Parallel Programming: Case Studies

Todd C. Mowry
CS 418
January 27, 2011

Parallel Application Case Studies

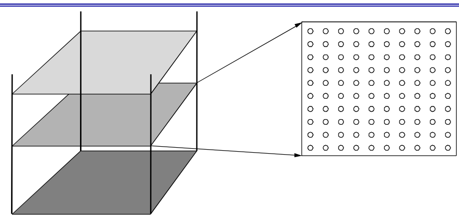
Examine Ocean and Barnes-Hut (others in book)

Assume cache-coherent shared address space

Five parts for each application

- Sequential algorithms and data structures
- Partitioning
- Orchestration
- Mapping
- Components of execution time on SGI Origin2000

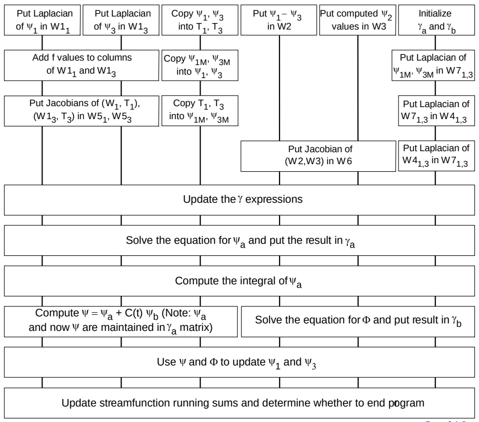
Case 1: Simulating Ocean Currents



(a) Cross sections (b) Spatial discretization of a cross section

- Model as two-dimensional grids
- Discretize in space and time
  - finer spatial and temporal resolution => greater accuracy
- Many different computations per time step
  - set up and solve equations
- Concurrency across and within grid computations

Time Step in Ocean Simulation



Partitioning

Exploit data parallelism

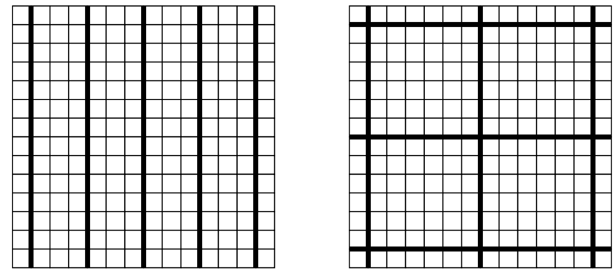
- Function parallelism only to reduce synchronization

Static partitioning within a grid computation

- Block versus strip
  - inherent communication versus spatial locality in communication
- Load imbalance due to border elements and number of boundaries

Solver has greater overheads than other computations

Two Static Partitioning Schemes



Strip Block

Which approach is better?

Orchestration and Mapping

Spatial locality similar to equation solver

- Except lots of grids, so cache conflicts across grids

Complex working set hierarchy

- A few points for near-neighbor reuse, three subrows, partition of one grid, partitions of multiple grids…
- First three or four most important
- Large working sets, but data distribution easy

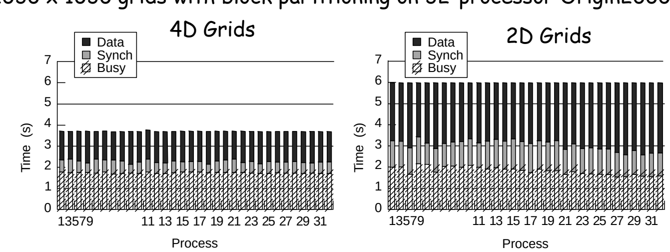
Synchronization

- Barriers between phases and solver sweeps
- Locks for global variables
- Lots of work between synchronization events

***Mapping*: easy mapping to 2-d array topology or richer**

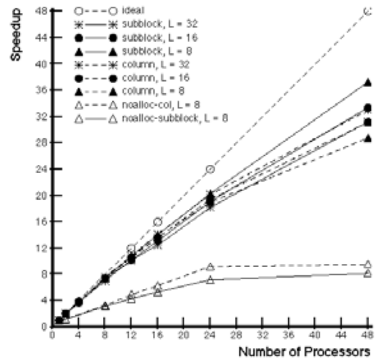
Execution Time Breakdown

•1030 x 1030 grids with block partitioning on 32-processor Origin2000



- **4D grids much better than 2D, despite very large caches on machine**
  - data distribution is much more crucial on machines with smaller caches
- **Major bottleneck in this configuration is time waiting at barriers**
  - imbalance in memory stall times as well

Impact of Line Size & Data Distribution

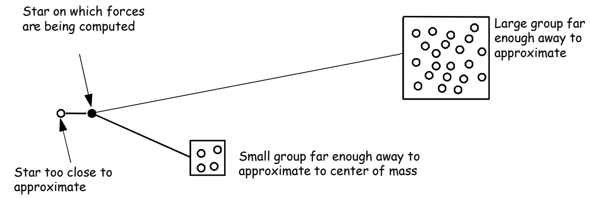


(a) 16 KByte Cache, Grid_98

no-alloc = round-robin page allocation; otherwise, data assigned to local memory. L = cache line size.

