Parallel Program Models

- Last Time
  - » Message Passing Model
  - » Message Passing Interface (MPI) Standard
  - » Examples
- Today
  - » Embarrassingly Parallel
  - » Master-Worker
- Reminders/Announcements
  - » Homework #3 is due Wednesday, May 18th at Office Hours; by 5pm
  - » Updated versions of the Tutorial and HW Problem #4 have been posted

Common Parallel Programming Paradigms

- Embarrassingly parallel programs
- Master/Worker programs
- Synchronous: Pipelined and Systolic
- Workflow

Embarrassingly Parallel Computations

- An embarrassingly parallel computation is one that can be divided into completely independent parts that can be executed simultaneously.
  - » (Nearly) embarrassingly parallel computations are those that require results to be distributed, collected and/or combined in some minimal way.
  - » In practice, nearly embarrassingly parallel and embarrassingly parallel computations both called embarrassingly parallel
- Embarrassingly parallel computations have potential to achieve maximal speedup on parallel platforms

Example: the Mandelbrot Computation

- Mandelbrot is an image computing and display computation.
- Pixels of an image (the "mandelbrot set") are stored in a 2D array.
- Each pixel is computed by iterating the complex function

where *c* is the complex number (*a*+*bi*) giving the position of the pixel in the complex plane

$$z_{k+1} = z_k^2 + c$$

Mandelbrot

- Computation of a single pixel:

$$z_{k+1} = z_k^{\ 2} + c$$

$$z_{k+1} = (a_k + b_k i)^2 + (c_{real} + c_{imag}\ i)$$

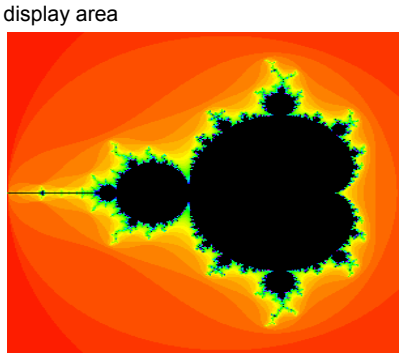
$$= (a_k^{\ 2} - b_k^{\ 2} + c_{real}) + (2 a_k b_k + c_{imag})i$$

- Subscript *k* denotes *k*th interation
- Initial value of *z* is 0, value of *c* is free parameter (position of the point in the complex plane)
- Iterations are continued until the magnitude of z is greater than 2 (which indicates that eventually z will become infinite) or the number of iterations reaches a given threshold.
- The magnitude of *z* is given by

$$z_{length} = \sqrt{a^2 + b^2}$$

Sample Mandelbrot Visualization

- Black points do not go to infinity
- Colors represent "lemniscates" which are basically sets of points which converge at the same rate
- Computation is visualized where pixel color corresponds to the number of iterations required to compute the pixel
  - » Coordinate system of Mandelbrot set is scaled to match the coordinate system of the display area

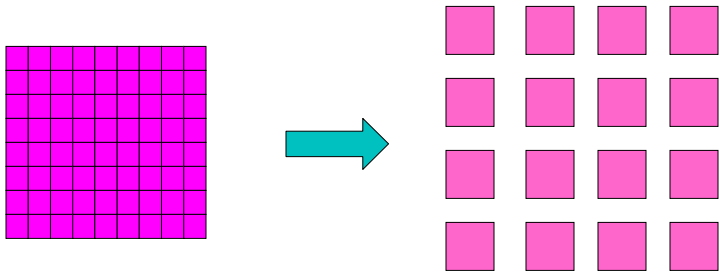


Mandelbrot Parallel Program

- Mandelbrot parallelism comes from **massive data** with the computation is performed across all pixels in parallel
  - » At each point, using different complex numbers *c*.
  - » Different input parameters result in different number of iterations (execution times) for the computation of different pixels.
  - » Embarrassingly Parallel – computation of any two pixels is completely independent.

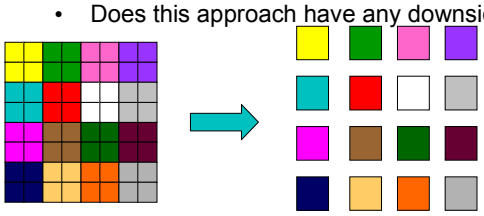
Static Mapping of Mandelbrot

- Organize Pixels into blocks, each block computed by one processor
- Mapping of blocks => processors greatly affects performance
- Ideally, want to load-balance the work across all processors
  - » Problem: Amount of Work is highly variable



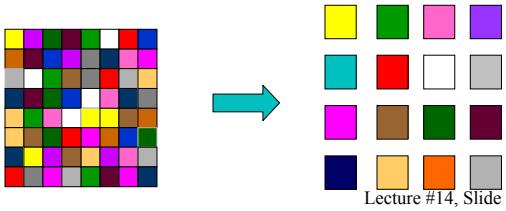
Static Mapping of Mandelbrot

- A Good load-balancing strategy for Mandelbrot is to randomize distribution of pixels
- Does this approach have any downside?



Block decomposition can unbalance load by clustering long-running pixel computations

Randomized decomposition balances load by distributing long-running pixel computations



Other Examples of Static Mapping?

