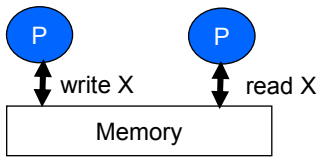
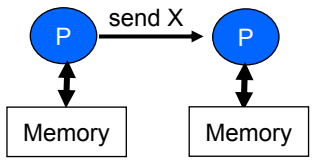
18-742

Lecture 4

Parallel Programming II





Spring 2005
Prof. Babak Falsafi
http://www.ece.cmu.edu/~ece742

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Reinhardt, Smith, and Singh of University of Illinois, Carnegie Mellon University, University of Wisconsin, Duke University, University of Michigan, and Princeton University.

Homework & Reading

Projects handout

- **On Friday**
- **Form teams, groups of two**

Homework

- **Homework 2 due by Friday**
- **Homework 3 will be review questions only**

Reading

- **Chapter 4**

Partitioning for Performance

- **Balance workload**
  - reduce time spent at synchronization
- **Reduce communication**
- **Reduce extra work**
  - determining and managing good assignment
- **These are at odds with each other**

Data vs. Functional Parallelism

- **Data Parallelism**
  - same ops on different data items
- **Functional (control, task) Parallelism**
  - pipeline
- **Impact on load balancing?**
- **Functional is more difficult**
  - longer running tasks

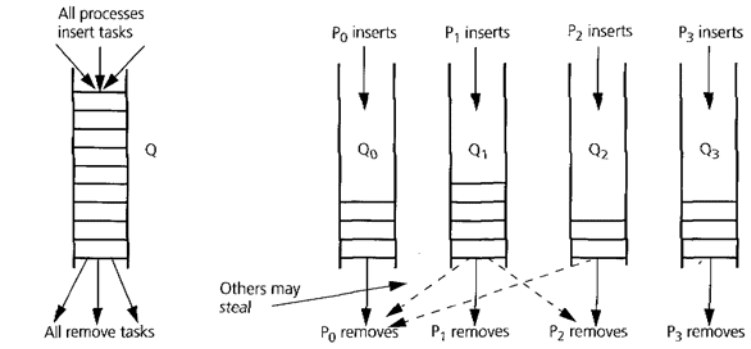
Impact of Task Granularity

- Granularity = Amount of work associated with task
- Large tasks
  - worse load balancing
  - lower overhead
  - less contention
  - less communication
- Small tasks
  - too much synchronization
  - too much management overhead
  - might have too much communication (affinity scheduling)

Impact of Concurrency

- Managing Concurrency
  - load balancing
- Static
  - Can not adapt to changes
- Dynamic
  - Can adapt
  - Cost of management increases
  - Self-scheduling (guided self-scheduling)
  - Centralized task queue
    - » contention
  - Distributed task queue
    - » Can steal from other queues
    - » Arch: Name data associated with stolen task

Task Queues



(a) Centralized task queue

(b) Distributed task queues (one per process)

Dynamic Load Balancing

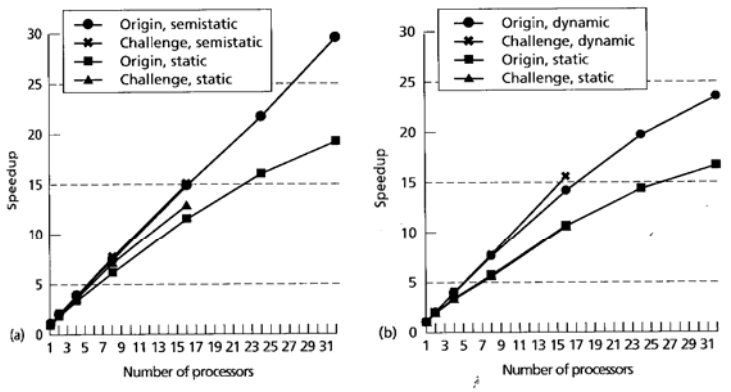


FIGURE 3.2 Illustration of the performance impact of dynamic partitioning for load balance. The graph in (a) shows the speedups of the Barnes-Hut application with and without semistatic partitioning, and the graph in (b) shows the speedups of Raytrace with and without dynamic tasking. Even in these applications that have a lot of parallelism, dynamic partitioning is important for improving load balance over static partitioning.

Impact of Synchronization and Serialization

- Too coarse synchronization
  - barriers instead of point-to-point synch
  - poor load balancing
- Too many synchronization operations
  - lock each element of array
  - costly operations
- Coarse grain locking
  - lock entire array
  - serialize access to array
- Architectural aspects
  - cost of synchronization operation
  - synchronization name space

Where Do Programs Spend Time?

