HWHW #5

chapter 6

2019/1165

Sim Eun yeong

Introduction

Multiprocessor	: Computer system with at least 2 processors (↔ uniprocessor)
Parallelism	ⓐ Task level parallelism (process level parallelism / just parallel) : utilizing multiple processors by running independent program simultaneously (different task) ⓑ parallel processing program (CPP) (same task, different data) : single program that runs on multiple processors simultaneously
Cluster	: set of computers connected over local area that function as single large multiprocessor
Multicore microprocessor	: A microprocessor containing multiple processors (cores) in single integrated circuit
SMP (Shared Memory Multiprocessor)	: Shared memory multiprocessor = parallel processor with single shared memory

Shared Mem Model

Parallel Programming

- Parallel programming need to get Significant performance Improvement 대? 갈수록 성능향상 안되고 hw 비싸지니깐 easy!

Considerations

① data / task decomposition (model 어떻게 나눌건지)
② coordination (balancing node --- minimize idling)
③ Communication overhead (Shouldn't be layer than computation)

Amdahl's Law

: Sequential part can limit speed up

↓
rule that performance enhancement possible with given improvement is limited by amount that improved feature is used

① Execution time after improvement

$T_{new} = T(p)/N + T(s)$ per process

② Speed-up $= T(s) / \{(T(s) - T(p)) + T(p)/N\}$

$= 1/\{(1-\text{fraction}) + \text{fraction}/N\}$

- ex) Speed-up = 90 Improvement = 100 (N)

$90 = 1/\{(1-f) + f/100\}$

(Parallel) $f = 0.999$ fraction time affected

To achieve speed up from 90 → 100 processors, sequential percentage can only be 0.1%

$((1-f) + f/N)$

Parallel Programming (Cont.)

Scaling

Example[1]

$*$ assume scalar Addition can't be parallelized

- Sum of 10 scalars
- Sum of 10x10 matrix

↓ Scalability

Time to add two elements ↓

① Single processor → Time = $(10+100) * t_{add}$ $= 110 * t_{add}$

$*$ Speed up $= 1/\{(1-F_p) + F_p/N\}$

② 10 processors → $Time_{10} = 10 * t_{add} + (100/10) * t_{add}$

$5.5 = 1/\{(1-F_p) + F_p/10\}$ $= 20 * t_{add}$ (ideal : $10 * t_{add}$)

③ 100 processors → $Time_{100} = 10 * t_{add} + (100/100) * t_{add}$

↓

$10 = 1/(1-F_p) + F_p/100$ $= 11 * t_{add}$ (ideal : $1 * t_{add}$)

Example[2]

- Sum of matrix 100 x 100

$\frac{10010}{100}$ 100.1

1, 110, 100

① Single processor → $Time_1 = (10 + 10000) * t_{add}$

② 10 processors → $Time_{10} = 10 * t_{add} + 10000/10 * t_{add}$

$= 1010 * t_{add}$ (ideal : $1001 * t_{add}$)

③ 100 processors → $Time_{100} = 10 * t_{add} + 10000/100 * t_{add}$

$= 110 * t_{add}$ (ideal : $100.1 * t_{add}$)

→ Problem size fix 시키고 Process 개수 늘리는 것

✓ **Strong scaling** : problem size fixed (작업의 크기를 고정시키고 Multiprocessor 통해 얻는 성능 개선)

(프로세서 ∝ 작업크기)

✓ **Weak scaling** : problem size proportional to number of processors

즉, Constant Performance (processor 늘려도 작업도 같이 늘어나므로 그대로…)

→ Process ∝ problem size

SISD, SIMD, MIMD, SPMD, Vector

Classification

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE instructions of x86
	Multiple	**MISD**: No examples today	**MIMD**: Intel Xeon e5345

- SPMD : Single Program Multiple Data ← MIMD의 생략된 형태
  - Conditional code for different processors

data-level parallelism

: parallelism on independent data

Vector processors

- Use vector register
  - Highly pipelined function units
  - Reduce instruction fetch bandwidth

vector element는 independent함

Vector vs Scalar

- Vector
  - Simplify data-parallel programming
  - vector element들 간 independent 함수 data hazards check할 필요 X
  - Regular access patterns
    → 메모리의 인접 vector element들이 configuration할 때 very efficient함
  - Avoid Control Hazard
    → Because no loops or to element-wise operation

- Conventional MIPS code

```
      l.d   $f0,a($sp)      ;load scalar a
      addiu r4,$s0,#512     ;upper bound of what to load
loop: l.d   $f2,0($s0)      ;load x(i)
      mul.d $f2,$f2,$f0     ;a x x(i)
      l.d   $f4,0($s1)      ;load y(i)
      add.d $f4,$f4,$f2     ;a x x(i) + y(i)
      s.d   $f4,0($s1)      ;store into y(i)
      addiu $s0,$s0,#8      ;increment index to x
      addiu $s1,$s1,#8      ;increment index to y
      subu  $t0,r4,$s0      ;compute bound
      bne   $t0,$zero,loop  ;check if done
```