- Sorting: Bucket Sort!
- Jacobi: the simple version for node-parallelism
- Web Search: Histogram Counts

Master-Worker Computations

- Explicit Coordinator for Computation, enables Dynamic Mapping
- Example: A Shared Queue for Work
  - » Master holds and allocates work
  - » Workers perform work
- Typical Master-Worker Interaction
  - » Worker
    - – While there is more work to be done
      - Request work from Master
      - Perform Work
      - (Provide results to Master)
      - (Add more work to the Master's Queue)
  - » Master
    - – While there is more work to be done
      - (Receive results and process)
      - (Receive additional work)
      - Provide work to requesting workers
- Have you seen any examples of this?

Variations on Master-Worker

- Many Variations in Structure
  - » Master can also be a worker
  - » Workers typically do not communicate (star type communication pattern)
  - » Typically a small amount of communication per unit computation
  - » Worker may return "results" to master or may just request more work
  - » Workers may sometimes return additional work (extending the computation) to the Master
- Programming Issues
  - » Master-Worker works best (efficiently) if granularity of tasks assigned to workers amortizes communication between M and W
    - – Computation Time Dominates Communication Time
  - » Speed of worker or execution time of task may warrant non-uniform assignment of tasks to workers
  - » Procedure for determining task assignment should be efficient
- Sound Familiar?

Desktop Grids: The Largest Parallel Systems

SETI HOME 67 TFlops/sec, 500,000 workers, $700,000

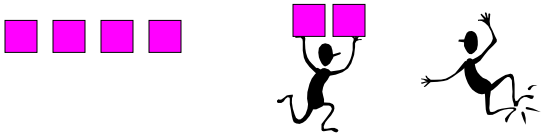
GIMPS 17.5 TFlops/sec, 80,000 workers

Folding@home distributed computing 186 TFlops/sec, 195,000 workers

climateprediction.net

Work-Stealing Variations

- Master/Worker may also be used with "peer to peer" variants which balance the load
- Work-stealing: Idle Processor initiates a redistribution when it needs more to do
  - » Processors A and B perform computation
  - » If B finishes before A, B can ask A for work
- Work-Sharing: Processor initiates a redistribution when it has too much to do
  - » If A's queue gets too larger, it asks B to help



A Massive Master-Worker Computation: MCell

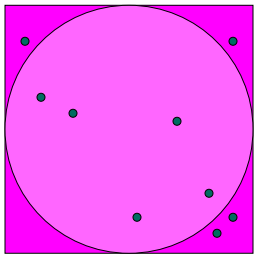
- **MCell** = General simulator for cellular microphysiology
- Uses Monte Carlo diffusion and chemical reaction algorithm in 3D to simulate complex biochemical interactions of molecules
  - Molecular environment represented as 3D space in which trajectories of ligands against cell membranes tracked
- Researchers need huge runs to model entire cells at molecular level.
  - 100,000s of tasks
  - 10s of Gbytes of output data
  - » Will ultimately perform execution-time computational steering , data analysis and visualization

Monte Carlo simulation

- Multiple calculations, each of which utilizes a randomized parameter
- Statistical Sampling to Approximate the Answer
- Widely used to solve Numerical and Physical Simulation Problems

Monte Carlo Calculation of Π

- Monte Carlo method for approximating : $\frac{\pi}{4}$
  - » Randomly choose a sufficient number of points in the square
  - » For each point p, determine if p is in the circle or the square
  - » The ratio of points in the circle to points in the square will provide an approximation of



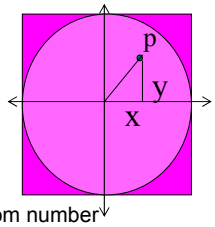
$$\frac{\pi}{4}$$

Master-Worker Monte Carlo Π

- Master:
  - » While there are more points to calculate
    - – (Receive value from worker; update circlesum or boxsum)
    - – Generate a (pseudo-)random value p=(x,y) in the bounding box
    - – Send p to worker
- Worker:
  - » While there are more points to calculate
    - – Receive p from master
    - – Determine if p is in the circle or the square

$$x^2 + y^2 \leq 1$$

    - – Send p's status to master; ask for more work
    - – (realistically do this thousands of times => parallel random number generation challenge)

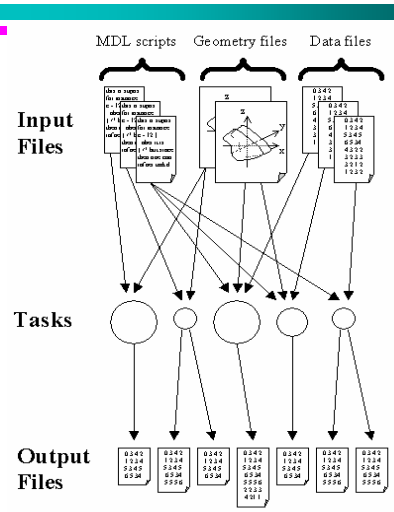


Master-Worker Programming Issues

- How many points should be assigned to a given processor?
  - » Should the initial number be the same as subsequent assignments?
  - » Should the assignment always be the same for each processor? for all processors?
- How long does random number generation, point location calculation, sending, receiving, updating master's sums take?
- What is the right *performance model* for this program on a given platform?