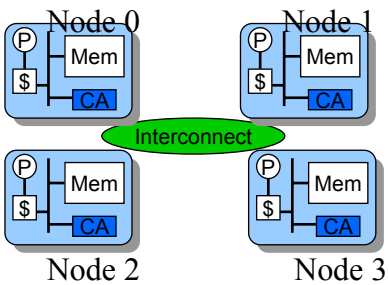
- Sequential
  - Performing real computation
  - Memory system stalls
- Parallel
  - Performing real computation
  - Stalled for local memory
  - Stalled for remote memory (communication)
  - Synchronizing (load imbalance and operations)
  - Overhead
- Speedup (p) = time(1)/time(p)
  - Amdahl's Law (low concurrency limits speedup)
  - Superlinear speedup possible (how?)

Cache Memory 101

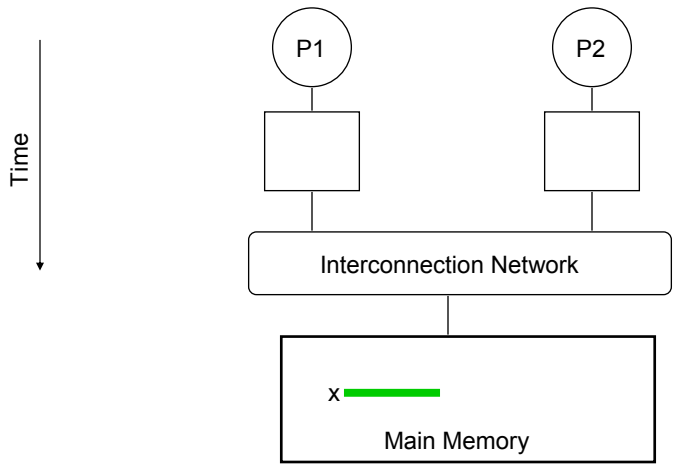
- **Locality + smaller HW is faster = memory hierarchy**
  - *Levels*: each smaller, faster, more expensive/byte than level below
  - *Inclusive*: data found in top also found in the bottom
- **Definitions**
  - *Upper* is closer to processor
  - *Block*: minimum unit of data present or not in upper level
  - Frame: HW (physical) place to put block (same size as block)
  - Address = *Block address* + *block offset address*
  - *Hit time*: time to access upper level, including hit determination
- 3C Model
  - compulsory, capacity, conflict
- Add another C: communication misses

Multiprocessor as an Extended Mem. Hier.

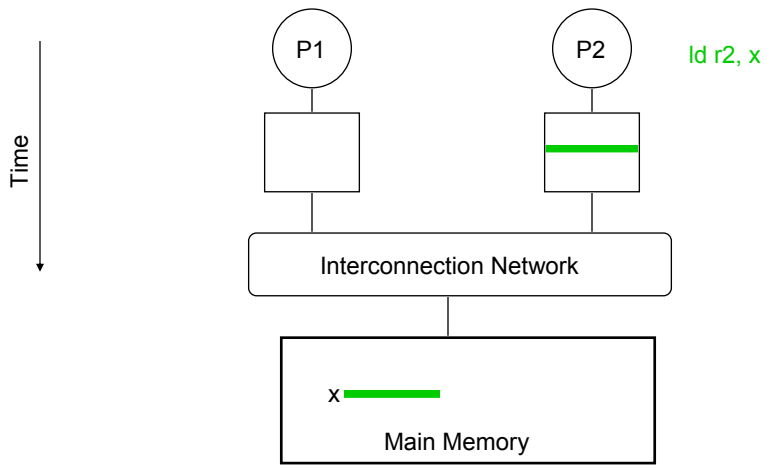
- **Example:**
  - **computation: 2 instructions per cycle (IPC)**
    - » **or 500 cycles per 1000 instructions**
  - **1 long miss per 1000 instructions**
    - » **200 cycles per long miss**
  - **=> 700 cycles per 1000 instructions (40% slowdown)**



