CS 598: Lecture 4 notes

Edgar Solomonik
University of Illinois at Urbana-Champaign

1 Rectangular matrix multiplication

These notes are based on the algorithm and analysis from this paper [1].

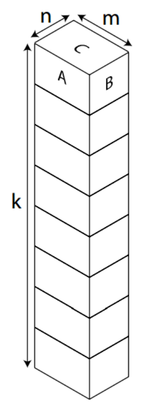
Consider matrix multiplication $C = A \cdot B$ where C is $m \times n$, A is $m \times k$, and B is $k \times n$,

$$\forall i \in [1, m], j \in [1, n], \quad C(i, j) = \sum_{l=1}^{k} A(i, l) \cdot B(l, j).$$

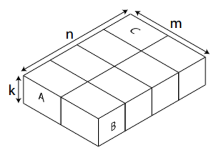
We would like to partition the $m \times n \times k$ cube of matrix multiplications to minimize the surface area of the maximum partition. Using a P processors we can define a $p_1 \times p_2 \times p_3$ grid of processors with $p_1 p_2 p_3 = P$, that yields an optimal bandwidth cost W,

$$W(m, n, k, P) = O\bigg(\min_{p_1 p_2 p_3 = P} \left[\frac{mn}{p_1 p_2} + \frac{mk}{p_1 p_3} + \frac{kn}{p_2 p_3} \right] - \frac{mn + mk + kn}{P} \bigg).$$

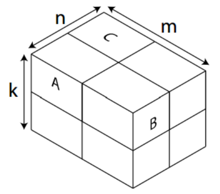
Figure 1 provides three examples of good partitions in which one, two, and three of p_1, p_2, and p_3 are greater than 1.



(a) One large dimension



(b) Two large dimensions



(c) Three large dimensions

We now sketch recursive algorithms that obtain the optimal costs in different scenarios, as well as combinations of these variants. The algorithms assume that $m \geq n \geq k$, so that C is the largest matrix and B is the smallest. It is straight-forward to derive algorithms for the other cases. The algorithms assume that A and B are both distributed evenly among all processors at the start of execution and distribute C evenly among all processors at the end. Each processor performs mnk/P of the mnk multiplications and roughly as many additions. However, the exact layout is not specified, it is most natural to implement the approaches using a blocked or a cyclic distribution, but choosing a good starting processor grid is non-trivial. We will assume throughout the analysis that the available memory is unlimited. Subsequently, we will also not keep track of latency/synchronization cost, which will be $S = O(\log P)$ for all algorithms.

1.1 1D partitions

We first consider partitioning one of the indices. It is best to always choose the one with the largest range, m, which corresponds to partitioning C and A, while communicating B. We define the recursive algorithm as follows,

$$\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot B,$$

where C_1 is $m/2 \times n$. The algorithm computes the following matrix multiplications recursively with $P/2$ processors:

$$C_1 = A_1 \cdot B \qquad C_2 = A_2 \cdot B.$$

These recursive calls can be done without needing to communicate data of A or of C, for instance by ensuring that the first half of the processors owns A_1 and computes C_1. However, we need to communicate the elements of B, since each one is initially stored on a single processor, and we would now need a copy of each element of B on each of the two sets of processors executing the recursive calls. Since each processor starts with kn/P elements of B and needs $2kn/P$ elements of B to execute the recursive call, kn/P elements must be sent and received by each processor. Therefore, we arrive at the cost recurrence,

$$W_{1\text{D}}(m, n, k, P) = W_{1\text{D}}(m/2, n, k, P/2) + kn/P.$$

At the base case, $W_{1\text{D}}(m/P, n, k, 1) = 0$, but the previous recursive level is executed with 2 processors and has cost $W_{1\text{D}}(2m/P, n, k, 2) = kn/2$. Overall, the cost is a geometric sum,

$$W_{1\text{D}}(m, n, k, P) = \sum_{i=1}^{\log_2(P)} kn/(P/2^{i-1}) = (kn/P) \cdot \sum_{i=1}^{\log_2(P)} 2^{i-1} \le kn.$$

The 1D algorithm obtains an optimal partitioning of the computation when $P \ge m/n$, since then $kn \le mk/P \le mn/P$, so we have cut the largest dimension of the $m \times n \times k$ cuboid, and it still corresponds to the largest dimension of each $m/P \times n \times k$ block. Intuitively, we can see that this cost is asymptotically optimal since mn/P and mk/P are the amount of data each processor owns of B and C, and communicating at least double that amount (at least one block) would lead to a higher bandwidth cost than that incurred by the 1D algorithm. In this case, we partition the largest dimension at every recursive step.

