Updates on Practical Strassen's Algorithms

Rodrigo Brandão
Devangi N. Parikh

UT Austin

BLIS Retreat 2023

Outline

Introduction

Introduction

Consider, $C+=AB$, where C, A and B are $m \times n$, $m \times k$ and $k \times n$ matrices respectively and m, n, k are even.

Introduction

Consider, $C+=AB$, where C, A and B are $m\times n$, $m\times k$ and $k\times n$ matrices respectively and m,n,k are even.

Consider the following partition of C, A and B:

$$\begin{pmatrix} C_0 & C_1 \\ C_2 & C_3 \end{pmatrix} += \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix}$$

Direct Computation

$$
\begin{aligned}
C_0 &:= A_0 B_0 + A_1 B_2 + C_0 \\
C_1 &:= A_0 B_1 + A_1 B_3 + C_1 \\
C_2 &:= A_2 B_0 + A_3 B_2 + C_2 \\
C_3 &:= A_2 B_1 + A_3 B_3 + C_3
\end{aligned}
$$

Direct Computation

$$
\begin{aligned}
C_0 &:= A_0B_0 + A_1B_2 + C_0 \\
C_1 &:= A_0B_1 + A_1B_3 + C_1 \\
C_2 &:= A_2B_0 + A_3B_2 + C_2 \\
C_3 &:= A_2B_1 + A_3B_3 + C_3
\end{aligned}
$$

8 multiplications, 8 additions

Fast Matrix Multiplication

Strassen's Algorithm[1]

$$
\begin{aligned}
&M_0 = (A_0 + A_3)(B_0 + B_3); && C_0 \mathrel{+}= M_0; C_3 \mathrel{+}= M_0;\\
&M_1 = (A_2 + A_3)B_0; && C_2 \mathrel{+}= M_1; C_3 \mathrel{-}= M_1;\\
&M_2 = A_0(B_1 - B_3); && C_1 \mathrel{+}= M_2; C_3 \mathrel{+}= M_2;\\
&M_3 = A_3(B_2 - B_0); && C_0 \mathrel{+}= M_3; C_2 \mathrel{+}= M_3;\\
&M_4 = (A_0 + A_1)B_3; && C_1 \mathrel{+}= M_4; C_0 \mathrel{-}= M_4;\\
&M_5 = (A_2 - A_0)(B_0 + B_1); && C_3 \mathrel{+}= M_5;\\
&M_6 = (A_1 - A_3)(B_2 + B_3); && C_0 \mathrel{+}= M_6;
\end{aligned}
$$

[1] V. Strassen, "Gaussian elimination is not optimal," in Numerische Mathematik, 1969

Strassen's Algorithm[1]

$$
\begin{aligned}
&M_0 = (A_0 + A_3)(B_0 + B_3); && C_0 {+}{=}\ M_0; C_3 {+}{=}\ M_0;\\
&M_1 = (A_2 + A_3)B_0; && C_2 {+}{=}\ M_1; C_3 {-}{=}\ M_1;\\
&M_2 = A_0(B_1 - B_3); && C_1 {+}{=}\ M_2; C_3 {+}{=}\ M_2;\\
&M_3 = A_3(B_2 - B_0); && C_0 {+}{=}\ M_3; C_2 {+}{=}\ M_3;\\
&M_4 = (A_0 + A_1)B_3; && C_1 {+}{=}\ M_4; C_0 {-}{=}\ M_4;\\
&M_5 = (A_2 - A_0)(B_0 + B_1); && C_3 {+}{=}\ M_5;\\
&M_6 = (A_1 - A_3)(B_2 + B_3); && C_0 {+}{=}\ M_6;
\end{aligned}
$$

7 multiplications, 22 additions

[1] V. Strassen, "Gaussian elimination is not optimal," in Numerische Mathematik, 1969

Fast Matrix Multiplication (FMM)

A $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ FMM algorithm[2] by partitioning

$$C = \left(\begin{array}{c|c|c} C_0 & \cdots & C_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline C_{(\widetilde{m}-1)\widetilde{n}} & \cdots & C_{\widetilde{m}\widetilde{n}-1} \end{array} \right), A = \left(\begin{array}{c|c|c} A_0 & \cdots & A_{\widetilde{k}-1} \\ \hline \vdots & & \vdots \\ \hline A_{(\widetilde{m}-1)\widetilde{k}} & \cdots & A_{\widetilde{m}\widetilde{k}-1} \end{array} \right),$$

$$\text{and } B = \left(\begin{array}{c|c|c} B_0 & \cdots & B_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline B_{(\widetilde{k}-1)\widetilde{n}} & \cdots & B_{\widetilde{k}\widetilde{n}-1} \end{array} \right)$$

where A_i, B_j, and C_p are the submatrices of A, B and C, with a single index in the row major order.

[2] A. R. Benson and G. Ballard, "A framework for practical parallel fast matrix multiplication," PPoPP 2015.

Fast Matrix Multiplication (FMM) (contd.)