- Vector MIPS code

```
      l.d     $f0,a($sp)    ;load scalar a
      lv      $v1,0($s0)    ;load vector x
      mulvs.d $v2,$v1,$f0   ;vector-scalar multiply
      lv      $v3,0($s1)    ;load vector y
      addv.d  $v4,$v2,$v3   ;add y to product
      sv      $v4,0($s1)    ;store the result
```

→ DAXPY
: double precision
a x X plus Y

SISD, SIMD, MIMD, SPMD, Vector

SIMD

: Operate element wise on vectors of data

- All processors가 different data를 가지고 same operation 수행
- reduced → instruction control hardware
- DP applications에 유용함

vector ⊗ MMX (Multi Media Extension)

Hardware Multithreading

Hardware Multithreading
: Increasing utilization of processor by switching to another when one thread is stalled
→ processor의 효율을 높이는 것 ★

Thread
: light-weight process ✓
- PC, register state, stack ← fast switching
- share single address space (share process)

Process
: include one or more threads, address space, OS state, ... ← slow switching

Fine-grain Multithreading
- switch thread after each clock (each instruction)
- Interleave instruction execution
- If one thread stalls, others are executed

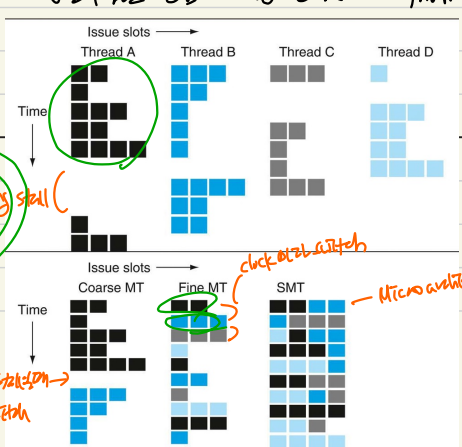
→ stall 시간을 overlap 해서 2개의 stall 없이 interleave!

Coarse-grain Multithreading
- only switch on long stall

like low-level cache miss

Simultaneous Multithreading

Simultaneous Multithreading (SMT)
: 동시에 실행할 수 있는 instruction 여러 개를 fetch … dynamically scheduled microarchitecture



SMP ⊂ SMT (Simultaneous Multithread)
long stall

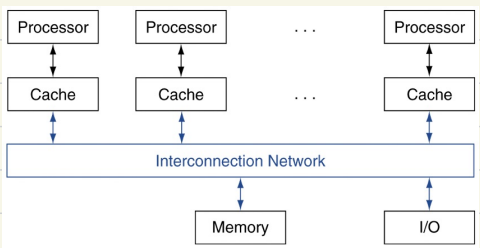
Multicore · Other Shared Memory Multiprocessor

Shared Memory Multiprocessor

SMP

: Shared Memory Multiprocessor

- HW provides single physical address space for processors
- Synchronize shared variables using locks
- Memory access time
  - UMA : uniform Memory Access
  - NUMA : Non-Uniform Memory Access



Synchronization : Process of coordinating behavior of two or more processes

lock : synchronization tool. allowing only 1 access to data at time!

Multicore · Other Shared Memory Multiprocs.

Example

: Sum 64K numbers on 64 processor UMA

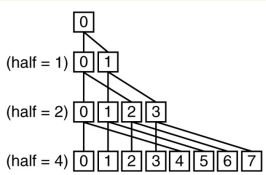
- Partition 1K numbers per processor

```
sum[Pn] = 0;
 for (i = 1000*Pn;
       i < 1000*(Pn+1); i += 1)
   sum[Pn] += A[i];
```

different partition?

API
(Application Program Interface)

- Need to add local sums → Processes data structure and return single value
  → use Reduction … use divide n conquer




```
half = 64;
do
  synch();
  if (half%2 != 0 && Pn == 0)
    sum[0] += sum[half-1];
    /* Conditional sum needed when half is odd;
       Processor0 gets missing element */
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] += sum[Pn+half];
while (half > 1);
```

UMA

Bonus Awkw

Introduction to Graphics Processing Units

GPU

: Graphic Processing Units

GPU Architecture

- highly data parallel
- Use thread switching to hide memory latency
- Memory : wide / high bandwidth
  한번에 transfer 할 수 있는 data 양이 많다!
- General purpose GPUs
  - hetero geneous CPU (sequential code) / GPU (parallel code) systems

NVIDIA Tesla

- Multiple SIMD processors, each as shown: