Multiprocessing and Multithreading

Slide courtesy: S. R. Sarangi

Slides adapted by: Sparsh Mittal

Multiprocessing

* The term multiprocessing refers to multiple processors working in parallel.
* This is a generic definition, and it can refer to multiple processors in the same chip, or processors across different chips.
* A multicore processor is a specific type of multiprocessor that contains all of its constituent processors in the same chip. Each such processor is known as a **core**.

Symmetric vs Asymmetric MPs

Symmetric Multiprocessing: *This paradigm treats all the constituent processors in a multiprocessor system as the same. Each processor has equal access to the operating system, and the I/O peripherals. These are also known as SMP systems.*

Asymmetric Multiprocessing: *This paradigm does not treat all the constituent processors in a multiprocessor system as the same. There is typically one master processor that has exclusive control of the operating system and I/O devices. It assigns work to the rest of the processors.*

Types of Parallelism

- Instruction Level Parallelism
  - Different instructions within a stream can be executed in parallel
  - Pipelining, out-of-order execution, speculative execution, VLIW
  - Dataflow

- Data Parallelism
  - Different pieces of data can be operated on in parallel
  - SIMD: Vector processing, array processing
  - Systolic arrays, streaming processors

- Task Level Parallelism
  - Different "tasks/threads" can be executed in parallel
  - Multithreading
  - Multiprocessing (multi-core)

4

Moore's Law

- A processor in a cell phone is 1.6 million times faster than IBM 360 (state of the art processor in the sixties)
- Transistor in the sixties/seventies
  - several millimeters
- Today
  - several nanometers
- The number of transistors per chip doubles roughly every two years → known as Moore's Law

Moore's Law - II

In 1965, Gordon Moore (co-founder of Intel) conjectured that the number of transistors on a chip will double roughly every year. Initially, the number of transistors was doubling every year. Gradualy, the rate slowed down to 18 months, and now it is about two years.

* Feature Size → the size of the smallest structure that can be fabricated on a chip

Year	Feature Size
2001	130 nm
2003	90 nm
2005	65 nm
2007	45 nm
2009	32 nm
2011	22 nm
2014	14 nm

Strongly vs Loosely Coupled Multiprocessing

Loosely Coupled Multiprocessing: *Running multiple unrelated programs in parallel on a multiprocessor is known as loosely coupled multiprocessing.*

Strongly Coupled Multiprocessing: *Running a set of programs in parallel that share their data, code, file, and network connections is known as strongly coupled multiprocessing.*

Shared Memory vs Message Passing

* Shared Memory
  * All the programs share the virtual address space.
  * They can communicate with each other by reading and writing values from/to shared memory.
* Message Passing
  * Programs communicate between each other by sending and receiving messages.
  * They do not share memory addresses.

Let us write a parallel program

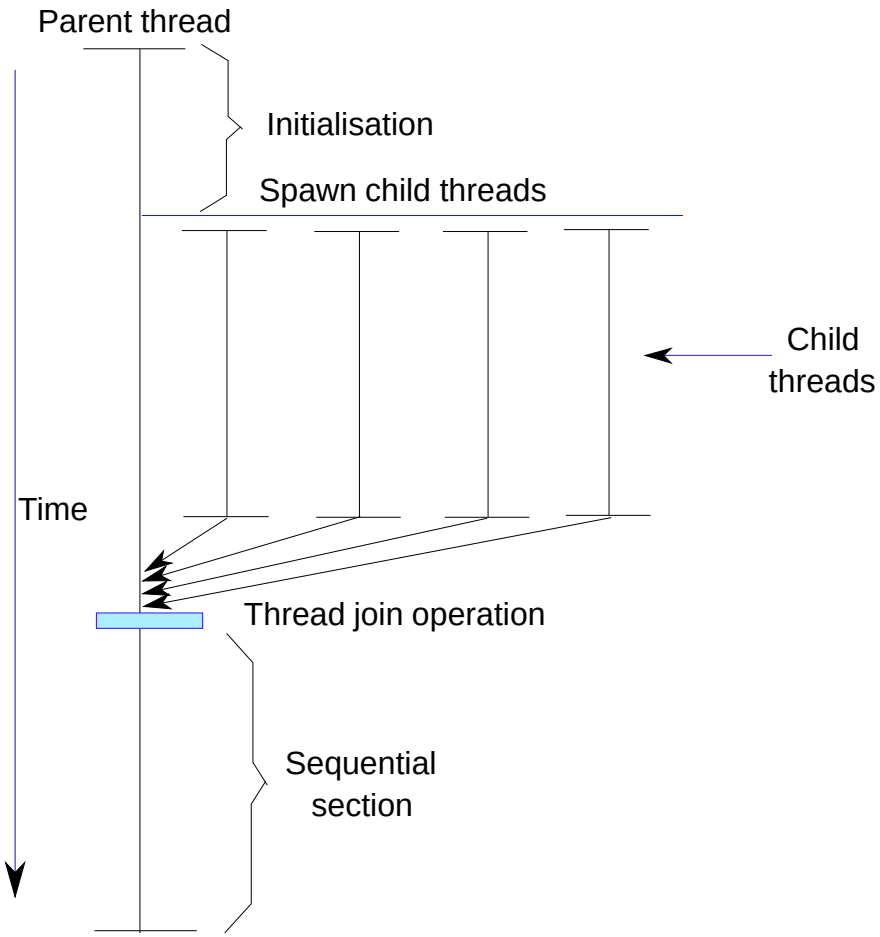
- Write a program using shared memory to add *n* numbers in parallel
  - Number of parallel sub-programs → NUMTHREADS
  - The array numbers contains all the numbers to be added
  - It contains NUMSIZE entries
- We use the OpenMP extension to C++

```
/* variable declaration */
int partialSums[N];
int numbers[SIZE];
int result = 0;

/* initialise arrays */
...

/* parallel section */
#pragma omp parallel {
        /* get my processor id */
        int myId = omp_get_thread_num();

        /* add my portion of numbers */
        int startIdx = myId * SIZE/N;
        int endIdx = startIdx + SIZE/N;
        for(int jdx = startIdx; jdx < endIdx; jdx++)
           partialSums[myId] += numbers[jdx];
}

/* sequential section */
for(int idx=0; idx < N; idx++)
        result += partialSums[idx];
```