MCell Application Architecture

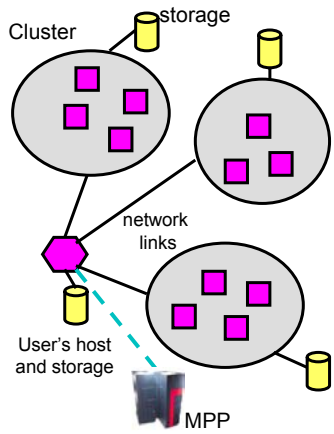
- Monte Carlo simulation performed on large parameter space
- In implementation, parameter sets stored in large shared data files
- Each task implements an "experiment" with a distinct data set
- Produce partial results during large-scale runs and use them to "steer" the simulation



MCell Programming Issues

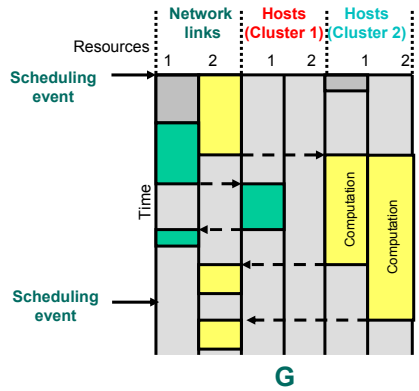
- Monte Carlo simulation can target either Clusters or Desktop Grids
  - » Could even target both if implementation were developed to *co-allocate*
- Although tasks are mutually independent, they share large input files
  - » Cost of moving files can dominate computation time by a large factor
  - » Most efficient approach is to co-locate data and computation
  - » Need intelligent scheduling considering data location in allocation of tasks to processors

Scheduling MCell



Contingency Scheduling Algorithm

- Allocation developed by dynamically generating a Gantt chart for scheduling unassigned tasks between scheduling events
- Basic skeleton
  1. Compute the next scheduling event
  2. Create a Gantt Chart G
  3. For each computation and file transfer currently underway, compute an estimate of its completion time and fill in the corresponding slots in G
  4. Select a subset T of the tasks that have not started execution
  5. **Until each host has been assigned enough work, heuristically assign tasks to hosts, filling in slots in G**
  6. Implement schedule



MCell Scheduling Heuristics

- Many heuristics can be used in the contingency scheduling algorithm
  - » **Min-Min** [task/resource that can complete the earliest is assigned first]

$$\min_i \{\min_j \{predtime(task_i, processor_j)\}\}$$

  - » **Max-Min** [longest of task/earliest resource times assigned first]

$$\max_i \{\min_j \{predtime(task_i, processor_j)\}\}$$

  - » **Sufferage** [task that would "suffer" most if given a poor schedule assigned first]

$$\max_{i,j} \{predtime(task_i, processor_j)\} - next \max_{i,j} \{predtime(task_i, processor_j)\}$$

  - » **Extended Sufferage** [minimal completion times computed for task on each cluster, sufferage heuristic applied to these]

$$\max_{i,j} \{predtime(task_i, cluster_j)\} - next \max_{i,j} \{predtime(task_i, cluster_j)\}$$

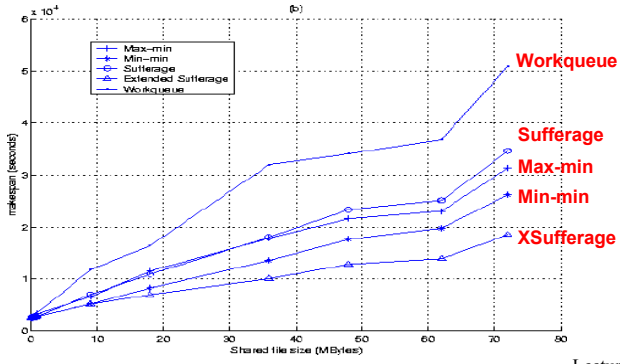
  - » **Workqueue** [randomly chosen task assigned first]

Which heuristic is best?

- How sensitive are the scheduling heuristics to the location of shared input files and cost of data transmission?
- Used the contingency scheduling algorithm to compare
  - » Min-min
  - » Max-min
  - » Sufferage
  - » Extended Sufferage
  - » Workqueue
- Ran the contingency scheduling algorithm on a simulator which reproduced file sizes and task run-times of real MCell runs.

MCell Simulation Results

- Comparison of the performance of scheduling heuristics when it is up to 40 times more expensive to send a shared file across the network than it is to compute a task
- "Extended sufferage" scheduling heuristic takes advantage of file sharing to achieve good application performance



Summary

- Embarrassingly Parallel Applications
  - » Static Mapping
  - » Randomized Mapping
- Master-Worker
  - » Flexible Implementations of Dynamic Mapping
  - » Communication: Work and Results
  - » Star Type Communication
- MCELL Example
  - » Monte Carlo Simulation
  - » Complex Scheduling Heuristics Embedded in Master-Worker