Intro to Parallel Computing

20 January 2009
CMPT370
Dr. Sean Ho
Trinity Western University

Carmel

- We'll be using carmel's “8” processors
  - OpenMP under gcc4
  - Linux command-line (ssh/PuTTY); no GUI
- If you don't have a carmel login yet, see Dave Friesen <davef@twu.ca>
- Try out the testbed program:
  - Download nbody and runtest to carmel
  - './runtest' and watch the results

Some UNIX / Cygwin tips

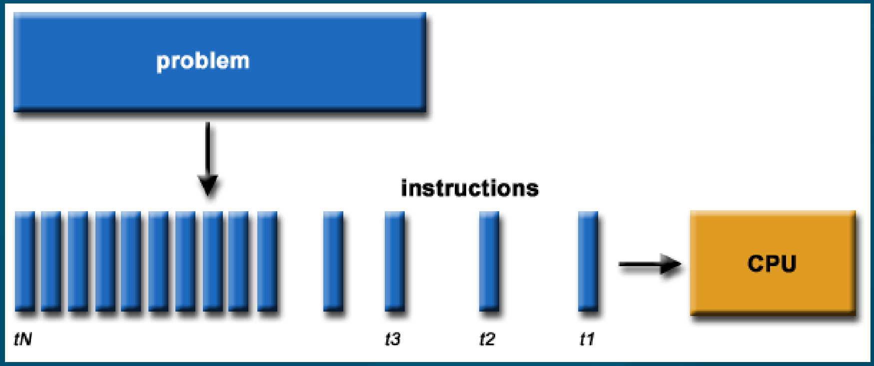
- Cygwin has command and filename completion:
  - Type first few characters and press <Tab>
- Job control:
  - When a program (e.g., fluid) is running, press Ctrl-Z in the Cygwin window to suspend it
  - Type “fg” to resume the suspended job
  - Or “bg” to let it run in the background
  - Use ampersand: “fluid &” to run in bg
  - “jobs” to show all current jobs

What's on for today

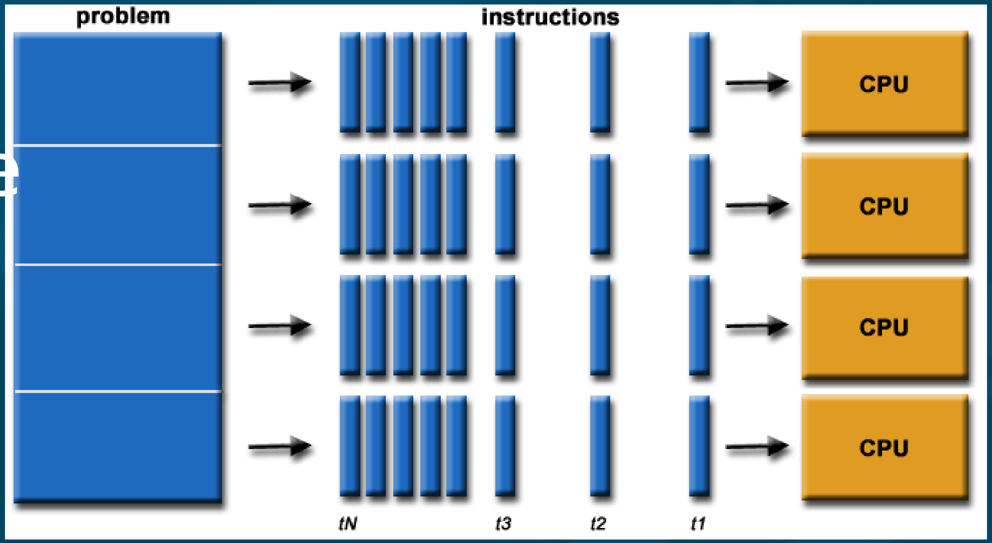
- Parallel computing concepts
  - Why do parallel?
  - vonNeumann abstraction: instructions, data
  - Flynn's taxonomy: SISD, SIMD, MISD, MIMD
  - Terms, measuring speedup
  - Design issues
- See tutorial from LLNL (Livermore) supercomputing centre

Parallel computing

- Sequential computing:
  - Divide task into instructions
  - 1 CPU executes serially