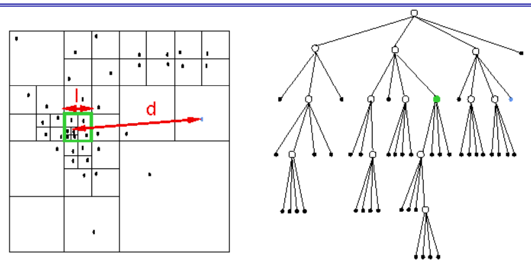
Case 2: Simulating Galaxy Evolution

- Simulate the interactions of many stars evolving over time
- Computing forces is expensive
- $O(n^2)$ brute force approach
- Hierarchical Methods take advantage of force law: $G \frac{m_1 m_2}{r^2}$



- Many time-steps, plenty of concurrency across stars within one

Barnes-Hut



2 d Spatial Domain

Quadtree Representation

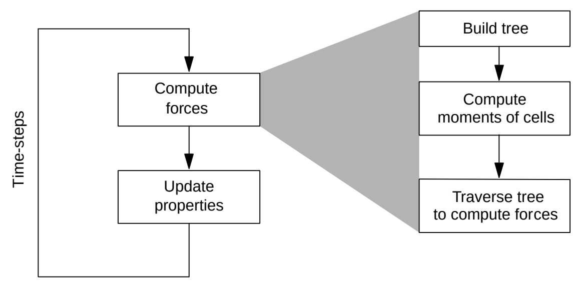
Locality Goal:

- particles close together in space should be on same processor

Difficulties:

- nonuniform, dynamically changing

Application Structure



- Main data structures: array of bodies, of cells, and of pointers to them
  - Each body/cell has several fields: mass, position, pointers to others
  - pointers are assigned to processes

Partitioning

Decomposition: bodies in most phases, cells in computing moments

Challenges for assignment:

- **Nonuniform body distribution => work and comm. Nonuniform**
  - Cannot assign by inspection
- **Distribution changes dynamically across time-steps**
  - Cannot assign statically
- **Information needs fall off with distance from body**
  - Partitions should be spatially contiguous for locality
- **Different phases have different work distributions across bodies**
  - No single assignment ideal for all
  - Focus on force calculation phase
- **Communication needs naturally fine-grained and irregular**

Load Balancing

Equal particles ≠ equal work.

- **<u>Solution</u>: Assign costs to particles based on the work they do**

Work unknown and changes with time-steps

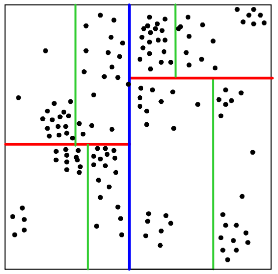
- **<u>Insight</u> : System evolves slowly**
- **<u>Solution</u>: *Count* work per particle, and use as cost for next time-step.**

Powerful technique for evolving physical systems

A Partitioning Approach: ORB

Orthogonal Recursive Bisection:

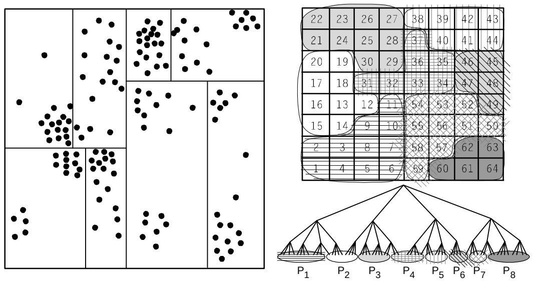
- **Recursively bisect space into subspaces with equal work**
  - Work is associated with bodies, as before
- **Continue until one partition per processor**



- High overhead for large number of processors

Another Approach: Costzones

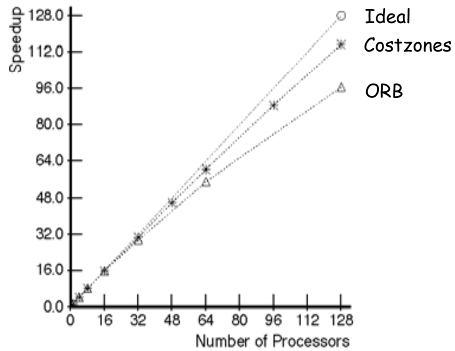
Insight: Tree already contains an encoding of spatial locality.