Then, $C := C + AB$ is computed by, for $r = 0, ..., R-1$,

$$M_r := \left(\sum_{i=0}^{\widetilde{m}\widetilde{k}-1} u_{ir} A_i \right) \times \left(\sum_{j=0}^{\widetilde{k}\widetilde{n}-1} v_{jr} B_j \right);$$
$$C_p += w_{pr} M_r \ (p = 0, ..., \widetilde{m}\widetilde{n} - 1)$$

where $(\times)$ is a matrix multiplication that can be done recursively, u_{ir}, v_{jr}, and w_{pr} are entries of a $(\widetilde{m}\widetilde{k}) \times R$ matrix U, a $(\widetilde{k}\widetilde{n}) \times R$ matrix V, and a $(\widetilde{m}\widetilde{n}) \times R$ matrix W, respectively.

One-level Strassen $\langle 2, 2, 2 \rangle$

The set of coefficients that determine the $\langle 2, 2, 2 \rangle$ algorithm is denoted as $\mathcal{T} = [\![U, V, W]\!]$, where

$$
\begin{array}{cc}
 & \begin{array}{ccccccc} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{array} \\
U = \begin{array}{c} A_0 \\ A_1 \\ A_2 \\ A_3 \end{array} & \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
V = \begin{array}{c} B_0 \\ B_1 \\ B_2 \\ B_3 \end{array} & \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} \\
W = \begin{array}{c} C_0 \\ C_1 \\ C_2 \\ C_3 \end{array} & \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}
\end{array}
$$

Finding other FMM

- For $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ partitioning of the matrices, the associated $\mathcal{T}$ has dimension $(\widetilde{m}\widetilde{k}, \widetilde{k}\widetilde{n}, \widetilde{m}\widetilde{n})$ represents the underlying matrix multiplication.
- Seek a rank-R decomposition of tensor $\mathcal{T}$, where $R < \widetilde{m}\widetilde{k}\widetilde{n}$

Reinforcement Learning + FMM[3]

- Agent "AlphaTensor" is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space.
- For matrices in $\mathbb{R}$ reduces number of multiples:
    1. $\langle 3, 4, 5 \rangle$ from 48 to 47.
    2. $\langle 4, 4, 5 \rangle$ from 64 to 63.
    3. $\langle 4, 5, 5 \rangle$ from 80 to 76.

[3] A. Fawzi et. al, "Discovering faster matrix multiplication algorithms with reinforcement learning," Nature 2023.

Practical FMM

Practical Strassen's Algorithm

	Conventional Implementations
Matrix Size	must be large
Matrix Shapes	must be square
No Additional Workspace	×
Parallelism	usually task parallelism

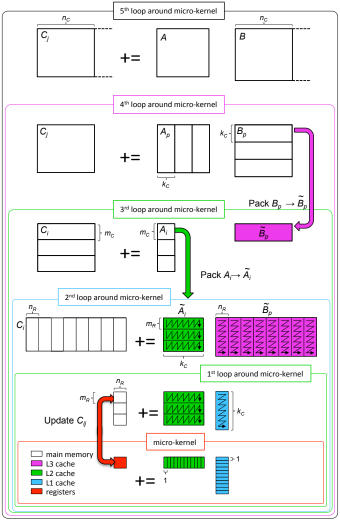
[4] J. Huang et. al, "Strassen's algorithm reloaded," in SC 16. IEEE, 2016.

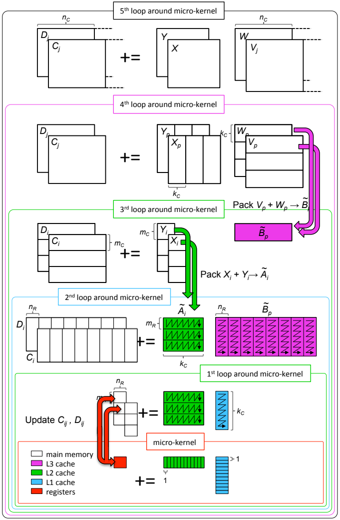
Practical Strassen's Algorithm

	Conventional Implementations	Strassen Reloaded Impl[4]
Matrix Size	must be large	speed up at smaller sizes
Matrix Shapes	must be square	speed up for rank-k
No Additional Workspace	×	✓
Parallelism	usually task parallelism	data parallelism

[4] J. Huang et. al, "Strassen's algorithm reloaded," in SC 16. IEEE, 2016.

Strassen Reloaded[4]





[4] J. Huang et. al, "Strassen's algorithm reloaded," in SC 16. IEEE, 2016.

Generating High-Performance Implementations of FMM[5]

- Generates code that takes as input $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ and $[\![U, V, W]\!]$ and as output generates implementations that build upon the primitives that combine taking linear combinations of matrices with the packing routines and/or micro-kernels that underlie BLIS.
- Provides a model of cost for each implementation that can then be used to choose the best FMM algorithm for a matrix of given size and shape.

