Implementing Strassen-like Fast Matrix Multiplication Algorithms with BLIS

Jianyu Huang, Leslie Rice

Joint work with Tyler M. Smith, Greg M. Henry, Robert A. van de Geijn

BLIS Retreat 2016
Volker Strassen
(Born in 1936, aged 80)

Original Strassen Paper (1969)

Gaussian Elimination is not Optimal

VOLKER STRASSEN

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices $A$ and $B$ of order $n$ from the coefficients of $A$ and $B$ with less than $4^n \log_2 n^2$ arithmetical operations (all logarithms in this paper are for base 2. Thus $\log 2 = 2.0$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order $n$, solving a system of $n$ linear equations in $n$ unknowns, computing a determinant of order $n$ etc. all requiring less than const $n^{\log_2 7}$ arithmetical operations.

This fact should be compared with the result of Klyuyev and Korovien-Scherbak [4] that Gaussian elimination for solving a system of linear equations is optimal if one restricts oneself to operations upon rows and columns as a whole. We also note that Wosnograd [2] modifies the usual algorithms for matrix multiplication and inversion and for solving systems of linear equations, trading roughly half of the multiplications for additions and subtractions.

It is a pleasure to thank D. Bussinger for inspiring discussions about the present subject and St. Eve and B. Fawzy for encouraging me to write this paper.

2. We define algorithms $\sigma_{n, k}$ which multiply matrices of order $m^2$, by induction on $k$: $\sigma_{n,k}$ is the usual algorithm for matrix multiplication (requiring $m^2$ multiplications and $m^2(m-1)$ additions). $\sigma_{n,1}$ already being known, define $\sigma_{n, k+1}$ as follows:

If $A$, $B$ are matrices of order $m^2$, $m^2+1$ to be multiplied, write

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{13} & a_{14} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{13} & b_{14} \end{pmatrix}, \quad AB = \begin{pmatrix} c_{11} & c_{12} \\ c_{13} & c_{14} \end{pmatrix},$$

where the $a_{ik}, b_{ij}, c_{ij}$ are matrices of order $m^2$. Then compute

$$1 = (a_{11} + a_{12}) (b_{11} + b_{12}),$$
$$II = (a_{11} + a_{12}) b_{13},$$
$$III = a_{13} (b_{11} - b_{12}),$$
$$IV = a_{13} (b_{11} + b_{12}),$$
$$V = (a_{11} + a_{12}) b_{14},$$
$$VI = (-a_{11} + a_{12}) (b_{11} + b_{12}),$$
$$VII = (a_{11} - a_{12}) (b_{11} + b_{12}).$$

* The results have been found while the author was at the Department of Statistics of the University of California, Berkeley. The author wishes to thank the National Science Foundation for their support (NSF GP-7454).
One-level Strassen’s Algorithm (In theory)

Assume $m$, $n$, and $k$ are all even. $A$, $B$, and $C$ are $m \times k$, $k \times n$, $m \times n$ matrices, respectively. Letting

$$C = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}, \quad A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}, \quad B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

We can compute $C := C + AB$ by

Direct Computation

$$C_{00} := A_{00}B_{00} + A_{01}B_{10} + C_{00};$$
$$C_{01} := A_{00}B_{01} + A_{01}B_{11} + C_{01};$$
$$C_{10} := A_{10}B_{00} + A_{11}B_{10} + C_{10};$$
$$C_{11} := A_{10}B_{01} + A_{11}B_{11} + C_{11};$$

8 multiplications, 8 additions

Strassen’s Algorithm

$$M_0 := (A_{00} + A_{11})(B_{00} + B_{11});$$
$$M_1 := (A_{10} + A_{11})B_{00};$$
$$M_2 := A_{00}(B_{01} - B_{11});$$
$$M_3 := A_{11}(B_{10} - B_{00});$$
$$M_4 := (A_{00} + A_{01})B_{11};$$
$$M_5 := (A_{10} - A_{00})(B_{00} + B_{01});$$
$$M_6 := (A_{01} - A_{11})(B_{10} + B_{11});$$
$$C_{00} := M_0 + M_3 - M_4 + M_7 + C_{00};$$
$$C_{01} := M_2 + M_4 + C_{01};$$
$$C_{10} := M_1 + M_3 + C_{10};$$
$$C_{11} := M_0 - M_1 + M_2 + M_5 + C_{11}.$$ 

