

Lecture 10: February 27

Lecturer: Jim Demmel

Scribes: Scott Beamer

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

This is the first lecture about dense linear algebra. The slides of these two lectures are in:
http://www.cs.berkeley.edu/~demmel/cs267_Spr08

10.1 Dense Linear Algebra

10.1.1 BLAS

Basic Linear Algebra Subroutines (BLAS) are used as building blocks for making larger functions. They are broken into 3 levels based on computational intensity ($q = \frac{\#flops}{\#mem\ ref.}$). The higher levels were developed later, and use more locality to get more computation done on the same amount of data.

- **BLAS Level 1** - example: dot product - $q = \frac{2n}{3n} = \frac{2}{3}$
- **BLAS Level 2** - example: matrix-vector multiplication - $q = \frac{2n^2}{n^2} = 2$
- **BLAS Level 3** - example: matrix-matrix multiplication - $q = \frac{4n^2}{2n^3} = \frac{n}{2}$

10.1.2 Matrix Layout for Parallel Computation

To do operations efficiently in parallel, the matrix of size $N \times N$ needs to be partitioned among the P processors.

- **1D Column Blocked Layout:** Each processor gets $\frac{N}{P}$ consecutive columns. Flaw: many algorithms (such as GE) focus on the front half so this gets poor load balance.
- **1D Column Cyclic Layout:** Each processor gets given $\frac{N}{P}$ distributed cyclically, i.e. processor k gets all columns x s.t. $x \bmod P = k$. Flaw: this gets less locality locally.
- **1D Column Block Cyclic Layout:** Like 1D Column Cyclic Layout, but each processor gets b consecutive columns to increase locality - Flaw: poor parallelism for row-centric algorithms.
- **2D Row and Column Blocked Layout:** The matrix is broken into a grid of P rectangles, and each processor gets one. Flaw: suffers from same problem as other layouts that aren't block cyclic.
- **2D Row and Column Block Cyclic Layout:** Like 1D Column Block Cyclic Layout, but done in 2D. This can actually be seen as a generalization of all the other layouts since it can produce the same layout depending on parameters.

10.1.3 Parallel Matrix-Vector Product

Let us consider problems of the form $y = y + Ax$, where A is a dense matrix. Lets use this as a case study to try out different layouts.

- **[Blocked] Row Layout:** This will achieve good load balance, but x and y will need to be broadcast to all processors.
- **[Blocked] Column Layout:** This also achieves decent load balance, but after computation the result will need to be reduced over all the processors.
- **[Blocked] 2D Layout:** This will need both a broadcast and a reduction, but this can perform well if well balanced.

10.1.4 Parallel Matrix Multiply

10.1.4.1 Background

We want to compute $C = C + AB$, where A , B , and C are all dense matrices. Using the basic naive algorithm (not Strassen) this will require $2n^3$ flops. When parallelizing this, the variables we have to work with are: data layout, machine topology, and scheduling. To model messages, let α be the fixed overhead, β be the time to transmit one word, and n be the number of words. Thus the message latency is $\alpha + n\beta$. To measure how well we are doing, we will use efficiency = $\frac{\text{time}(\text{serial})}{p \cdot \text{time}(\text{parallel})}$, so a perfect (linear) speedup is when efficiency = 1.

10.1.4.2 Serial Analysis

Although the computation of $C += AB$ can be easily expressed as $C_{ij} += \sum_k A_{jk} B_{ki}$, the order in which the loop variables are used has an affect. Consider the standard i - j - k ordering:

```

for i
  for j
    for k
      C_{ji} += A_{jk} B_{ki}

```

The innermost loop is a dot product (BLAS 1), and the middle loop is a matrix-vector multiply of A times row i (BLAS 2). Compare this to the k - i - j loop ordering:

```

for k
  for i
    for j
      C_{ji} += A_{jk} B_{ki}

```

The innermost loop here is a saxpy (BLAS 1), and the middle loop is a rank 1 update (BLAS 2). All of these could of course be generalized to blocks.

10.1.4.3 1D Column Layout

Assume the matrix is of size $N \times N$ which is divisible by P (number of processors). Computing the product on a system on a bus without broadcast ends up being slower than the serial implementation, because each pair of processors must communicate, but only one pair can communicate at a time.

With nearest neighbor communication (or at least ring or bus with broadcast), a good speedup of $\frac{1}{1+O(\frac{P}{N})}$ can be achieved, because pairs can communicate simultaneously.

10.1.4.4 Cannon's Algorithm

Cannon's algorithm uses a square 2D decomposition. Each processor gets a square of the grid, and the algorithm works by carefully done shifts. Each cycle, the processor is presented with the correct block of A and the correct block of B to multiply and add onto its block of C . After the initial shift, each cycle the blocks of A are circularly shifted left, and the blocks of B are circularly shift up by one cell. These shifts are efficient because each processor only has to send one block and receive one block, all the data it receives is useful, and there is no redundant work. The initial shift correctly skews the matrices by shifting each row i of A left by i cells (or each column j of B up by j cells).

Cannon is able to achieve an impressive efficiency of $\frac{1}{1+O(\frac{\sqrt{P}}{N})}$, but it has some limitations. Most of its limitations make it harder to generalize for: non-square matrix dimensions, non-square number of processors, or more complicated matrix layouts.

10.1.4.5 SUMMA

SUMMA (Scalable Universal Matrix Multiply) is slightly less efficient than Cannon's but is used more often because it is simpler and easier to generalize. The matrix is broken into a 2D grid, and each processor is assigned a square. SUMMA works by broadcasting chunks of blocks across rows or columns. A is broken into thin columns of width b and B is broken up into thin rows of height b . In a big loop, each iteration one column from A is considered and one row from B is considered. If the strip crosses a processor's area, it broadcasts the intersecting portion to the other processors in its row if its A (or the other processors in its column if its B). Each iteration each processor receives one row broadcast and one column broadcast (unless it is the one broadcasting) and multiplies these and adds it onto its portion of C .

The performance of SUMMA is decent, and can be tuned by adjusting b . When b is smaller, less memory is used but the algorithm is less efficient, so when b is greater, more memory is used but at higher efficiency.

10.1.5 Recursive Layouts

Recursive layouts are an area of open research that hopes to help with both cache hierarchies and parallelism. Fortunately, many of them can be transparently implemented as a permutation, and do not affect the user's view of the layout. The goal is to make any piece of the matrix near any other given piece of the matrix.