The Notion of Threads

- We spawn a set of separate threads
- Properties of threads
  - A thread shares its address space with other threads
  - It has its own program counter, set of registers, and stack
  - A process contains multiple threads
  - Threads communicate with each other by writing values to memory or via synchronization operations

Operation of the Program



Message Passing

* Typically used in loosely coupled systems
* Consists of multiple processes.
* A process can send (unicast/ multicast) a message to another process
* Similarly, it can receive (unicast/ multicast) a message from another process

Flynn's Taxonomy

Flynn's Classification

* Instruction stream → Set of instructions that are executed
* Data stream → Data values that the instructions process
* Four types of multiprocessors : SISD, SIMD, MISD, MIMD

SISD and SIMD

* SISD → Standard uniprocessor
* SIMD → One instruction, operates on multiple pieces of data. Vector processors have one instruction that operates on many pieces of data in parallel. For example, one instruction can compute the $\sin^{-1}$ of 4 values in parallel.

MISD

- MISD → Mutiple Instruction Single Data
  - Very rare in practice
  - Consider an aircraft that has a MIPS, an ARM, and an X86 processor operating on the same data (multiple instruction streams)
  - We have different instructions operating on the same data
  - The final outcome is decided on the basis of a majority vote.

MIMD

* MIMD → Multiple instruction, multiple data (two types, SPMD, MPMD)
* SPMD → Single program, multiple data. Examples: OpenMP example that we showed, or the MPI example that we showed. We typically have multiple processes or threads executing the same program with different inputs.
* MPMD → A master program, delegates work to multiple slave programs. The programs are different.

Summary

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

21

Multithreading

Multithreading

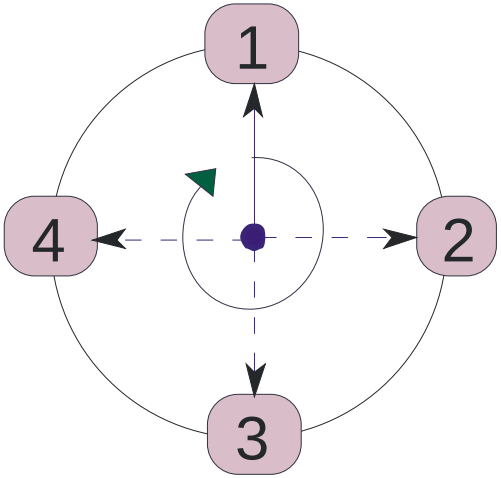
- Multithreading → A design paradigm that proposes to run multiple threads on the same pipeline.
- Three types
  - Coarse grained
  - Fine grained
  - Simultaneous

Analogy

* Consider a car which can be shared by 4 people (A, B, C, D) in following way:
* Option1: Four people ride the car simultaneously almost all the time to go to their destinations.
* Option2: A uses the car for 15 days, B for 14 days, C for 18 days and D for 16 days (and repeat).
* Option3: A uses the car for 8 hours, B for 9 hours, C for 5 hours and D for 7 hours (and repeat).
* Match the above three options to three types of multithreading

Coarse Grained Multithreading

* Assume that we want to run 4 threads on a pipeline
* Run thread 1 for n cycles, run thread 2 for n cycles, ….