7 multiplications, 22 additions

Multi-level Strassen’s Algorithm (In theory)

- One-level Strassen (1+14.3% speedup)
  - 8 multiplications → 7 multiplications;
- Two-level Strassen (1+30.6% speedup)
  - 64 multiplications → 49 multiplications;
- d-level Strassen ($n^3/n^{2.803}$ speedup)
  - $8^d$ multiplications → $7^d$ multiplications;

If originally $m = n = k = 2^d$, where $d$ is an integer, then the cost becomes

$$(7/8)^{\log_2(n)} 2n^3 = n^{\log_2(7/8)} 2n^3 \approx 2n^{2.807}$$

flops.
Multi-level Strassen’s Algorithm (In theory)

\[ M_0 := (A_{00} + A_{11})(B_{00} + B_{11}); \]
\[ M_1 := (A_{10} + A_{11})B_{00}; \]
\[ M_2 := A_{00}(B_{01} - B_{11}); \]
\[ M_3 := A_{11}(B_{10} - B_{00}); \]
\[ M_4 := (A_{00} + A_{01})B_{11}; \]
\[ M_5 := (A_{10} - A_{00})(B_{00} + B_{01}); \]
\[ M_6 := (A_{01} - A_{11})(B_{10} + B_{11}); \]
\[ C_{00} += M_0 + M_3 - M_4 + M_6 \]
\[ C_{01} += M_2 + M_4 \]
\[ C_{10} += M_1 + M_3 \]
\[ C_{11} += M_0 - M_1 + M_2 + M_5 \]

- One-level Strassen (1+14.3% speedup)
  - 8 multiplications → 7 multiplications ;
- Two-level Strassen (1+30.6% speedup)
  - 64 multiplications → 49 multiplications;
- \(d\)-level Strassen (\(n^3/n^{2.803}\) speedup)
  - \(8^d\) multiplications → \(7^d\) multiplications;
Strassen’s Algorithm (In practice)

\[ M_0 := (A_{00} + A_{11})(B_{00} + B_{11}); \]
\[ M_1 := (A_{10} + A_{11})B_{00}; \]
\[ M_2 := A_{00}(B_{01} - B_{11}); \]
\[ M_3 := A_{11}(B_{10} - B_{00}); \]
\[ M_4 := (A_{00} + A_{01})B_{11}; \]
\[ M_5 := (A_{10} - A_{00})(B_{00} + B_{01}); \]
\[ M_6 := (A_{01} - A_{11})(B_{10} + B_{11}); \]
\[ C_{00} += M_0 + M_3 - M_4 + M_6 \]
\[ C_{01} += M_2 + M_4 \]
\[ C_{10} += M_1 + M_3 \]
\[ C_{11} += M_0 - M_1 + M_2 + M_5 \]
Strassen’s Algorithm (In practice)

- One-level Strassen (1+14.3% speedup)
  - 7 multiplications + 22 additions;
- Two-level Strassen (1+30.6% speedup)
  - 49 multiplications + 344 additions;
Strassen’s Algorithm (In practice)

- One-level Strassen (1+14.3% speedup)
  - 7 multiplications + 22 additions;
- Two-level Strassen (1+30.6% speedup)
  - 49 multiplications + 344 additions;
- $d$-level Strassen ($n^3/n^{2.803}$ speedup)
  - Numerical unstable; Not achievable

\begin{align*}
M_0 &:= (A_{00} + A_{11})(B_{00} + B_{11}); \\
M_1 &:= (A_{10} + A_{11})B_{00}; \\
M_2 &:= A_{00}(B_{01} - B_{11}); \\
M_3 &:= A_{11}(B_{10} - B_{00}); \\
M_4 &:= (A_{00} + A_{01})B_{11}; \\
M_5 &:= (A_{10} - A_{00})(B_{00} + B_{01}); \\
M_6 &:= (A_{01} - A_{11})(B_{10} + B_{11}); \\
C_{00} &= M_0 + M_3 - M_4 + M_6 \\
C_{01} &= M_2 + M_4 \\
C_{10} &= M_1 + M_3 \\
C_{11} &= M_0 - M_1 + M_2 + M_5
\end{align*}
To achieve practical high performance of Strassen’s algorithm......