(a) ORB (b) Costzones

- Costzones is low-overhead and very easy to program

Barnes-Hut Performance



- **Speedups on simulated multiprocessor**
- **Extra work in ORB is the key difference**

Orchestration and Mapping

Spatial locality: Very different than in Ocean, like other aspects

- **Data distribution is much more difficult**
  - Redistribution across time-steps
  - Logical granularity (body/cell) much smaller than page
  - Partitions contiguous in physical space does not imply contiguous in array
  - But, good temporal locality, and most misses logically non-local anyway
- **Long cache blocks help within body/cell record, not entire partition**

Temporal locality and working sets:

- **Important working set scales as $1/\theta^2 \log n$**
- **Slow growth rate, and fits in second-level caches, unlike Ocean**

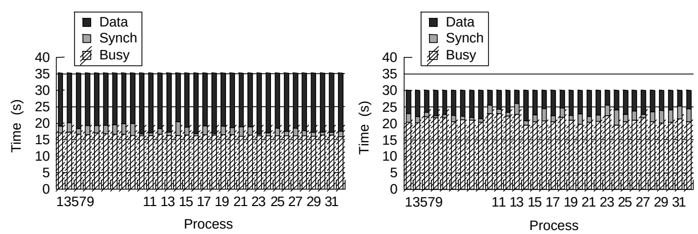
Synchronization:

- **Barriers between phases**
- **No synch within force calculation: data written different from data read**
- **Locks in tree-building, pt. to pt. event synch in center of mass phase**

***Mapping*: ORB maps well to hypercube, costzones to linear array**

Execution Time Breakdown

- 512K bodies on 32-processor Origin2000
  - Static, quite randomized in space, assignment of bodies versus costzones



(a) Static assignment of bodies

(b) Semistatic costzone assignment

- Problem with static case is communication/locality, not load balance!

Case 3: Raytrace

Rays shot through pixels in image are called *primary rays*

- **Reflect and refract when they hit objects**
- **Recursive process generates ray tree per primary ray**

Hierarchical spatial data structure keeps track of primitives in scene

- **Nodes are space cells, leaves have linked list of primitives**

Tradeoffs between execution time and image quality

Partitioning

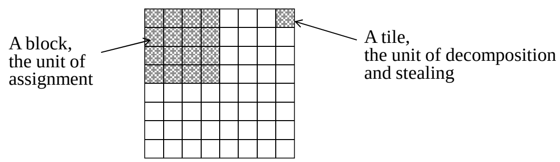
Scene-oriented **approach**

- Partition scene cells, process rays while they are in an assigned cell

Ray-oriented **approach**

- Partition primary rays (pixels), access scene data as needed
- Simpler; used here

Need dynamic assignment; use contiguous blocks to exploit spatial coherence among neighboring rays, plus tiles for task stealing



Could use 2-D interleaved (scatter) assignment of tiles instead

Orchestration and Mapping

Spatial locality

- Proper data distribution for ray-oriented approach very difficult
- Dynamically changing, unpredictable access, fine-grained access
- Better spatial locality on image data than on scene data
  - Strip partition would do better, but less spatial coherence in scene access

Temporal locality

- Working sets much larger and more diffuse than Barnes-Hut
- But still a lot of reuse in modern second-level caches
  - SAS program does not replicate in main memory

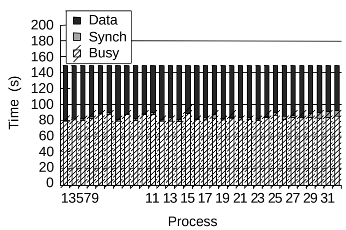
Synchronization:

- One barrier at end, locks on task queues

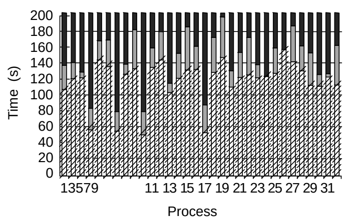
Mapping**: natural to 2-d mesh for image, but likely not important**

Execution Time Breakdown

With task stealing



Without task stealing



- **Task stealing clearly very important for load balance**