Parallel Computing
Slides from Prof. Jeffrey Hollingsworth
What is Parallel Computing?

● **Does it include:**
  – super-scalar processing (more than one instruction at once)?
  – client/server computing?
    • what if RPC calls are non-blocking?
  – vector processing (same instruction to several values)?
  – collection of PC’s **not** connected to a network?

● **For us, parallel computing requires:**
  – more than one processing element
  – nodes connected to a communication network
  – nodes working together to solve a single problem
Why Parallelism

- **Speed**
  - need to get results faster than possible with sequential
    - a weather forecast that is late is useless
  - could come from
    - more processing elements (P.E.)
    - more memory (or cache)
    - more disks

- **Cost: cheaper to buy many smaller machines**
  - this is only recently true due to
    - VLSI
    - commodity parts
What Does a Parallel Computer Look Like?

- **Hardware**
  - processors
  - communication
  - memory
  - coordination

- **Software**
  - programming model
  - communication libraries
  - operating system
Processing Elements (PE)

- **Key Processor Choices**
  - How many?
  - How powerful?
  - Custom or off-the-shelf?

- **Major Styles of Parallel Computing**
  - SIMD - Single Instruction Multiple Data
    - one master program counter (PC)
  - MIMD - Multiple Instruction Multiple Data
    - separate code for each processor
  - SPMD - Single Program Multiple Data
    - same code on each processor, separate PC’s on each
  - Dataflow - instruction waits for operands
    - “automatically” finds parallelism
MIMD

Processors

Program Counter
Program #1

Program Counter
Program #2

Program Counter
Program #3
SPMD

Processors

Program Counter

Program

Program Counter

Program

Program Counter

Program

Program
Dataflow

instruction

I4
Communication Networks

- **Connect**
  - PE’s, memory, I/O

- **Key Performance Issues**
  - latency: time for first byte
  - throughput: average bytes/second

- **Possible Topologies**
  - bus - simple, but doesn’t scale
  - ring - orders delivery of messages
Topologies (cont)

- tree - needs to increase bandwidth near the top

- mesh - two or three dimensions

- hypercube - needs a power of number of nodes
Communication Networks

- **Connect**
  - PE’s, memory, I/O

- **Key Performance Issues**
  - latency: time for first byte
  - throughput: average bytes/second

- **Possible Topologies**
  - bus - simple, but doesn’t scale
  - ring - orders delivery of messages
Topologies (cont)

– tree - needs to increase bandwidth near the top

– mesh - two or three dimensions

– hypercube - needs a power of number of nodes
Memory Systems

- **Key Performance Issues**
  - latency: time for first byte
  - throughput: average bytes/second

- **Design Issues**
  - Where is the memory
    - divided among each node
    - centrally located (on communication network)
  - Access by processors
    - can all processors get to all memory?
    - is the access time uniform?
Coordination

- **Synchronization**
  - protection of a single object (locks)
  - coordination of processors (barriers)

- **Size of a unit of work by a processor**
  - need to manage two issues
    - load balance - processors have equal work
    - coordination overhead - communication and sync.
  - often called “grain” size - large grain vs. fine grain
Sources of Parallelism

- **Statements**
  - called “control parallel”
  - can perform a series of steps in parallel

- **Loops**
  - called “data parallel”
  - most common source of parallelism
  - each processor gets one (or more) iterations to perform
Example of Parallelism

- **Easy (embarrassingly parallel)**
  - multiple independent jobs (i.e..., different simulations)
- **Scientific**
  - Largest users of parallel computing
  - dense linear algebra (divide up matrix)
  - physical system simulations (divide physical space)
- **Databases**
  - biggest commercial success of parallel computing (divide tuples)
    - exploits semantics of relational calculus
- **AI**
  - search problems (divide search space)
  - pattern recognition and image processing (divide image)
Metrics in Application Performance

- **Speedup (often call strong scaling)**
  - ratio of time on n nodes to time on a single node
  - hold problem size fixed
  - should really compare to best serial time
  - goal is linear speedup
  - super-linear speedup is possible due to:
    - adding more memory
    - search problems

- **Weak Scaling (also called Iso-Speedup)**
  - scale data size up with number of nodes
  - goal is a flat horizontal curve

- **Amdahl's Law**
  - max speedup is $1/(\text{serial fraction of time})$

- **Computation to Communication Ratio**
  - goal is to maximize this ratio
Metrics in Application Performance

- **Speedup**
  - ratio of time on n nodes to time on a single node
  - hold problem size fixed
  - should really compare to best serial time
  - goal is linear speedup
  - super-linear speedup is possible due to:
    - adding more memory
    - search problems