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Conventional Implementations</th>
<th>Our Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Must be large</td>
<td>![Sad Face]</td>
<td></td>
</tr>
</tbody>
</table>

| Matrix Shape | Must be square | ![Sad Face] |

| No Additional Workspace | ![X] | ![Sad Face] |

| Parallelism | | |
To achieve practical high performance of Strassen’s algorithm...

<table>
<thead>
<tr>
<th></th>
<th>Conventional Implementations</th>
<th>Our Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Size</td>
<td>Must be large</td>
<td></td>
</tr>
<tr>
<td>Matrix Shape</td>
<td>Must be square</td>
<td></td>
</tr>
<tr>
<td>No Additional Workspace</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Usually task parallelism</td>
<td></td>
</tr>
</tbody>
</table>
To achieve practical high performance of Strassen’s algorithm......

<table>
<thead>
<tr>
<th>Feature</th>
<th>Conventional Implementations</th>
<th>Our Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Size</td>
<td>Must be large</td>
<td>Yes</td>
</tr>
<tr>
<td>Matrix Shape</td>
<td>Must be square</td>
<td>Yes</td>
</tr>
<tr>
<td>No Additional Workspace</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Usually task parallelism</td>
<td>Can be data parallelism</td>
</tr>
</tbody>
</table>
Outline

• Review of State-of-the-art GEMM in BLIS
• Strassen’s Algorithm Reloaded
• Theoretical Model and Practical Performance
• Extension to Other BLAS-3 Operation
• Extension to Other Fast Matrix Multiplication
• Conclusion
Level-3 BLAS Matrix-Matrix Multiplication (GEMM)

- (General) matrix-matrix multiplication (GEMM) is supported in the level-3 BLAS* interface as

  \[
  \text{dgemm}( \text{transa}, \text{transb}, m, n, k, \alpha, A, \text{lda}, B, \text{ldb}, \beta, C, \text{ldc} )
  \]

- Ignoring transa and transb, GEMM computes

  \[
  C := \alpha AB + \beta C;
  \]

- We consider the simplified version of GEMM

  \[
  C := \alpha AB + C
  \]

State-of-the-art GEMM in BLIS

• BLAS-like Library Instantiation Software (BLIS) is a portable framework for instantiating BLAS-like dense linear algebra libraries.

• BLIS provides a refactoring of GotoBLAS algorithm (best-known approach) to implement GEMM.

• GEMM implementation in BLIS has 6-layers of loops. The outer 5 loops are written in C. The inner-most loop (micro-kernel) is written in assembly for high performance.
  - Partition matrices into smaller blocks to fit into the different memory hierarchy.
  - The order of these loops is designed to utilize the cache reuse rate.
State-of-the-art GEMM in BLIS

- BLAS-like Library Instantiation Software (BLIS) is a portable framework for instantiating BLAS-like dense linear algebra libraries.

- BLIS provides a refactoring of GotoBLAS algorithm (best-known approach) to implement GEMM.

- GEMM implementation in BLIS has 6-layers of loops. The outer 5 loops are written in C. The inner-most loop (micro-kernel) is written in assembly for high performance.
  - Partition matrices into smaller blocks to fit into the different memory hierarchy.
  - The order of these loops is designed to utilize the cache reuse rate.