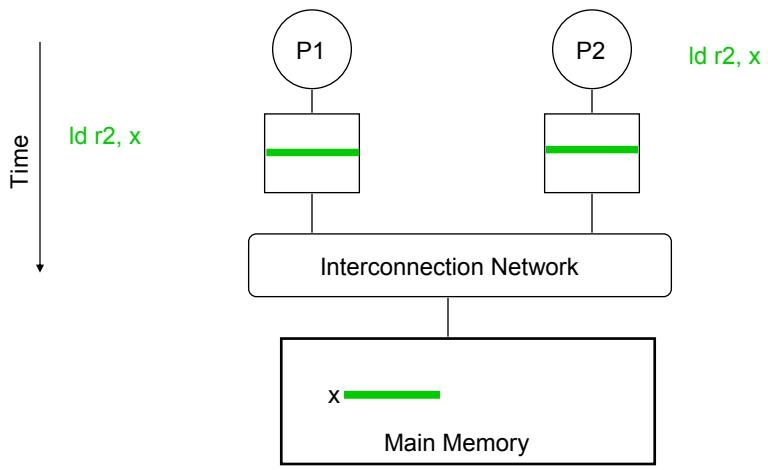
Cache Coherent Shared Memory



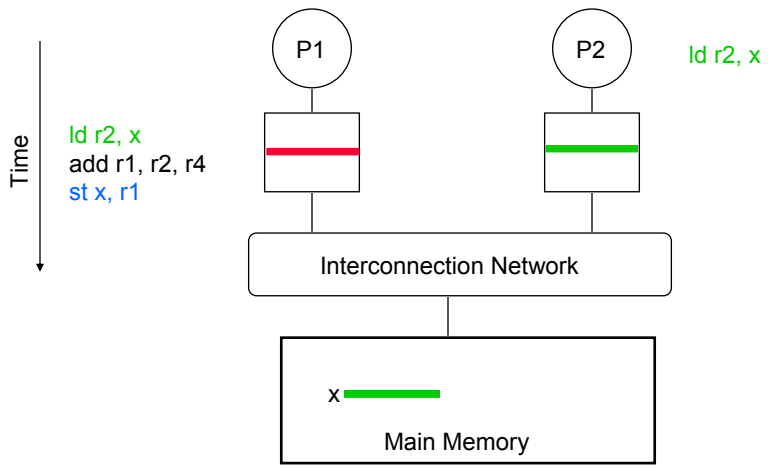
Cache Coherent Shared Memory



Cache Coherent Shared Memory



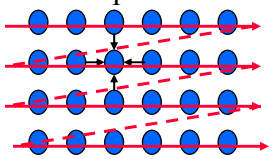
Cache Coherent Shared Memory



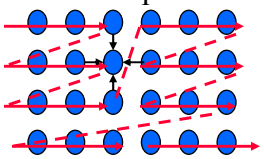
Reorder Computation for Caches

- **Exploit Temporal and Spatial Locality**
  - **Temporal locality affects replication**
  - **Touch too much data == capacity misses**
- **Computation Blocking**

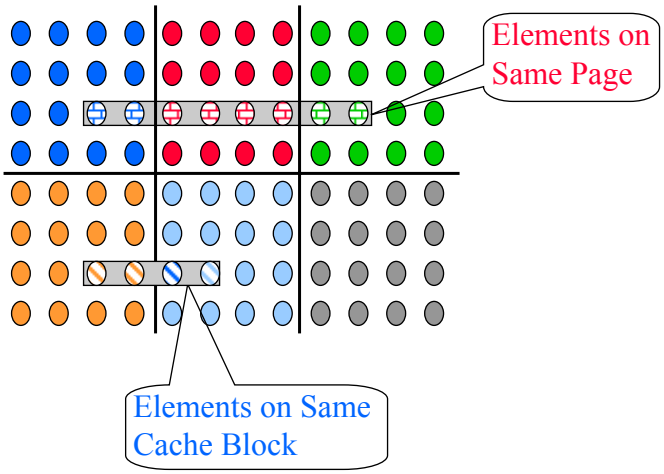
Naïve Computation Order



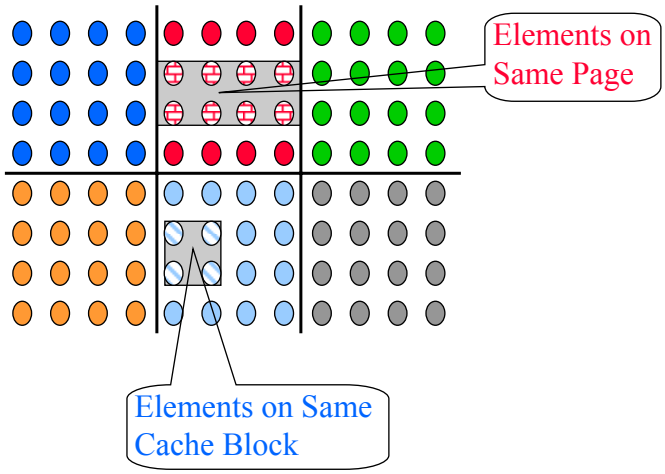
Blocked Computation order



Reorder Data for Caches: Before



Reorder Data for Caches: With Blocking



Scaling: Why is it important?

