2110412 Parallel Computer Architecture: Multi-core Tech.

4: Advance ILP, Multiprocessor

Krerk Piromsopa, Ph.D.
Department of Computer Engineering
Chulalongkorn University

Exposing and Exploiting ILP

- Basic Scheduling and Loop Unrolling
- Software Pipelining: Symbolic Loop Unrolling
- Predicated Instructions

Loop Unroll

C Code:

```
For (i=1000; i>0; i=i-1)
  x[i] = x[i] +s;
```

The straightforward MIPS code (without scheduling)

```
Loop:
  L.D F0, 0(R1) ; F0=array element
  ADD.D F4, F0, F2 ; add scalar in F2
  S.D F4, 0(R1) ; store result
  DADDUI R1, R1, #-8 ; decrement pointer 8 bytes (per DW)
  BNE R1, R2, Loop ; branch R1!=R2
```

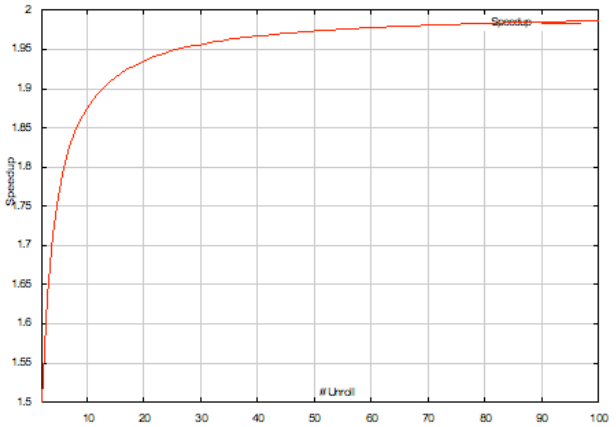

	Original Loop	Unroll	ReSchedule
1	L.D F0,0(R1)	L.D F0,0(R1)	L.D F0,0(R1)
2	stall	stall	L.D F3,8(R1)
3	ADD.D F4,F0,F2	ADD.D F1,F0,F2	ADD.D F1,F0,F2
4	stall	stall	ADD.D F4,F3,F2
5	stall	stall	DADDUI R1,R1,#-16
6	S.D F4,0(R1)	S.D F1,0(R1)	S.D F1,16(R1)
7	DADDUI R1,R1,#-8	L.D F3,-8(R1)	BNE R1,R2, Loop
8	stall	stall	S.D F4,8(R1)
9	BNE R1,R2, Loop	ADD.D F4,F3,F2	
10	Stall	stall	
11		stall	
12		S.D F4,-8(R1)	
13		DADDUI R1,R1,#-16	
14		stall	
15		BNE R1,R2, Loop	
16		Stall	
17			
18			
19		Maximum # of times?	
20			
21			
22			

POP QUIZ:

```
Loop:
  L.D F0, 0(R1) ; F0=array element
  ADD.D F4, F0, F2 ; add scalar in F2
  S.D F4, 0(R1) ; store result
  DADDUI R1, R1, #-8 ; decrement pointer 8 bytes (per DW)
  BNE R1, R2, Loop ; branch R1!=R2
```

Assume the number of iterations is unknown, but large.

What is the actual maximum number of times the simple loop can be unrolled using the given MIPS code? What is the limiting resource? Show how to increase the number of times the loop may be unrolled by transforming the MIPS code to make less intensive use of the limiting resource. How much does this transformation improve performance?



Software Pipeline

- Increasing the possibility that the unrolled loop can be scheduled without stalls.
- Interleaves instructions from different iterations.

