ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Winter 2010

Dense matrix problems

- we'll take a look at
  - inner and outer products of two vector
  - matrix-vector and matrix-matrix multiplication
  - LU and Cholesky decompositions
  - QR factorizations
  - computing eigenvalues and eigenvectors
  - fast Fourier transforms
- when solving a problem of size n on p processors, we will assume
  - that p, and occasionally $\sqrt{p}$ divides n
  - that p is a perfect square, when forming two-dimensional process grids
  - matrices are $n \times n$ – square, not rectangular
  - we are memory constrained and data replication must be minimized
- these problems have been studied extensively and form the core of scientific computing on parallel machines
  - excellent implementations available
  - interest has been revived due to the expected disruption by multi-core architectures

Vector inner product

- the inner product of two n-vectors x, y is given by

$$x^T y = \sum_{i=1}^{n} x_i y_i$$

  which requires n multiplications and $n - 1$ additions
- parallelization strategy:
  - n fine grain tasks, numbered $i = 1, \ldots, n$, that store x_i and y_i, and compute $x_i y_i$
  - communication is a sum reduction over n fine grain tasks
  - coarsening is achieved by coalescing n/p tasks together, assuming that each process can accommodate the data storage requirements
  - and mapping each coarse grain task to a process

Vector outer product

- the outer product of two n-vectors x and y is the $n \times n$ matrix A given by

  $$A_{ij} = x_i y_j$$

  which requires n^2 multiplications
- parallelization strategies are determined by the storage requirements
  - build a two-dimensional grid of n^2 fine grain tasks numbered (i, j), with $i, j = 1, \ldots, n$; each one computes $x_i y_j$
  - assuming no data replication is allowed
    - let task $(i, 1)$ store x_i and task $(1, i)$ store y_i
    - or, let task (i, i) store both x_i and y_i
  - either way, the task that owns each element must broadcast it to the other tasks: x_i along the i^{th} task row, y_j along the j^{th} task column
  - coarsening to p tasks can be accomplished by
    - combining n/p rows or columns
    - forming $(n/\sqrt{p}) \times (n/\sqrt{p})$ grid of fine grain tasks
  - and each coarse grain task can be assigned to a process
- either way, naïve broadcasting of the components of x and y would require as much total memory as replication
  - storage can be reduced by circulating portions of x and y through the tasks, with each task using the available portion and passing it on

The product of a matrix with a vector

- given an $n \times n$ matrix A and an n-vector x, the matrix vector product yields an n-vector y whose components are given by

$$y_i = \sum_{j=1}^{n} A_{ij} x_j$$

  requiring a total of n^2 multiply-add operations
- once again, the parallelization strategy is determined by how the data is distributed among fine grain tasks
  - build a two-dimensional grid of n^2 fine grain tasks numbered (i, j), with $i, j = 1, \ldots, n$; each one computes $A_{ij} x_j$
  - task (i, j) has $a_{i,j}$, but if no data replication is allowed
    - let task $(i, 1)$ store x_i and task $(1, i)$ store y_i
    - or, let task (i, i) store both x_i and y_i
  - the task that owns x_j must broadcast it along the j^{th} task row, and y_i is formed by sum reduction along the i^{th} task column
  - coarsening into p tasks can be accomplished by combining n/p rows/columns, or by forming $(n/\sqrt{p}) \times (n/\sqrt{p})$ blocks
  - and each coarse grain task can be assigned to a process

Coarsening along rows or columns

- for one-dimensional coarsening into n/p task rows
  - if x is stored in one task, it must be broadcast to all others
  - if x is distributed among tasks, with n/p components per task, then multiple broadcasts are required
  - each task computes the inner product of its n/p rows of A with the entire x to produce n/p components of y
- for one-dimensional coarsening into n/p task columns
  - n/p components of x are distributed among the tasks
  - each task computes the linear combination of its n/p columns with coefficients from its copy of x
  - since the right parts of x are already available, no communication is required
  - y is generated by a sum reduction across tasks
- these two are *duals* of each other
  - row coarsening begins with broadcast, followed by communication-free inner products
  - column coarsening begins with communication-free linear combinations, follows by a reduction

Two dimensional coarsening

- for two dimensional coarsening, we form $(n/\sqrt{p}) \times (n/\sqrt{p})$ blocks of fine grain task
  - each one holding a $(n/\sqrt{p}) \times (n/\sqrt{p})$ block of A
  - with components of x distributed either across one task row, or along the diagonal, n/p components per task

s