- **Over time:**
  - **computer systems become larger and more powerful**
    - » **more powerful processors**
    - » **more processors**
    - » **also range of system sizes within a product family**
  - **problem sizes become larger**
    - » **simulate the entire plane rather than the wing**
  - **required accuracy becomes greater**
    - » **forecast the weather a week in advance rather than 3 days**
- **Scaling:**
  - **How do algorithms and hardware behave as systems, size, accuracies become greater?**
- **Intuitively:**
  - **"Performance" should scale linearly with cost**

Cost

- Cost is a function of more than just the processor.
- Cost is a complex function of many hardware components and software
- Cost is often not a "smooth" function
  - Often a function of packaging
    - » how many pins on a board
    - » how many processors on a board
    - » how many boards in a chassis

Aside on Cost-Effective Computing

- Isn't Speedup(P) < P inefficient?
- If only throughput matters, use P computers instead?

- But much of a computer's cost is NOT in the processor [Wood & Hill, IEEE Computer 2/95]
- Let Costup(P) = Cost(P)/Cost(1)
- Parallel computing cost-effective:
- Speedup(P) > Costup(P)
- E.g. for SGI PowerChallenge w/ 500MB:
- Costup(32) = 8.6

Questions in Scaling

- **Fundamental Question:**

  What do real users actually do when they get access to larger parallel machines?

- **Constant Problem Size**
  - **Just add more processors**
- **Memory Constrained**
  - **Scale data size linearly with # of processors**
  - **Can significantly increase execution time**
- **Time Constrained**
  - **Keep same wall clock time as processors are added**
  - **Solve largest problem in same amount of time**

Problem Constrained Scaling

- **User wants to solve same problem, only faster**
  - **E.g., Video compression & VLSI routing**
- $Speedup_{PC}(p) = \frac{Time(1)}{Time(p)}$

- ***Assessment***
  - ***Good: easy to do & explain***
  - ***May not be realistic***
  - ***Doesn't work well for much larger machine (c.f., Amdahl's Law)***

Time Constrained Scaling

- **Execution time is kept fixed as system scales**
  - **User has fixed time to use machine or wait for result**
- **Performance = Work/Time as usual, and time is fixed, so**

$$\textit{SpeedupTC}(p) = \frac{Work(p)}{Work(1)}$$

- **Assessment**
  - **Often realistic (e.g., best weather forecast over night)**
  - **Must understand application to scale meaningfully (would scientist scale grid, time step, error bound, or combination?)**
  - **Execution time on a single processor can be hard to get (no uniprocessor may have enough memory)**

Memory Constrained Scaling

- **Scale so memory usage per processor stays fixed**
- **Scaled Speedup: Is Time(1) / Time(p)?**

$$\textit{Speedup}_{MC}(p) = \frac{Work(p)}{Time(p)} \times \frac{Time(1)}{Work(1)} = \frac{Increase\ in\ Work}{Increase\ in\ Time}$$

- **Assessment**
  - **Realistic for memory-constrained programs (e.g., grid size)**
  - **Can lead to large increases in execution time if work grows faster than linearly in memory usage**
  - **e.g. matrix factorization**
    - **» 10,000-by 10,000 matrix takes 800MB and 1 hour on uniprocessor**
    - **» With 1,000 processors, can run 320K-by-320K matrix**
    - **» but ideal parallel time grows to 32 hours!**

Scaling Down

- Scale down to shorten evaluation time on hardware and especially on simulators
- "Scale up" issues apply in reverse
- Must watch out if problem size gets too small
  - Communication dominates computation (e.g., all boundary elements)
  - Problem size gets too small for realistic caches, yielding too many cache hits
    - » Scale caches down considering application working sets
    - » E.g., if a on a realistic problem a realistic cache could hold a matrix row but not whole matrix
    - » Scale cache so it hold only row or scaled problem's matrix

The SPLASH2 Benchmarks

- Kernels
  - Complex 1D FFT
  - Blocked LU Factorization
  - Blocked Sparse Cholesky Factorization
  - Integer Radix Sort
- Applications
  - Barnes-Hut: interaction of bodies
  - Adaptive Fast Multipole (FMM): interaction of bodies
  - Ocean Simulation
  - Hierarchical Radiosity
  - Ray Tracer (Raytrace)
  - Volume Renderer (Volrend)
  - Water Simulation with Spatial Data Structrue (Water-Spatial)
  - Water Simulation without Spatial Data Structure (Water-Nsquared)