	Original Loop	Unroll	ReSchedule
1	L.D F0,0(R1)	L.D F0,0(R1)	L.D F0,0(R1)
2	stall	stall	L.D F3,8(R1)
3	ADD.D F4,F0,F2	ADD.D F1,F0,F2	ADD.D F1,F0,F2
4	stall	stall	ADD.D F4,F3,F2
5	stall	stall	DADDUI R1,R1,#-16
6	stall	stall	stall
7	stall	stall	stall
8	stall	stall	stall
9	S.D F4,0(R1)	S.D F1,0(R1)	S.D F1,16(R1)
10	DADDUI R1,R1,#-8	L.D F3,-8(R1)	BNE R1,R2, Loop
11	stall	stall	S.D F4,8(R1)
12	BNE R1,R2, Loop	ADD.D F4,F3,F2	
13	Stall	stall	
14		stall	
15		stall	
16		stall	
17		stall	
18		S.D F4,-8(R1)	
19		DADDUI R1,R1,#-16	
20		stall	
21		BNE R1,R2, Loop	
22		Stall	

```
L.D  F0,0(R1)
L.D  F3,8(R1)
ADD.D  F1,F0,F2
ADD.D  F4,F3,F2
S.D  F1,16(R1)
S.D  F4,8(R1)
L.D  F0,0(R1)
L.D  F3,8(R1)
ADD.D  F1,F0,F2
ADD.D  F4,F3,F2
S.D  F1,16(R1)
S.D  F4,8(R1)
ADD.D  F1,F0,F2
ADD.D  F4,F3,F2
```

```
    L.D      F0,0(R1)  ; load M[i]
    L.D      F3,-8(R1)  ; load M[i-1]
    ADD.D    F1,F0,F2   ; adds to M[i]
    ADD.D    F4,F3,F2   ; adds to M[i-1]
    L.D      F0,-16(R1); load M[i-2]
    L.D      F3,-24(R1); load M[i-3]
    DADDUI  R1,R1,#-40     ; Adjust the offset
    stall
Loop:
    S.D      F1,40(R1) ; stores into M[i]
    S.D      F4,32(R1) ; stores into M[i-1]
    ADD.D    F1,F0,F2   ; adds to M[i-2]
    ADD.D    F4,F3,F2   ; adds to M[i-3]
    DADDUI  R1,R1,#-16
    L.D      F0,24(R1) ; load M[i-4]
    L.D      F3,16(R1) ; load M[i-5]
    BNE     R1,R2, Loop
    S.D      F1,40(R1)
    S.D      F4,32(R1)
    ADD.D    F1,F0,F2
    ADD.D    F4,F3,F2
    stall
    stall
    stall
    stall
    S.D      F1,24(R1)
    S.D      F4,16(R1)
```


Predicate Instruction

- C Code
  - if (A==0) { S=T; }
- Assembly Code
  - BNEZ R1, L
  - ADDU R2, R3, R0
- With conditional instruction
  - CMOVZ R2, R3, R1

Predicate in action

First Slot	Second Slot

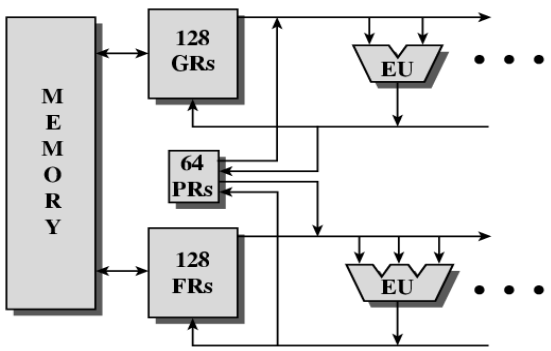
LW R1, 40(R2)
 	
BEQZ R10, L
LW R8, 0(R10)
LW R9, 0(R8)
	ADD R3, R4, R5
	ADD R6, R3, R7

First Slot	Second Slot

LW R1, 40(R2)
LWC R8, 0(R10), R10
BEQZ R10, L
LW R9, 0(R8)
	ADD R3, R4, R5
	ADD R6, R3, R7

IA64

- Intel & HP
- Explicit Parallel instruction Computing (EPIC)
  - Instruction level parallelism
  - VLIW
  - Branch predication
  - Speculative loading



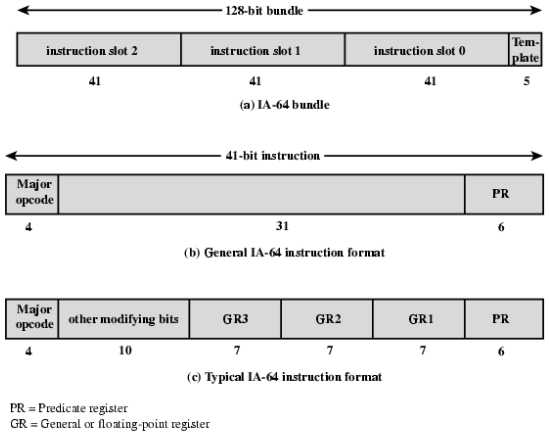
Execution Units

- I-Unit
  - Integer arithmetic
  - Shift and add
  - Logical
  - Compare
  - Integer multimedia ops
- M-Unit
  - Load and store
    - Between register and memory
  - Some integer ALU
- B-Unit
  - Branch instructions
- F-Unit
  - Floating point instructions

