communicate every 1 microsecond
  - Grid computers can only support coarse-grain parallel applications or embarrassingly parallel applications
    - Communicate every second

Advantages:
- Memory is scalable with number of processors.
  - Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory
  - Without interference and without the overhead incurred with trying to maintain cache coherency
- Cost effectiveness: can use commodity, off-the-shelf processors and networking

Disadvantages:
- The programmer is responsible for the details associated with data communication
  - It may be difficult to map existing data structures, based on global memory, to this memory organization
  - E.g. MPI

CIT 595 29

Distributed Memory Architectures contd..

CIT 595 30