Implementation

- Steps to minimise the context switch overhead
- For a 4-way coarse grained MT machine
  - 4 program counters
  - 4 register files
  - 4 flags registers
  - A context register that contains a thread id.
  - Zero overhead context switching → Change the thread id in the context register

Advantages

- Assume that thread 1 has an L2 miss
  - **Wait** for 200 cycles
  - Schedule thread 2
  - Now let us say that thread 2 has an L2 miss
  - Schedule thread 3
- We can have a **sophisticated algorithm** that switches every n cycles, or when there is a **long latency event** such as an L2 miss.
  - Minimises idle cycles for the entire system

Fine Grained Multithreading

- The switching granularity is very small
  - 1-2 cycles
- **Advantage** :
  - Can take advantage of low latency events such as division, or L1 cache misses
  - Minimise idle cycles to an even greater extent
- **Correctness Issues**
  - We can have instructions of 2 threads simultaneously in the pipeline.
  - We never forward/interlock for instructions across threads

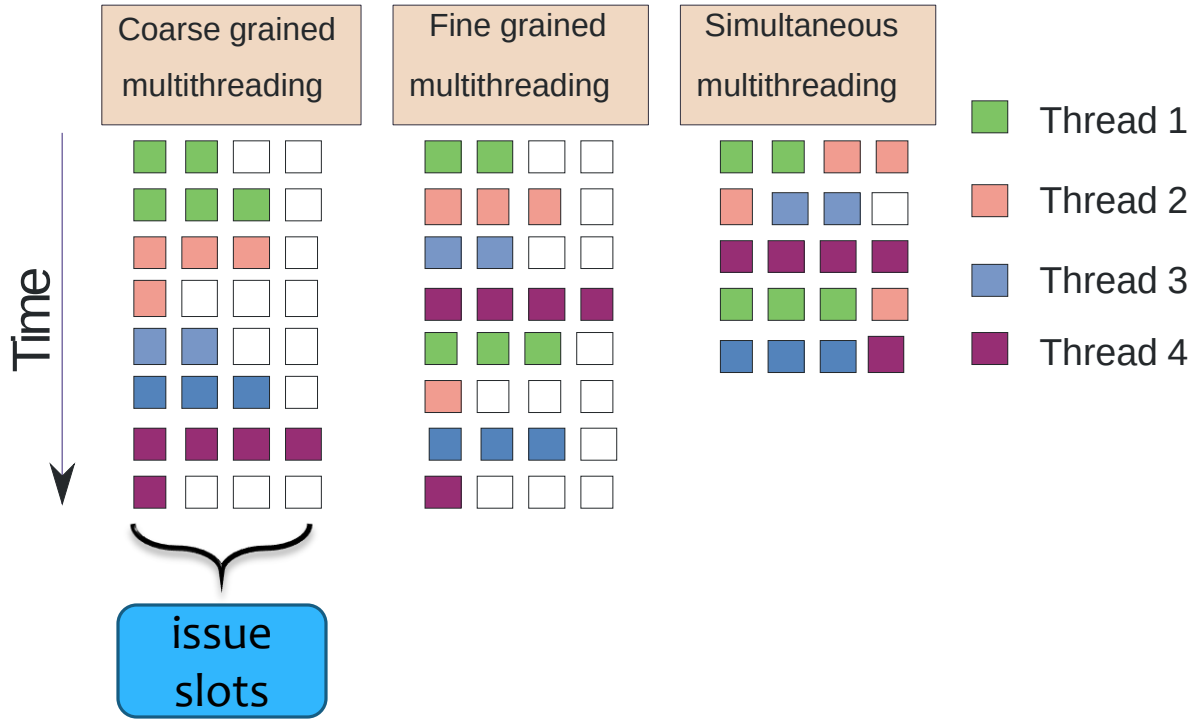
Simultaneous Multithreading

* Most modern processors have multiple issue slots
  * Can issue multiple instructions to the functional units
  * For example, a 3 issue processor can fetch, decode, and execute 3 instructions per cycle
  * If a benchmark has low ILP (instruction level parallelism), then fine and coarse grained multithreading cannot **really help**.

Simultaneous Multithreading

- Main Idea
  - Partition the issue slots across threads
  - **Scenario** : In the same cycle
    - Issue 2 instructions for thread 1
    - and, issue 1 instruction for thread 2
    - and, issue 1 instruction for thread 3
- Support required
  - Need smart instruction selection logic.
    - Balance fairness and throughput

Summary


Coarse grained multithreading
Fine grained multithreading
Simultaneous multithreading
Time
issue slots
Thread 1
Thread 2
Thread 3
Thread 4