Comparison

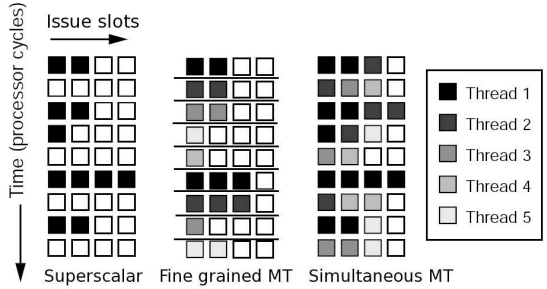
Superscalar	IA-64
RISC-line instructions, one per word	RISC-line instructions bundled into groups of three
Multiple parallel execution units	Multiple parallel execution units
Reorders and optimizes instruction stream at run time	Reorders and optimizes instruction stream at compile time
Branch prediction with speculative execution of one path	Speculative execution along both paths of a branch
Loads data from memory only when needed, and tries to find the data in the caches first	Speculatively loads data before its needed, and still tries to find data in the caches first

Instruction Format



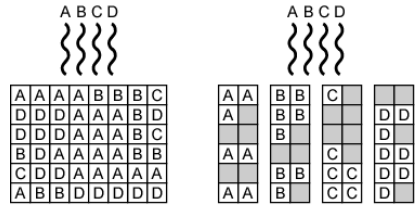
Thread-Level Parallelism

- No Thread
- Temporal
  - Coarse
  - Fine
- Simultaneous



Chip Multiprocessor

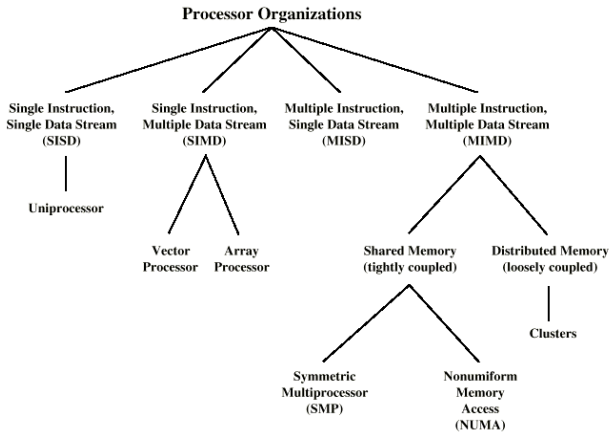
- Each processor is assigned thread



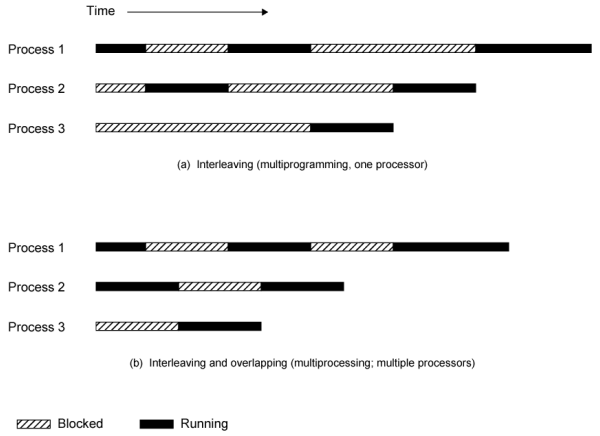
(j) simultaneous multithreading (SMT)

(k) chip multiprocessor

Taxonomy of Parallel Processor

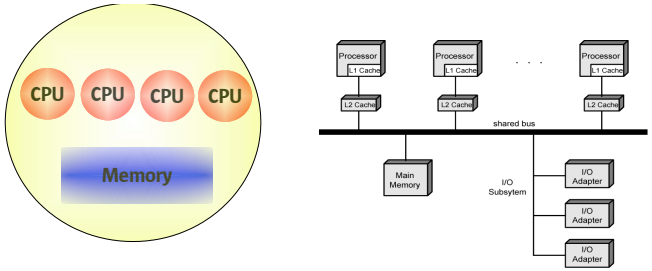


Multiprogramming and Multi Processing



SMP

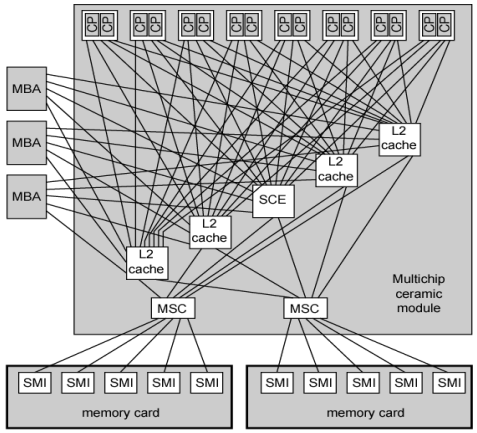
- Symmetric Shared-Memory Architectures (SMPs)
- Uniform Memory Access (UMA)
- Processors share memory



Advantage/Disadvantage

- Advantage
  - Simplicity, Flexibility, Reliability
- Disadvantage
  - Performance Limitation
  - Cache Coherency

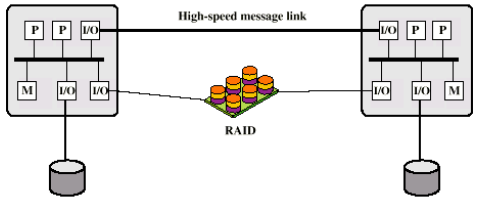
IBM z990



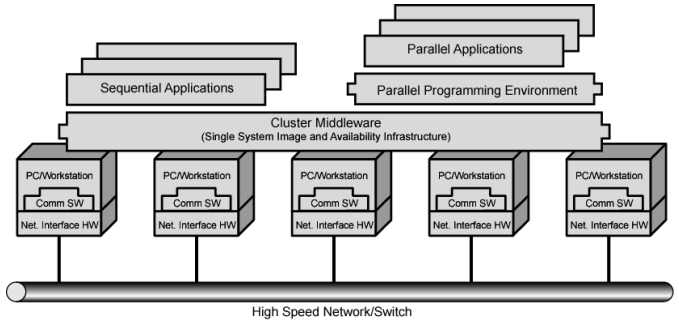
CP = central processor
MBA = memory bus adapter
MSC = main store control
SCE = system control element
SMI = synchronous memory interface

Cluster

- A group of interconnected computers (nodes) working together as a unified resource (one machine).
- Scalability, Availability, and Performance

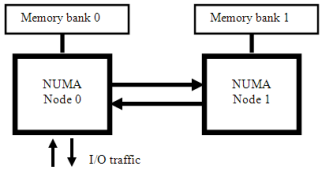


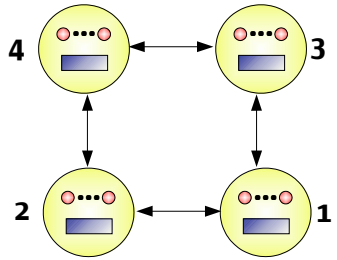
Cluster Architecture

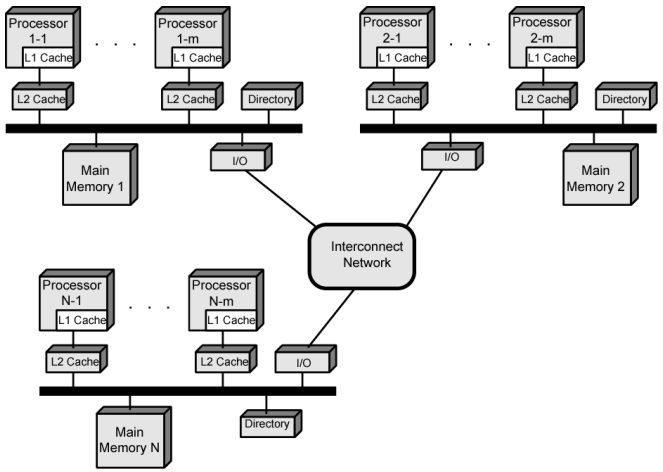


NUMA

- Non-Uniform memory access
- Access Times may vary







Comparison:

- SMP has limit to number of processors (16 - 64 processors)
- Each cluster has own memory
  - (Application do not see global memory)
- NUMA - Maintain large system wide memory with with scalability (1000+)

Exercise

From the following code fragment, assume that all data references are shown, assume that all values are defined before use, and that only b and c are used again after this segment. You may ignore any possible exceptions. The individual statements are numbered to provide an easy reference.

```
1. if (a>c) {
2.  d = d + 5;
3.  a = b + d +e;
} else {
4.  e = e +2;
5.  f = f +2;
6.  c = c +f;
}
7. b = a + f;
```

List the control dependences

The End
Questions?