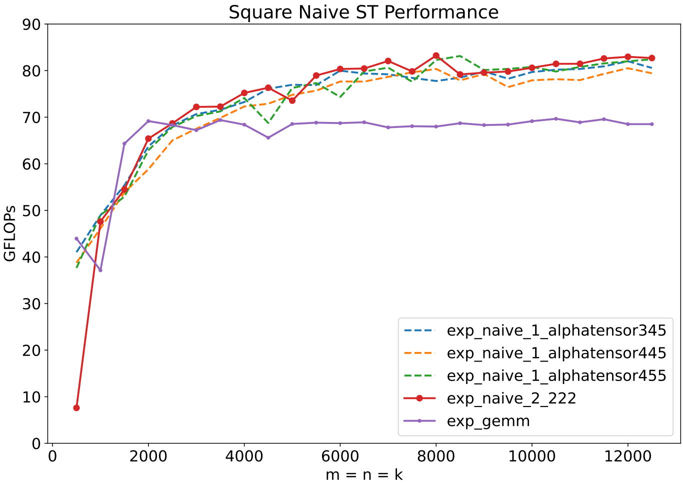
[5] J. Huang et. al, "Generating Families of Practical Fast Matrix Multiplication Algorithms," in IPDPS 2017.

Performance Results

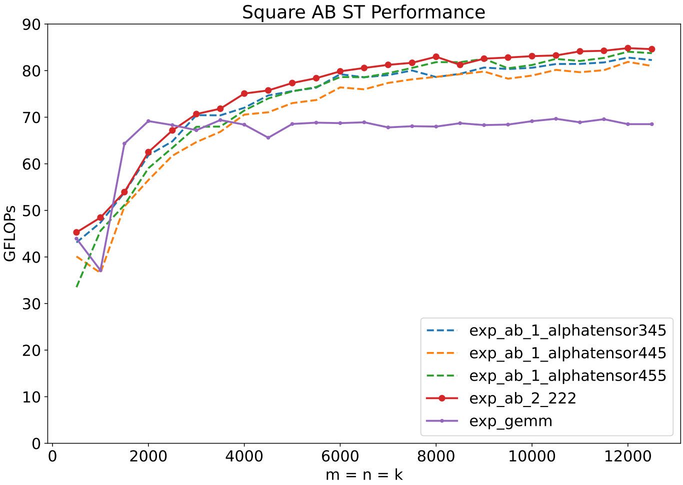
Setup

- Updated the code generator to support Intel's Haswell Architecture
- Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.80/4.20GHz processor
- Three-level cache: L1 128 KB, L2 1 MB, L3 8 MB

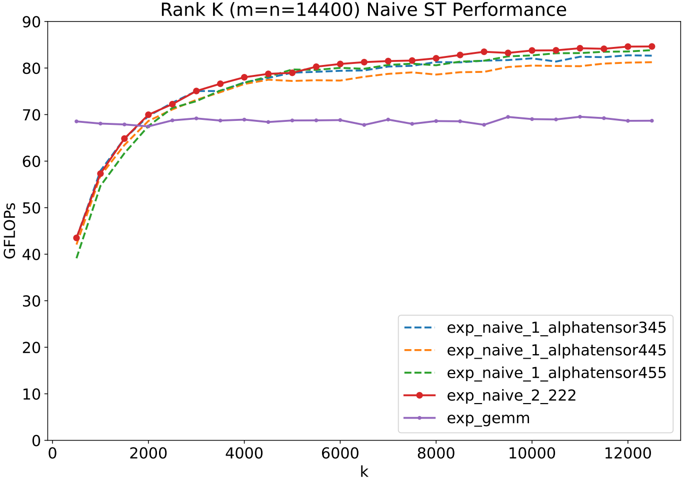
Results - Single-Threaded Naive (m=k=n)



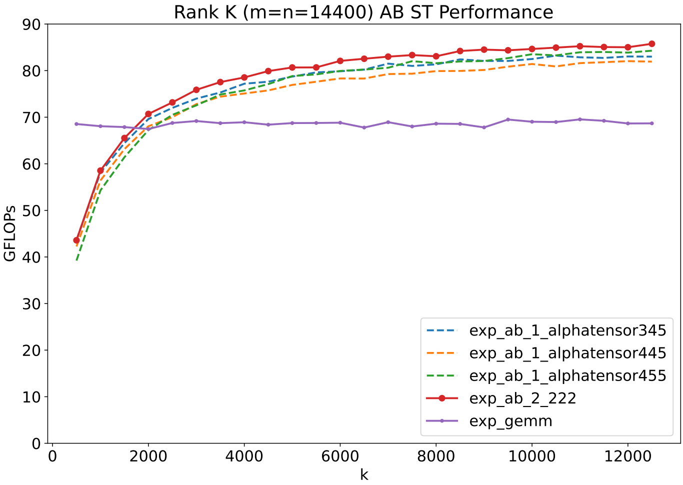
Results - Single-Threaded AB (m=k=n)



Results - Single-Threaded Naive (m=n=14400)



Results - Single-Threaded AB (m=n=14400)



Conclusions

Conclusions

- For a single thread, Naive and AB Strassen continues to perform generally better than the other FMM algorithms.

Future Work

- Create a BLIS plugin that implements Strassen and other FMM using the BLIS framework.
- Perform experiments with multi-threading as well as on other architectures.
- Investigate the necessary conditions for an FMM algorithm to outperform Strassen.

Thank you!