Parallel Computational Models

Formally, a Computational Model is a Coherent collection of mechanisms for
- communication
- synchronization
- partitioning
- placement
- scheduling

Computational model defined at all levels of abstraction:
- Machine hardware - e.g., a shared-memory machine
- Language - e.g., a message passing language
- Algorithm - e.g., a CREW (concurrent read, exclusive write) shared-memory algorithm

Let's review some examples to build intuition
For Reference: Sequential Programming Model

Blackboard captures state

Designers

I/O

State stored in memory

Process operates on data in memory

I/O puts external data into memory

Shared Memory Parallel Programming Model

Blackboard captures state

Communication: via shared memory

Synchronization: shared memory locks

Discuss:
What is it good for; not good for?
Shared memory programming models have many variants; we will revisit in a later lecture
**Shared Memory Machine Model**

**Uniform access shared memory (SRAM)**

- Processor 0
- Processor 1

- **Communication:** via shared memory
- **Synchronization:** shared memory locks

  Locks can be done by holding the bus and performing back to back load-store

(Historical: MIMD - Multiple instruction multiple data)

**Shared Memory Machine Model**

**Non-uniform access shared memory (SRAM)**

- Processor 0
- Processor 1

- **Can replace bus with a ring (Beehive), or mesh (Tile processor)**
Beehive and Modern Manycores also have Per-Processor Caches

Non-uniform access shared memory (SRAM)

Caches introduces the cache coherence problem - we will study this in depth later in the course

Historical note: the cache coherence problem occupied computer architects for an entire decade in the 90’s!

Need for Synchronization

Uniform access shared memory (SRAM)

When should Processor 0 read the data being written by Processor 1?

Producer-Consumer synchronization
Need for Synchronization

Uniform access shared memory (SRAM)

Shared data

How to safely increment shared counter?

Final state is not correct!

Mutual exclusion synchronization
(Atomic: if open, then lock, else retry)
Hold bus captive for read/write on lock
**Need for Synchronization**

Uniform access shared memory (SRAM)

Processor 0 Processor 1

BUS

Shared data

How to safely increment shared counter?

Mutual exclusion synchronization
(Atomic: if open, then lock, else retry)
Hold bus captive for read/write on lock

**Shared Memory Algorithm Model**

PRAM – Parallel Random Access Memory

Shared memory

Processors

Variants
- Multiple simultaneous R,W -- CRCW PRAM
- Exclusive writes only -- CREW PRAM
- Exclusive R & W -- EREW PRAM

... may be realistic..... or not.

Summary: we just saw shared memory programming model, shared memory machine model, and shared memory algorithm model
Message Passing Parallel Programming Model

Communication: via messages
Synchronization: via messages

Discuss:
What is it good for; not good for?
Inspired by object oriented programming model

Message Passing Parallel Machine Model

Communication via messages.
Send/receive msg are new instructions
(Historical: MIMD - Multiple instruction multiple data)

Synchronization via messages
Message can achieve communication and synchronization in a single action
Message Passing Parallel Machine Model

Communication via messages
Can also use a bus or mesh (or other interconnect) for communication

Can replace local memories with private caches (as in Beehive).
Cache demand fetch data from main memory as needed.
Since there is no shared data in the message passing model, there is no cache coherence problem.
Producer Consumer Synchronization in Message Passing Model

Producer sends data when it is ready, so receiver can assume received data in message is good.

Message can achieve communication and synchronization in a single action.

Mutual Exclusion Synchronization in Message Passing Model

Processor 0 is in charge of counter object. If you want to increment counter, send message to processor 0. Processor 0 serializes multiple requests. Message can also contain a piece of code (or a pointer to code) that Processor 0 should run (variously called active message, future).

Msg: incr by 1

Msg: incr by 1

Msg: incr by 1

Msg: if val smaller than count, replace count with val
Message Passing Model
To Share or Not to Share

Counter
object
memory

local
memory

local
memory

local
memory

Processor 0
Processor 1
RING

Message passing models often share immutable initial state, read-only data, etc. This immutable shared data is often copied into all local memories at initialization.

No sharing of data (almost).
Discuss: One thing is still shared, what is it? So, how do you share/bootstrap?

Message passing models often share immutable initial state, read-only data, etc. This immutable shared data is often copied into all local memories at initialization.

Summary: we just saw shared memory programming model, shared memory machine model, and shared memory algorithm model.
Many More Computational Models Exist

Streaming model
Dataflow model
Data parallel model
Hybrid models
Invent a new one and get a phd...
Details in 6.846