Block matrices and partitioned algorithms

Numerical Linear Algebra

Suppose we want to solve a linear system $Ax = b$, where A is large (say $n \times n$ with $n > 10^4$ or higher). If A is dense (i.e. not sparse) and full (i.e. without structure) a good choice could be to use the LU factorization.

Note that to store the matrix requires n^2 doubles (e.g. for $n = 10^4$ a total of 0.8 GB of memory!). On the other hand, assume that the machine has essentially two levels of memory hierarchy: fast (e.g. stack memory) and slow (e.g. heap memory). And assume that the slow memory area can store the matrix A but the fast memory works at most with words of size $M \ll n^2$.

Our goal is to reorganize the algorithms so that we can deal with blocks of matrices of the desired size ($\sim M$) to be able to compute the solution and to optimize the algorithm performance.

We start by investigating the classical matrix-matrix multiplication algorithm (`gemm` operation).[1]

Ex.1 Given $A, B \in \mathbb{R}^{n\times n}$ write a function that performs the matrix multiplication update $C = AB + C$ (if one initializes $C = 0$ then the routine performs the matrix multiplication).
Discussion: There are 6 possibilities to reorder the loops, referred as ijk,jik,ikj,jki,kij and kji variations. Each variant has the same amount of flops but features a different inner loop and middle loop operations.

This provides the idea of how one can reorder the algorithm to become rich in level 3 (matrix-matrix) operations of a given size.

A possibility could be to divide the matrix A into blocks

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{N1} & A_{N2} & \dots & A_{NN} \end{pmatrix}$$

where $A_{ij} \in \mathbb{R}^{b\times b}$, being $n = Nb$.

Ex.2 Write a block matrix multiplication algorithm (e.g. Demmel p.69 algorithm), so that we use matrix-matrix multiplication of $b \times b$ matrices to perform the multiplication update $C = AB + C$ of $n \times n$ matrices.

Finally we describe a block-partitioned LU algorithm. A *block-partitioned algorithm* is a scalar algorithm in which the operations are organized in matrix operations (at block level). This is different from a *block algorithm*. For example, the block LU refers to a factorization where L is block matrix with identity matrices in the diagonal (not only ones in the diagonal!). The block-partitioned LU algorithm computes the (classical) LU factorization of the original matrix using block operations.

[1] See p.10 Golub-Van Loan, also p.68 Demmel

The block-partitioned LU is described here. Choose $r \geq 1$ and write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{pmatrix} \begin{pmatrix} I_r & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & I_{n-r} \end{pmatrix}$$

where $A \in \mathbb{R}^{n \times n}$ and $A_{11} \in \mathbb{R}^{r \times r}$. Then, we proceed as follows:

i) Factor $A_{11} = L_{11}U_{11}$ (without pivoting).

ii) Solve $L_{11}U_{12} = A_{12}$ to obtain U_{12}.

iii) Solve $L_{21}U_{11} = A_{21}$ to obtain L_{21}.

iv) Form $S = A_{22} - L_{21}U_{12}$ (Schur complement of A_{11}).

v) Repeat previous steps on S to obtain L_{22} and U_{22}.

This algorithm performs the LU factorization of A with the same number of flops but using small block matrices at each steps.

Ex.3 Write an implementation of the previous algorithm. Use LU with pivoting to solve the linear systems. Note that the LU factorization of A_{11} is required without pivoting, hence write a simple routine to compute this factorization using Gaussian elimination.