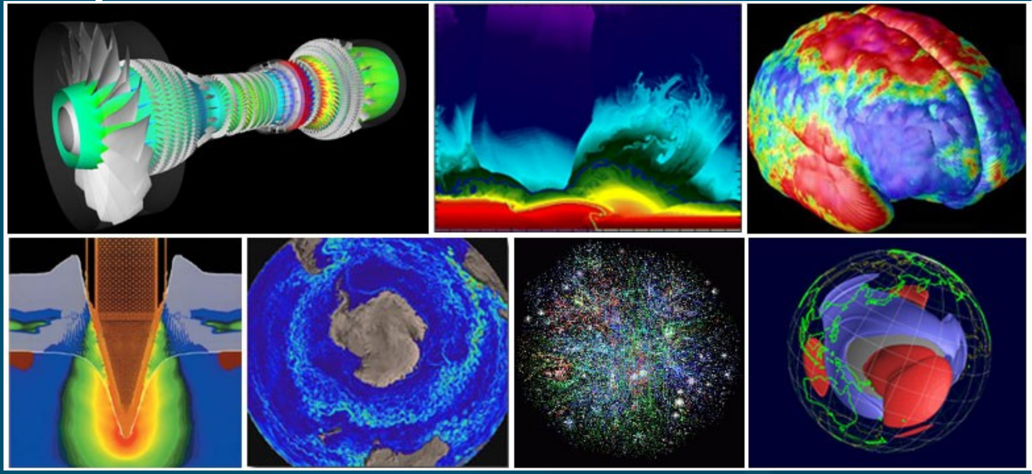


- Parallel computing:
  - Multiple tasks
  - Multiple CPUs execute in parallel
- Accomplish more in the same amount of time



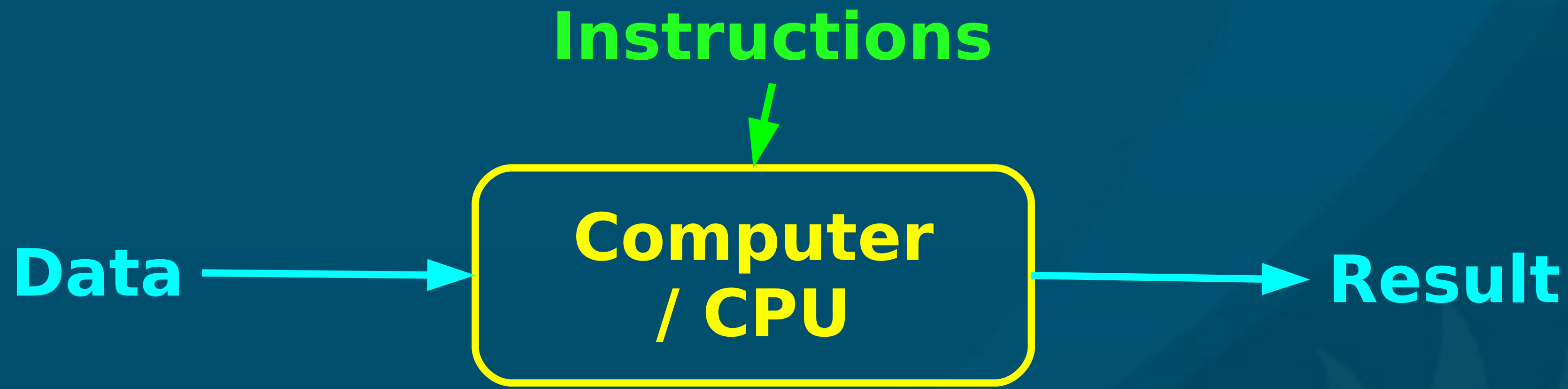
Applications of parallel compute

- Large, compute-intensive problems that can be divided up somehow
  - Weather modelling
  - 3D rendering
  - Data mining
  - Modelling nuclear reactions
  - Protein folding, drug binding sites
  - Aircraft design / comput. fluid dynamics
  - Large-scale satellite / medical image analysis
  - High-visibility website (Amazon, Google, etc.)