- BLIS opens the black box of GEMM, leading to many applications built on BLIS.
  - Ch ench D. Yu, Jianyu Huang, Woody Austin, Bo Xiao, and George Biros. "Performance Optimization for the k-Nearest Neighbors Kernel on x86 Architectures." In *SC’15.*

GotoBLAS algorithm for GEMM in BLIS

\[ mC + mA \times kB \rightarrow C \]
GotoBLAS algorithm for GEMM in BLIS

\[
\begin{align*}
C_j &= A + B_j \\
\end{align*}
\]

Loop 5 \( \text{for } j_c = 0 : n - 1 \text{ steps of } n_c \)
\[
J_c = j_c : j_c + n_c - 1
\]

GotoBLAS algorithm for GEMM in BLIS

\[ m \begin{bmatrix} C \\ n \end{bmatrix} = m \begin{bmatrix} A \times k \\ B \end{bmatrix} \]

Loop 5  \( \text{for } j_c = 0 : n - 1 \text{ steps of } n_c \)
\[ J_c = j_c : j_c + n_c - 1 \]

Loop 4  \( \text{for } p_c = 0 : k - 1 \text{ steps of } k_c \)
\[ P_c = p_c : p_c + k_c - 1 \]
\[ B(P_c, J_c) \rightarrow B_p \]

Field G. Van Zee, and Tyler M. Smith. “Implementing high-performance complex matrix multiplication.” In ACM Transactions on Mathematical Software (TOMS), accepted pending modifications.
Field G. Van Zee, and Tyler M. Smith. “Implementing high-performance complex matrix multiplication.” In ACM Transactions on Mathematical Software (TOMS), accepted pending modifications.
GotoBLAS algorithm for GEMM in BLIS

\[
\begin{align*}
&\mathbf{C}_{ij} + = \mathbf{A}_{ij} \times \mathbf{B}_{ij} \\
&\mathbf{C}_{ij} + = \mathbf{A}_{ij} \times \mathbf{B}_{ij} \\
&\mathbf{C}_{ij} + = \mathbf{A}_{ij} \times \mathbf{B}_{ij} \\
&\mathbf{C}_{ij} + = \mathbf{A}_{ij} \times \mathbf{B}_{ij} \\
&\mathbf{C}_{ij} + = \mathbf{A}_{ij} \times \mathbf{B}_{ij}
\end{align*}
\]

Loop 1
for \( i_r = 0 : m_r - 1 \) steps of \( m_r \)
\( I_r = i_r : i_r + m_r - 1 \)

Loop 2
for \( j_r = 0 : n_r - 1 \) steps of \( n_r \)
\( J_r = j_r : j_r + n_r - 1 \)

Loop 3
for \( i_c = 0 : m_c - 1 \) steps of \( m_c \)
\( I_c = i_c : i_c + m_c - 1 \)
\( B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow \mathbf{B}_p \)

Loop 4
for \( p_c = 0 : k - 1 \) steps of \( k_c \)
\( \mathcal{P}_c = p_c : p_c + k_c - 1 \)
\( \mathbf{B}(\mathcal{P}_c, \mathcal{J}_c) \rightarrow \mathbf{B}_p \)

Loop 5
for \( j_c = 0 : n - 1 \) steps of \( n_c \)
\( \mathcal{J}_c = j_c : j_c + n_c - 1 \)

Pack \( \mathbf{A}_i \rightarrow \mathbf{A}_j \)

Pack \( \mathbf{B}_p \rightarrow \mathbf{B}_p \)

endfor
endfor
endfor
endfor
endfor

GotoBLAS algorithm for GEMM in BLIS

\[
\begin{align*}
&\text{Loop } 5 \quad \text{for } j_c = 0 : n - 1 \text{ steps of } n_c \\
&\quad \mathcal{J}_c = j_c : j_c + n_c - 1 \\
&\text{Loop } 4 \quad \text{for } p_c = 0 : k - 1 \text{ steps of } k_c \\
&\quad \mathcal{P}_c = p_c : p_c + k_c - 1 \\
&\quad B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow B_p \\
&\text{Loop } 3 \quad \text{for } i_c = 0 : m - 1 \text{ steps of } m_c \\
&\quad \mathcal{I}_c = i_c : i_c + m_c - 1 \\
&\quad A(\mathcal{I}_c, \mathcal{P}_c) \rightarrow A_i \\
&\quad // \text{macro-kernel} \\
&\text{Loop } 2 \quad \text{for } j_r = 0 : n_c - 1 \text{ steps of } n_r \\
&\quad \mathcal{J}_r = j_r : j_r + n_r - 1 \\
&\text{Loop } 1 \quad \text{for } i_r = 0 : m_c - 1 \text{ steps of } m_r \\
&\quad \mathcal{I}_r = i_r : i_r + m_r - 1 \\
&\quad // \text{micro-kernel} \\
&\text{Loop } 0 \quad \text{for } p_r = 0 : p_c - 1 \text{ steps of } 1 \\
&\quad C_c(\mathcal{I}_r, \mathcal{J}_r) \leftarrow a A_i(\mathcal{I}_r, p_r) \tilde{B}_p(p_r, \mathcal{J}_r) \\
&\text{endfor} \\
&\text{endfor} \\
&\text{endfor} \\
&\text{endfor} \\
&\text{endfor}
\end{align*}
\]

