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 factorization and Cholesky decomposition
  - QR factorization
  - 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

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

  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

$$A_{ij} = x_i y_j \tag{2}$$

  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

$$y_i = \sum_{j=1}^{n} A_{ij} x_j \tag{3}$$

  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_{ij}, 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
- the algorithm combines the features of row/column coarsening
  - components of x are broadcast along task columns
  - each task performs n^2/p multiplications locally and sums $n/\sqrt{p}$ sets of products
  - sum reductions along task rows produce the components of y by combining the component products

Matrix multiplication

- the product of two $n \times n$ matrices A and B is an $n \times n$ matrix C given by

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \tag{4}$$

  where each one of n^2 entries requires n multiply-adds for a total of n^3 operations
- matrix multiplication can be viewed as
  - n^2 inner products
  - the sum of n outer products
  - n matrix vector products
- each one produces a parallel algorithm for matrix multiplication
- but we'll explore a direct solution instead

Partitioning and communication patterns

- we build a three dimensional array of n^3 fine grain tasks
  - with $i, j, k = 1, \ldots, n$, let task (i, j, k) be responsible for computing the product $A_{ij}B_{jk}$
  - assuming no data replication, we have to distribute the data for A and B among $2n^2$ tasks
  - suppose that task (i, j, j) holds A_{ij} and task (i, j, i) holds B_{ij}
  - we will refer to tasks along i and j as task rows and columns
  - and tasks along k as *layers*
- the communication requirements among tasks are satisfied if we
  - broadcast the entries of the k^{th} column of A from task (i, j, j) to each task row in the k^{th} layer
  - broadcast the entries of the k^{th} row of B from task (i, j, i) to each task column of the k^{th} layer
  - form the result C_{ij} by the sum reduction of the values held by all the tasks layers k

Coarsening

- there are four natural ways to coarsen our $n \times n \times n$ fine grain tasks into p coarse grain tasks
  - by task rows: combine the $(n/p) \times n \times n$ tasks along a given task row
  - by task columns: combine the $n \times (n/p) \times n$ tasks along a given task column
  - partition the layers in a two dimensional grid by combining $(n/\sqrt{p}) \times (n/\sqrt{p}) \times n$ fine grain tasks
  - using three dimensional blocks by combining $(n/\sqrt[3]{p}) \times (n/\sqrt[3]{p}) \times (n/\sqrt[3]{p})$ tasks
- the two one dimensional coarsening strategies are similar
  - for row coarsening
    - each task needs only the part of A it already has, but needs all B entries
    - so global communication is required to broadcast the n^2/p entries of B held by each task
  - conversely, for column coarsening
    - each task needs only the parts of B that it already has, but it needs all of A
    - so global communication is required to broadcast the n^2/p entries of A held by each task
  - if accumulating A or B on each processor is not feasible, tasks can circulate portions of the array in a ring

Coarsening using a two dimensional grid

- block matrix multiplication has the same overall form as actual product, with scalar operations replaced by the matrix product of blocks!
- you should verify that

$$C_{ij} = \sum_{k=1}^{\sqrt{p}} A_{ik} B_{kj} \tag{5}$$

  for $i, j = 1, \ldots, \sqrt{p}$
- assume that task (i, j) has local access to block A_{ij} and B_{ij} and computes block C_{ij} of the result
- this requires all blocks A_{ik} and B_{kj} for $k = 1, \ldots, \sqrt{p}$ to be communicated
  - first, a global broadcast of A blocks across each task row
  - followed by a global broadcast of B blocks across each task column
- memory requirements can be addressed by either of the following:
  - broadcast blocks of A across rows while circulating blocks of B across columns in lock step, so that they arrive at a given task at the same time
  - circulate blocks of A horizontally and blocks of B vertically, after an initial circular shift, so that blocks meet at a given task at the right time