Instructions and data

- Computers since the 1960s have used the (John) vonNeumann model of computing:



- CPU follows instructions to operate on data
- vonNeumann's abstraction:
  - Instructions and data both stored in memory
  - Self-modifying code: rewrite own instructions

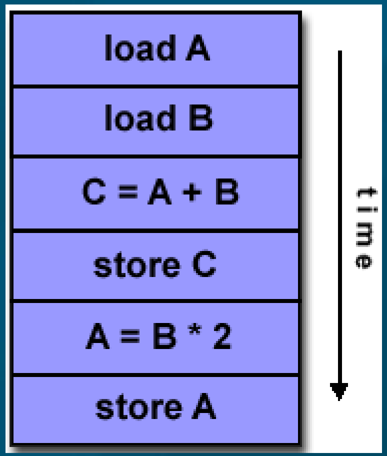
Flynn's taxonomy

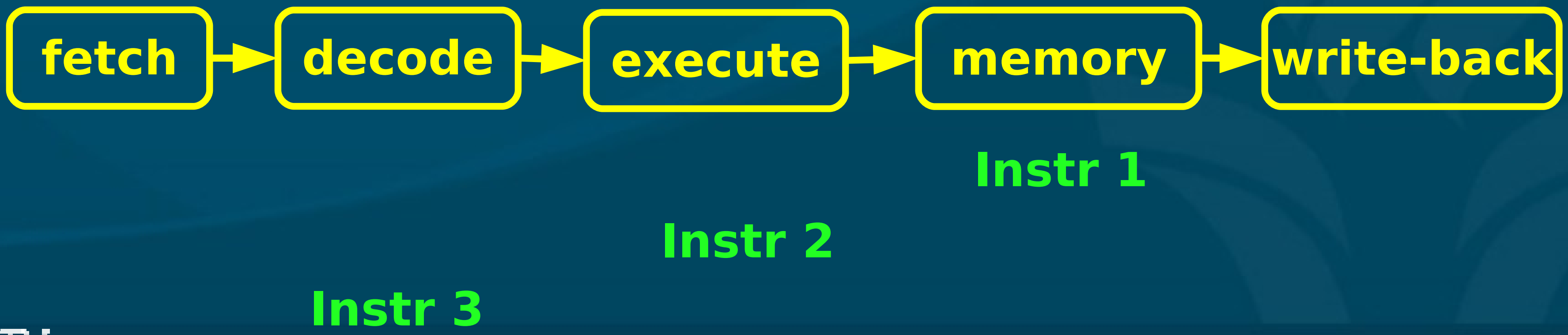
SISD: Single instr Single data	**SIMD:** Single instr Multiple data
MISD: Multiple instr Single data	**MIMD:** Multiple instr Multiple data

- Multiple data: divide up the problem by data
  - Each processor operates on a chunk of data
- Multiple instructions:
  - Each processor does a different task

SISD: single instr, single data

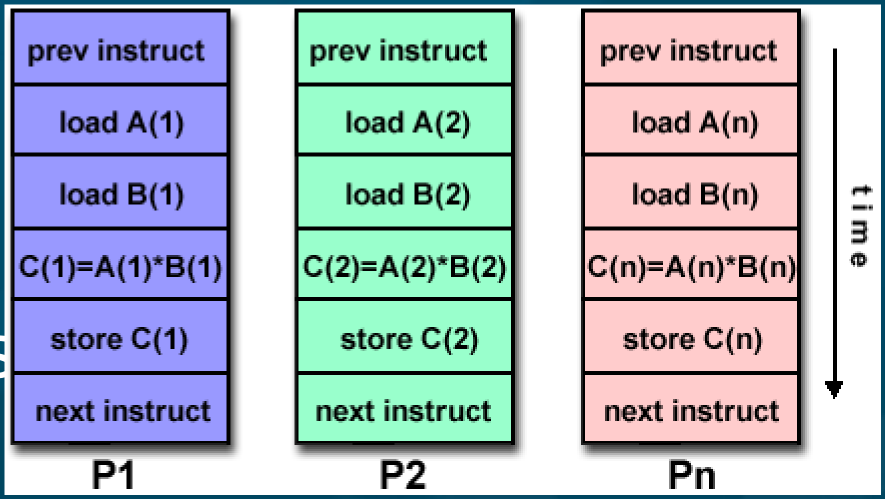
- SISD is the classical uniprocessor situation
  - Serial execution
- Processing can still be pipelined:
  - Each instruction has multiple parts
  - Each part is done by a different CPU component