GotoBLAS algorithm for GEMM in BLIS

\[ mC \leftrightarrow n \leftrightarrow mA \times k \rightarrow kB \]

Loop 5 \hspace{1cm} for \( j_c = 0 : n - 1 \) steps of \( n_c \)
\[ J_c = j_c : j_c + n_c - 1 \]
Loop 4 \hspace{1cm} for \( p_c = 0 : k - 1 \) steps of \( k_c \)
\[ P_c = p_c : p_c + k_c - 1 \]
\[ B(P_c, J_c) \rightarrow B_p \]
Loop 3 \hspace{1cm} for \( i_c = 0 : m - 1 \) steps of \( m_c \)
\[ I_c = i_c : i_c + m_c - 1 \]
\[ A(I_c, P_c) \rightarrow A_i \]
\[ // \text{macro-kernel} \]
Loop 2 \hspace{1cm} for \( j_r = 0 : n_c - 1 \) steps of \( n_r \)
\[ J_r = j_r : j_r + n_r - 1 \]
Loop 1 \hspace{1cm} for \( i_r = 0 : m_c - 1 \) steps of \( m_r \)
\[ I_r = i_r : i_r + m_r - 1 \]
\[ //\text{micro-kernel} \]
Loop 0 \hspace{1cm} for \( p_r = 0 : p_c - 1 \) steps of \( 1 \)
\[ C_c(I_r, J_r) \leftarrow \alpha A_i(I_r, p_r) B_p(p_r, J_r) \]
endfor
endfor
endfor
endfor
endfor

Outline

• Review of State-of-the-art GEMM in BLIS
• Strassen’s Algorithm Reloaded
• Theoretical Model and Practical Performance
• Extension to Other BLAS-3 Operation
• Extension to Other Fast Matrix Multiplication
• Conclusion
One-level Strassen’s Algorithm Reloaded

One-level Strassen’s Algorithm Reloaded

\[
M_0 := \alpha(A_{00} + A_{11})(B_{00} + B_{11});
\]
\[
M_1 := \alpha(A_{10} + A_{11})B_{00};
\]
\[
M_2 := \alpha A_{00}(B_{01} - B_{11});
\]
\[
M_3 := \alpha A_{11}(B_{10} - B_{00});
\]
\[
M_4 := \alpha(A_{00} + A_{01})B_{11};
\]
\[
M_5 := \alpha(A_{10} - A_{00})(B_{00} + B_{01});
\]
\[
M_6 := \alpha(A_{01} - A_{11})(B_{10} + B_{11});
\]
\[
C_{00} += M_0 + M_3 - M_4 + M_6;
\]
\[
C_{01} += M_2 + M_4;
\]
\[
C_{10} += M_1 + M_3;
\]
\[
C_{11} += M_0 - M_1 + M_2 + M_5;
\]

General operation for one-level Strassen:

\[
M := \alpha(X + Y)(V + W);
\]

\[
M_0 := \alpha(A_{00} + A_{11})(B_{00} + B_{11});
\]
\[
M_1 := \alpha(A_{10} + A_{11})B_{00};
\]
\[
M_2 := \alpha A_{00}(B_{01} - B_{11});
\]
\[
M_3 := \alpha A_{11}(B_{10} - B_{00});
\]
\[
M_4 := \alpha(A_{00} + A_{01})B_{11};
\]
\[
M_5 := \alpha(A_{10} - A_{00})(B_{00} + B_{01});
\]
\[
M_6 := \alpha(A_{01} - A_{11})(B_{10} + B_{11});
\]
\[
C_{00} += M_0; C_{11} += M_0;
\]
\[
C_{01} += M_1; C_{11} -= M_1;
\]
\[
C_{01} += M_2; C_{11} += M_2;
\]
\[
C_{01} += M_3; C_{10} += M_3;
\]
\[
C_{01} += M_4; C_{00} -= M_4;
\]
\[
C_{10} += M_5;
\]
\[
C_{00} += M_6;
\]

