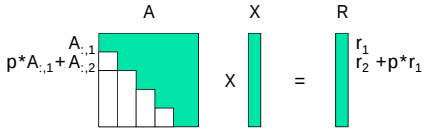
Parallel Programming

Kai Shen

Gaussian Elimination: Dependencies

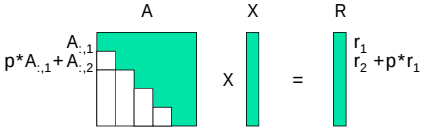
- Simplification – ignore pivoting and final solving step
- Reduce an equation matrix into an equivalent upper-diagonal




```
for i=1 to N
   for j=i+1 to N
      zero out A_{i,j} by adding p*A_{:,i} to A_{:,j}
```

- Dependencies:
  - Outer loop instances (e.g., i=2 depends on i=1)?
  - Inner loop instances (e.g., j=3 depends on j=2)?
  - Within one inner loop instance?

Gaussian Elimination: Task Decomposition

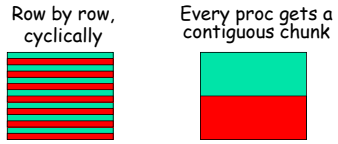



```
for i=1 to N
   for j=i+1 to N
      zero out A_{i,j} by adding p*A_{:,i} to A_{:,j}
```

- Parallelism:
  - Inner loop instances (row-wise)
  - Within one inner loop instance (column-wise)
- Task decomposition:
  - Row vs. column partitioning?
  - 2-dimensional partitioning

Gaussian Elimination: Task Decomposition

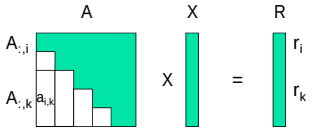
- Row partitioning at different granularities



- Better load balancing under cyclic partitioning

Pivoting

- Reduce an equation matrix into an equivalent upper-diagonal




```
for i=1 to N
    swap A:,i (ri) with A:,k (rk) where ai,k has largest absolute value
    for j=i+1 to N
        zero out Ai,j by adding p*A:,i to A:,j
```

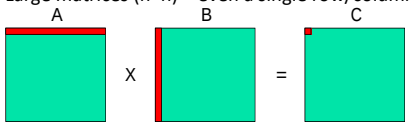
- How does it complicate the parallelization (particularly row vs. column partitioning)?

Dynamic Task Assignment

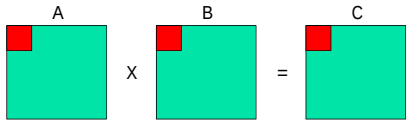
- How does it work?
  - Maintain a centralized queue of ready tasks, protected by synchronization primitives like mutex lock
  - Each thread grabs a task at the beginning; grabs another task after completing the current one
  - New tasks may be generated on the fly and added to queue
- Advantage(s) and disadvantage(s)
- Does it help Gaussian Elimination with pivoting?

Block Computation

- Matrix multiplication (sequential)
  - Large matrices (n×n) – even a single row/column won't fit in cache

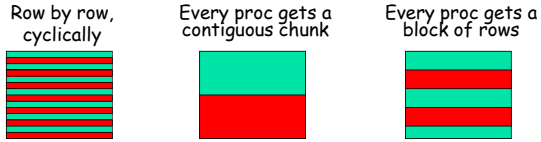


  - How many times each element of A,B has to be loaded?
- Block-by-block multiplication
  - Three blocks (b×b) fit into the cache at the same time



Gaussian Elimination: Task Decomposition

- Row partitioning at different granularities



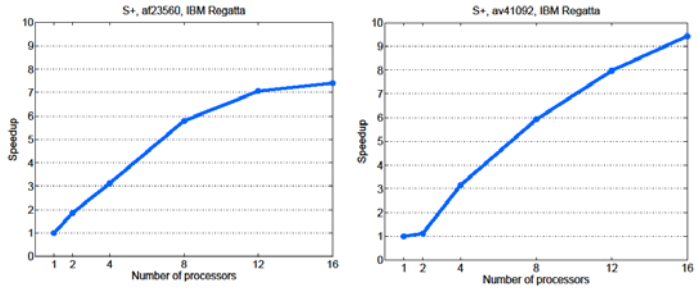
- A good block size to efficiently utilize processor cache, you produce reasonable load balancing

Irregular Parallelism

- Real problems contain large, sparse matrices
- Solve them as dense matrices waste time on zero-element operations
- Sparse matrix computation
  - Load imbalance (now you can really benefit from dynamic task assignment)
  - Managing nonzero fillins (hard to do block operations and utilize cache effectively)

Example Speedup Results

- The speed ratio over the best sequential run



Alternative Methods for Solving Linear Equations

- Faster solutions if we tolerate less precision
  - Less-than-precise pivoting
  - Iterative method

Assignment #2 Shared Memory Parallel Programming

- Pre-assignment
  - We provide you sequential/parallel SOR code (a different version, called red-black ordering)
  - You read, understand, and run it
- Main assignment
  - Parallel Gaussian Elimination (with pivoting, but without block computation or sparse computation)
  - Different settings (input matrices, machines, proc #'s)
  - Analysis and comparison (Row/column/2D partitioning, blocking sizes, static vs. dynamic task assignment)
  - Grading based on your written report!

Assignment #2

- All should have accounts in CS grad/research network
- Familiarize with the parallel machines
  - Multi-chip/socket
  - Multi-core
  - Hardware threading (or Intel hyperthreads)

Suggestions on Testing

- Incremental testing:
  - Try an artificial, small, dense input matrix first
  - Try to parallelize without partial pivoting first

Pthreads Synchronization Primitives

- Mutex lock (mutual exclusion)
  - pthread_mutex_lock
  - pthread_mutex_unlock
- Condition variable (waiting for a condition)
  - pthread_cond_wait
  - pthread_cond_signal
  - pthread_cond_broadcast

Barrier Synchronization

- A barrier is set for all threads
- A thread can proceed beyond its barrier if and only if all threads have reached respective barrier points
- One way to implement the barrier:
  - Count the number of arrivals at the barrier
  - When a thread arrives, wait if this is not the last one, otherwise unlock everyone else and proceed
  - Mutex lock to protect the arrival number, condition variable to implement wait and broadcast