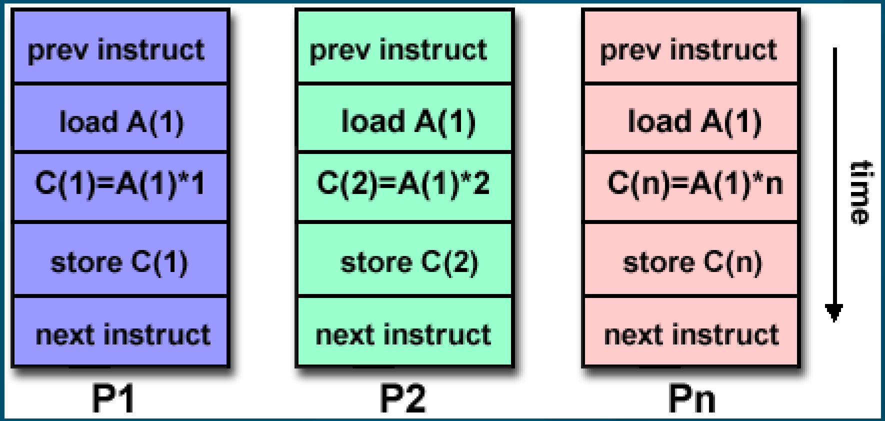
SIMD: single instr, multiple data

- SIMD: same operations on multiple data in parallel
  - Quite common on today's CPUs
  - Intel MMX, SSE, Apple Altivec
  - CM-2, Cray C90
- Vector processing: perform one operation on a whole vector of numbers
  - Add two RGBA values (128 bits each)

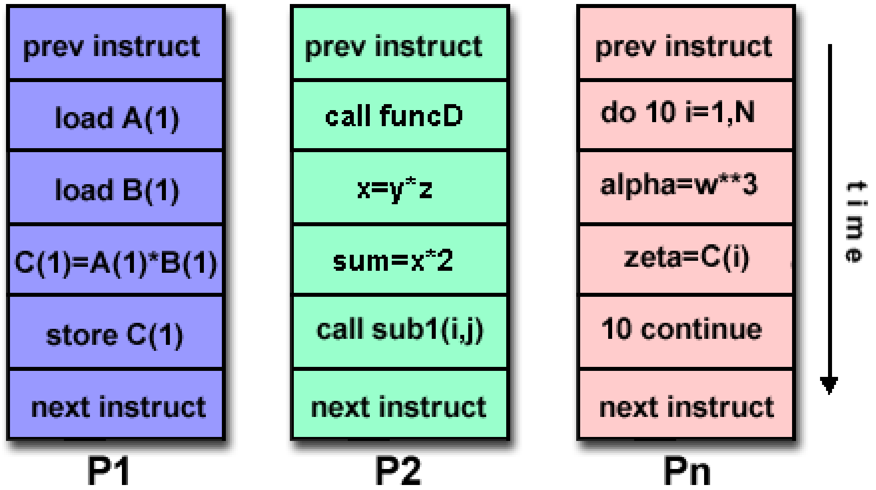


MISD: multiple instr, single data

- MISD:
  - Run the same data through different programs in parallel
  - Not often seen in hardware
  - Potential applications:
    - One encrypted message to crack; try several algorithms in parallel
    - One satellite image to process; run several image processing filters in parallel



MIMD: mult instr, mult data



- MIMD:
  - Each processor is independent, runs its own task on its own data
- "Parallel computer" usually means MIMD
  - Most modern supercomputers
  - Dual-core (e.g., carmel Xeon)
- MIMD is most flexible, but also most complex:
  - Synchronization between processors
  - Shared memory access

Measuring speedup

- A parallelizable task can be broken up into discrete tasks (SIMD/MIMD), one per processor
- The parallel speedup is:

  (serial execution time) / (parallel execution time)

- Ideal speedup is linear with # processors
- Reality is not so sweet:
  - Overhead in setting up parallel tasks
  - Communication between processors
  - Synchronization points mean waiting for slowest task

Design issues in parallel computing

- Memory model
  - Shared: all CPUs access same memory (SMP)
  - Distributed: each CPU local memory (cluster)
- Granularity: how often to communicate?
  - Coarse: lots of computation between communication events
  - Fine: processors frequently talk to each other
- Scalability
  - How many processors do we want to scale?
  - Communications network?