1.2 2D partitions

We now consider partitioning two of the indices. It is again always best to choose the largest ones, which in our case are the dimensions of C: m and n. With this partitioning we can keep C stationary while communication A and B. We define the recursive algorithm as follows,

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_2 \end{bmatrix},$$

where $C_{xy} = A_x \cdot B_y$ is $m/2 \times n/2$ for $x, y \in \{1, 2\}$. The algorithm computes those four matrix multiplications recursively with $P/4$ processors.

These recursive calls can be done without needing to communicate partial sums of C. However, we need to communicate the elements of A and of B, since each one is initially stored on a single processor, and we would now need a copy of each element of A_1 on the set of processors computing $C_{11} = A_1 \cdot B_1$ as well as the set of processors computing $C_{12} = A_1 \cdot B_2$. The same holds for the elements of A_2, B_1, and B_2. Therefore, we arrive at the cost recurrence,

$$W_{2\text{D}}(m, n, k, P) = W_{2\text{D}}(m/2, n/2, k, P/4) + \frac{mk + kn}{P}.$$

At the base case, $W_{\text{2D}}(m/\sqrt{P}, n/\sqrt{P}, k, 1) = 0$, but the previous recursive level is executed with 4 processors and has cost $W_{\text{2D}}(2m/\sqrt{P}, 2n/\sqrt{P}, k, 4) = \frac{mk+kn}{2\sqrt{P}}$. Overall, the cost is again a geometric sum,

$$W_{\text{2D}}(m,n,k,P) = \frac{mk+kn}{P}\sum_{i=1}^{\log_4(P)} 2^{i-1} \le \frac{mk+kn}{P}2^{\log_4(P)} = \frac{mk+kn}{P}P^{\log_4(2)} = \frac{mk+kn}{\sqrt{P}}.$$

To obtain a better 2D algorithm when $m \neq n$, we want to use a $p_1 \times p_2$ processor grid where $p_1 \neq p_2$. Ideally, we would like the ratio $m : n$ to equal the ratio $p_1 : p_2$. When $m \geq nP$, this suggests using a 1D algorithm, so we now focus on the cast $m < nP$. One way to express a rectangular processor grid is by recursively combination of 1D and 2D partitioning. After subdividing the processor grid using into partitions of size Pn/m, each partition is assigned a square submatrix of C to compute, so we switch to 2D partitioning, which maintains this property. We can get a combined cost expression for thus 1D+2D algorithm by combining the cost of the first $\log_2(m/n)$ steps of the 1D algorithm, which the 2D algorithm executed on a subproblem of size $n \times n \times k$ using Pn/m processors:

$$\begin{aligned}
W_{\text{1D+2D}}(m,n,k,P) &= \left[(kn/P)\sum_{i=1}^{\log_2(m/n)} 2^{i-1}\right] + W_{\text{2D}}(n,n,k,Pn/m) \\
&\le mk/P + W_{\text{2D}}(n,n,k,Pn/m) \\
&= mk/P + \frac{2nk}{\sqrt{Pn/m}} \\
&= mk/P + 2k\sqrt{mn/P}.
\end{aligned}$$

We can notice that $mk/P = (k\sqrt{m/P})\sqrt{m/P} < (k\sqrt{m/P})\sqrt{n} = k\sqrt{mn/P}$, since $m < nP$, so

$$W_{\text{1D+2D}}(m,n,k,P) < 3k\sqrt{mn/P}.$$

1.3 3D partitions

When all three dimensions are balanced and sufficient memory is available, it is best to partition all three indices at once, resulting in a 3D processor grid. With this partitioning we will need to communicate not only elements of A and B, but partial sums of elements contributing to C. We define the recursive algorithm as follows,

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

where $C_{xy} = A_{x1} \cdot B_{1y} + A_{x2} \cdot B_{2y}$ is $m/2 \times n/2$ for $x, y \in \{1,2\}$, while each A_{xy} and B_{xy} are $m/2 \times k/2$ and $k/2 \times n/2$, respectively.

