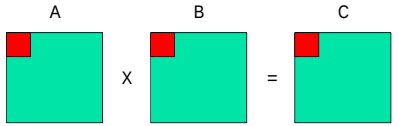
Parallel Programming

Kai Shen

A Sidetrack

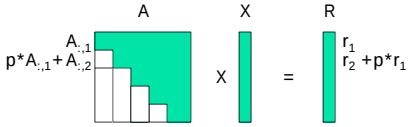
- Matrix multiplication



- Efficient use of cache
- Block-by-block multiplication

Gaussian Elimination: Dependencies

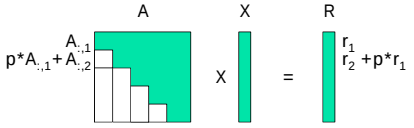
- Simplification – ignore pivoting and final solving step
- Reduce an equation matrix into an equivalent upper-diagonal



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, column, 2-dimensional

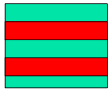
Gaussian Elimination: Partitioning Granularity

- Row partitioning at different granularities

Row by row, cyclically

Every proc gets a contiguous chunk

Every proc gets a block of rows



- A good block size to efficiently utilize processor cache

Comparison of Alternative Approaches

- Row partitioning, column partitioning, 2-dimensional partitioning
- Blocking at different sizes
- Static vs. dynamic task assignment
- … …

Suggestions on Testing

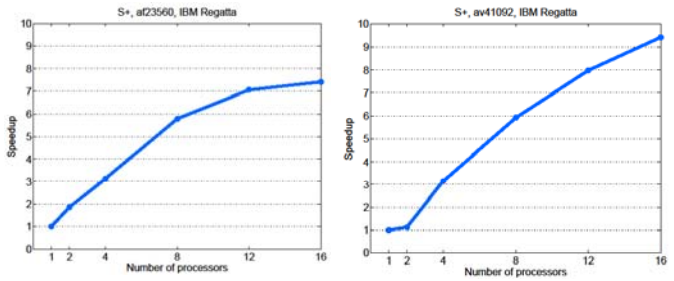
- Incremental testing:
  - Try an artificial, small, dense input matrix first
  - Try to parallelize without partial pivoting first

Irregular Parallelism

- Real problems contain large, sparse matrices
- Solve them as dense matrices waste time on zero-element operations
- Sparse matrix computation
  - Load imbalance
  - Managing nonzero fillins

Example Speedup Results

- The speed ratio over the best sequential run



Alternative Methods for Solving Linear Equations

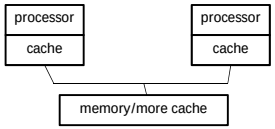
- Faster solutions if we tolerate less precision
  - Less-than-precise pivoting
  - Iterative method

Instructional-Level Parallelism

- Instructional-level parallelism
  - A single CPU core, but multiple functional units (arithmetic computation, floating-point computation, …)
  - Fetch/decode multiple instructions at a time and execute them in parallel
  - ⇨ possibly retire multiple instructions in a cycle
- Manage control dependencies
  - Stop before branch
  - Speculative execution on branch but rollback if mis-speculate
- Manage data dependencies
  - Recognize and observe dependencies
  - Resolve artificial dependencies through renaming

Cache Coherence Problem

- In shared memory multiprocessors (except multithreading), each processor has a local cache



- For each data item in memory, additional copies may exist in processor local caches
  - after one processor updates the data, another processor's local copy may be incoherent
  - in cases of write-through and write-back caches

Cache Coherence

- Coherence means the system semantics is the same as that of a system without multiple processor-local caches
- Multiprocessor cache coherent if there exists a hypothetical sequential order of all operations for each data location:
  - returned value in the read operation is that written by last write in the sequential order
  - the sequential order matches the order of operations from each processor

Sequential Memory Consistency

- It means the memory semantics is the same as that of a uni-processor system
- Sequential consistent if there exists a hypothetical sequential order of all operations on all locations:
  - returned value in the read operation is that written by last write in the sequential order
  - the sequential order matches the order of operations from each processor