CMSC 714
High Performance Computing
Lecture 2 - Introduction
http://www.cs.umd.edu/class/spring2017/cmsc714
Alan Sussman

Notes
- Accounts handed out and first assignment probably late next week

Last time
- Why parallel computing?
  - speed, cost
- Parallel computing basics
  - Processing elements, memory, network, disks
  - SIMD, MIMD, SPMD, dataflow
  - networks
    - bus, ring, tree, mesh (2D or 3D), hypercube
  - memory
    - latency and throughput (bandwidth)
    - shared vs. distributed (physically and logically)
    - UMA vs. NUMA

Coordination
- Since parallelism in our view is processors working together to solve a problem
- Synchronization
  - protection of a single object (e.g., locks)
  - coordination of processors (e.g., 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 synchronization
  - often called "grain" size - coarse grain vs. fine grain
SOURCES OF PARALLELISM

Sources of Parallelism

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

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

Examples of Parallelism

- **Easy (embarrassingly parallel)**
  - multiple independent jobs (i.e., different simulations)

- **Scientific**
  - dense linear algebra (divide up matrix)
  - physical system simulations (divide physical space)

- **Databases**
  - biggest success of parallel computing (divide tuples)
    - exploits semantics of relational algebra

- **AI**
  - search problems (divide search space)
  - pattern recognition and image processing (divide image)

Metrics in Application Performance

- **Speedup**
  - ratio of time on one node to time on n nodes
  - 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/cache
    - search problems

- **Iso-Speedup (or scaled 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), or
  - $1 / (1 - f + f/s)$ as $s \to \infty$

- **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 high performance on different platforms
- **Hybrid Approach – old language(s), new constructs**
  - 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 bulges 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 dimensional grid
    - two for latitude and longitude
    - one for elevation
    - Total of M * N * L points
- **Design Choice: where is the grid point?**
  - left, right, or center of the interval for a grid element
  - in multiple dimensions this multiplies:
    - for 3 dimensions have 27 possible positions

Variables

- **One dimensional**
  - m - geo-potential (gravitational effects)
- **Two dimensional**
  - pl - "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 * pi[i,j] * u[i,j,k]
vstar[i,j,k] = m[j] * pi[i,j] * v[i,j,k]
sdot[i,j,k] = pi[i,j] * sigmadot[i,j]
end
```

```plaintext
D = 4 * ((ustar[i,j,k] + ustar[i-1,j,k]) * (q[i,j,k] + q[i-1,j,k]) + terms in \{i,j,k\}\{+,-\}\{1,2\})
pqi[i,j,k] = pqi[i,j,k] + D * delat
similar terms for pju, piv, piT, and pi
end
```

- End of loop over longitude, latitude, altitude

```plaintext
q[i,j,k] = pqi[i,j,k]/pi[i,j,k]
u[i,j,k] = piju[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 element, 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 Version**

- decompose data to specific processors
  - assign a cube to each processor
    - maximize volume to surface ratio
    - which 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
    - do sends earlier in computation to hide communication time
- **Advantages**
  - easier to debug? maybe
  - consider data locality explicitly with data decomposition
    - better performance/scaling
- **Problems**
  - harder to get the code running

**Database Applications**

- Too much data to fit in memory (or sometimes disk)
  - data mining applications (K-Mart had a 4-5TB database several years ago)
  - imaging applications (NASA and others have sites with multiple petabytes)
    - use a fork lift to load tapes by the pallet
- **Sources of parallelism**
  - within a large transaction
  - among multiple transactions
- **Join operation**
  - form a single table from two tables based on a common field
    - try to split join attribute into disjoint buckets
      - if know data distribution is uniform its easy
      - if not, try hashing
Parallel Search (TSP)

- may appear to be faster than 1/n
  - but this is not really the case either

- Algorithm
  - compute a path on a processor
    - if our path is shorter than the shortest one, send it to the others.
    - stop searching a path when it is longer than the shortest.
  - before computing next path, check for word of a new min path
  - stop when all paths have been explored.

- Why it appears to be faster than 1/n speedup
  - we found the path that was shorter sooner
  - however, the reason for this is a different search order!

Ensuring a fair speedup

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

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