For $x, y, z \in \{1,2\}$ we execute the subproblem $A_{xz} \cdot B_{zy}$ recursively using $P/8$ processors. Thus each set of $P/8$ processors requires $mk/4$ elements of A, $kn/4$ elements of B, and generates a partial sum for $mn/4$ elements of C. Taking into account the elements each processor owns at the beginning, it is necessary for each processor to receive mk/P elements of A, kn/P elements of B, and then to send mn/P partial sums to elements of C to the final owner of the full sums (elements of output C). Therefore, the cost recurrence for the 3D algorithm is

$$W_{\text{3D}}(m,n,k,P) = W(m/2, n/2, k/2, P/8) + \beta \cdot \frac{mn+mk+kn}{P}.$$

At the base case, $W_{\text{3D}}(m/P^{1/3}, n/P^{1/3}, k/P^{1/3}, 1) = 0$, but the previous recursive level is executed with 8 processors and has cost $W_{\text{3D}}(2m/P^{1/3}, 2n/P^{1/3}, 2k/P^{1/3}, 8) = \frac{mn+mk+kn}{2\sqrt{P}}$. Once again the cost is a geometric sum,

$$W_{\text{3D}}(m,n,k,P) = \frac{mn+mk+kn}{P}\sum_{i=1}^{\log_8(P)} 2^{i-1} \leq \frac{mn+mk+kn}{P}2^{\log_8(P)} = \frac{mn+mk+kn}{P^{2/3}}.$$

To obtain a 3D algorithm that defines a cuboid $p_1 \times p_2 \times p_3$ processor grid, matching the ratios $p_1 : p_2 : p_3$ with $m : n : k$, we can recursively use a 1D, 2D, and 3D partitioning. We assume that $m < np$ and that $n < k\sqrt{Pn/m} \Rightarrow n < k^2P/m$, otherwise the 1D+2D algorithm is optimal. In particular, we employ $\log_2(m/n)$ levels of the 1D algorithm, followed by $\log_2(n/k)$ levels of the 2D algorithm, then switch to the 3D algorithm with sets of $P/(m/n)/(n/k)^2 = Pk^2/(mn)$ processors working on subproblems with dimensions $k \times k \times k$. Combining the costs for these three stages, yields the total cost for the 1D+2D+3D algorithm:

$$\begin{aligned} W_{\text{1D+2D+3D}}(m,n,k,P) &= \left[(kn/P)\sum_{i=1}^{\log_2(m/n)} 2^{i-1}\right] + \left[\frac{2nk}{Pn/m}\sum_{i=1}^{\log_2(n/k)} 2^{i-1}\right] + W_{\text{3D}}\Big(k,k,k,\frac{Pk^2}{mn}\Big) \\ &\leq mk/P + \frac{2mn}{P} + \frac{3k^2}{(Pk^2/(mn))^{2/3}} \\ &= mk/P + \frac{2mn}{P} + 3\Big(\frac{mnk}{P}\Big)^{2/3}. \end{aligned}$$

Since we have $n > k$, the first term is less than mn/P, and furthermore using $n < k^2P/m$,

$$mn/P = (mn/P)^{2/3}(mn/P)^{1/3} < (mn/P)^{2/3}k^{2/3} = (mnk/P)^{2/3}.$$

Substituting this inequality into our total cost, we obtain

$$W_{\text{1D+2D+3D}}(m,n,k,P) < 6\Big(\frac{mnk}{P}\Big)^{2/3}.$$

Now, recall the Loomis-Whitney inequality [2] tells us that for any set $V \in [1,\mathbb{N}]^3$, the surface area (cardinality of the union of 2D projections) is of size at least $|V|^{2/3}$. We have to assign at least mnk/P multiplications to some processor, so the communication cost must have be at least $W_{\text{MM}}(m,n,k,P) > (mnk/P)^{2/3}$. Thus the 1D+2D+3D algorithm is asymptotically communication optimal for the given problem.

References

[1] James Demmel, David Eliahu, Armando Fox, Shoaib Kamil, Benjamin Lipshitz, Oded Schwartz, and Omer Spillinger. Communication-optimal parallel recursive rectangular matrix multiplication. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2013.

[2] Lynn H. Loomis and Hassler Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the AMS*, 55:961–962, 1949.