- **Iso-Speedup**
  - scale data size up with number of nodes
  - goal is a flat horizontal curve

- **Amdahl's Law**
  - max speedup is 1/(serial fraction of time)

- **Computation to Communication Ratio**
  - goal is to maximize this ratio
How to Write Parallel Programs

- **Use old serial code**
  - compiler converts it to parallel
  - called the dusty deck problem

- **Serial Language plus Communication Library**
  - no compiler changes required!
  - PVM and MPI use this approach

- **New language for parallel computing**
  - requires all code to be re-written
  - hard to create a language that provides performance on different platforms

- **Hybrid Approach**
  - HPF - add data distribution commands to code
  - add parallel loops and synchronization operations
Application Example - Weather

- **Typical of many scientific codes**
  - computes results for three dimensional space
  - compute results at multiple time steps
  - uses equations to describe physics/chemistry of the problem
  - grids are used to discretize continuous space
    - granularity of grids is important to speed/accuracy

- **Simplifications (for example, not in real code)**
  - earth is flat (no mountains)
  - earth is round (poles are really flat, earth buldges at equator)
  - second order properties
Grid Points

- **Divide Continuous space into discrete parts**
  - for this code, grid size is fixed and uniform
    - possible to change grid size or use multiple grids
  - use three grids
    - two for latitude and longitude
    - one for elevation
    - Total of $M \times N \times L$ points

- **Design Choice: where is the grid point?**
  - left, right, or center of the grid
    - in multiple dimensions this multiples:
      - for 3 dimensions have 27 possible points
Variables

- **One dimensional**
  - m - geo-potential (gravitational effects)

- **Two dimensional**
  - pi - “shifted” surface pressure
  - sigmadot - vertical component of the wind velocity

- **Three dimensional (primary variables)**
  - \(<u,v>\) - wind velocity/direction vector
  - T - temperature
  - q - specific humidity
  - p - pressure

- **Not included**
  - clouds
  - precipitation
  - can be derived from others
Serial Computation

- Convert equations to discrete form
- Update from time $t$ to $t + \delta t$

```plaintext
foreach longitude, latitude, altitude
    $ustar[i,j,k] = n \cdot pi[i,j] \cdot u[i,j,k]$
    $vstar[i,j,k] = m[j] \cdot pi[i,j] \cdot v[i,j,k]$
    $sdot[i,j,k] = pi[i,j] \cdot sigmadot[i,j]$
end

foreach longitude, latitude, altitude
    $D = 4 \cdot ((ustar[i,j,k] + ustar[i-1,j,k]) \cdot (q[i,j,k] + q[i-1,j,k]) +$
    terms in $\{i,j,k\}{+,-}\{1,2\}$
    $piq[i,j,k] = piq[i,j,k] + D \cdot delat$
    similar terms for $piu, piv, piT,$ and $pi$
end

foreach longitude, latitude, altitude
    $q[i,j,k] = piq[i,j,k]/pi[i,j,k]$
    $u[i,j,k] = piu[i,j,k]/pi[i,j,k]$
    $v[i,j,k] = piv[i,j,k]/pi[i,j,k]$
    $T[i,j,k] = piT[i,j,k]/pi[i,j,k]$
end
```
Shared Memory Version

- in each loop nest, iterations are independent
- use a parallel for-loop for each loop nest
- synchronize (barrier) after each loop nest
  - this is overly conservative, but works
  - could use a single sync variable per item, but would incur excessive overhead
- potential parallelism is $M \times N \times L$
- private variables: $D, i, j, k$
- Advantages of shared memory
  - easier to get something working (ignoring performance)
- Hard to debug
  - other processors can modify shared data
Distributed Memory Weather

- decompose data to specific processors
  - assign a cube to each processor
    - maximize volume to surface ratio
    - minimizes communication/computation ratio
  - called a <block,block,block> distribution

- need to communicate \{i,j,k\}\{+,-\}\{1,2\} terms at boundaries
  - use send/receive to move the data
  - no need for barriers, send/receive operations provide sync
    - sends earlier in computation too hide communication time

- Advantages
  - easier to debug?
  - consider data locality explicitly with data decomposition

- Problems
  - harder to get the code running
Ensuring a fair speedup

- $T_{\text{serial}} = \text{faster of}$
  - best known serial algorithm
  - simulation of parallel computation
    - use parallel algorithm
    - run all processes on one processor
  - parallel algorithm run on one processor

- If it appears to be super-linear
  - check for memory hierarchy
    - increased cache or real memory may be reason
  - verify order operations is the same in parallel and serial cases