\[
M := \alpha(X + Y)(V + W);
\]

\[
C += \gamma_0 M; D += \gamma_1 M;
\]

\[
\gamma_0, \gamma_1, \delta, \epsilon \in \{-1, 0, 1\}.
\]
High-performance implementation of the general operation?

\[ M := \alpha (X + \delta Y)(V + \varepsilon W); \]
\[ C += \gamma_0 M; D += \gamma_1 M; \]
\[ \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}. \]

*http://shpc.ices.utexas.edu/index.html*
\( M := \alpha (X + \delta Y)(V + \varepsilon W); \quad C := \gamma_0 M; \quad D := \gamma_1 M; \)
\( \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}. \)

*Jianyu Huang*, Tyler Smith, Greg Henry, and Robert van de Geijn.
“Strassen’s Algorithm Reloaded.” In *SC’16.*
\( M := \alpha(X + \delta Y)(V + \varepsilon W); \quad C += \gamma_0 M; \quad D += \gamma_1 M; \quad \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}. \)

**Loop 5**  
\[ \text{for } j_c = 0 : n - 1 \text{ steps of } n_c \]  
\[ J_c = j_c : j_c + n_c - 1 \]  

*Jianyu Huang*, Tyler Smith, Greg Henry, and Robert van de Geijn.  
“Strassen’s Algorithm Reloaded.” In *SC’16*. 
\[ M := \alpha (X + \delta Y)(V + \epsilon W); \quad C := \gamma_0 M; \quad D := \gamma_1 M; \quad \gamma_0, \gamma_1, \delta, \epsilon \in \{-1, 0, 1\}. \]

Loop 5  
for \( j_c = 0 : n - 1 \) steps of \( n_c \)
\[ J_c = j_c : j_c + n_c - 1 \]

Loop 4  
for \( p_c = 0 : k - 1 \) steps of \( k_c \)
\[ P_c = p_c : p_c + k_c - 1 \]
\[ V(P_c, J_c) + \epsilon W(P_c, J_c) \rightarrow \tilde{B}_p \]

*Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn.*  
“Strassen’s Algorithm Reloaded.” In *SC’16.*
\( M := \alpha (X + \delta Y) (V + \epsilon W); \quad C += \gamma_0 M; \quad D += \gamma_1 M; \quad \gamma_0, \gamma_1, \delta, \epsilon \in \{-1, 0, 1\}. \)

Loop 5  
\text{for } j_c = 0 : n - 1 \text{ steps of } n_c

\text{Loop 4  for } p_c = 0 : k - 1 \text{ steps of } k_c

\text{Loop 3  for } i_c = 0 : m - 1 \text{ steps of } m_c

\text{endfor}  
\text{endfor}  
\text{endfor}

*Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn.
“Strassen’s Algorithm Reloaded.” In *SC’16.*
\[ M := \alpha (X + \delta Y) (V + \varepsilon W) \; \quad C += \gamma_0 M \; \quad D += \gamma_1 M \]
\[ \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\} \]

**Loop 5**  
for \( j_c = 0 : n - 1 \) steps of \( n_c \)  
\( J_c = j_c : j_c + n_c - 1 \)

**Loop 4**  
for \( p_c = 0 : k - 1 \) steps of \( k_c \)  
\( P_c = p_c : p_c + k_c - 1 \)

**Loop 3**  
for \( i_c = 0 : m - 1 \) steps of \( m_c \)  
\( I_c = i_c : i_c + m_c - 1 \)

**Loop 2**  
for \( j_r = 0 : n_c - 1 \) steps of \( n_r \)  
\( J_r = j_r : j_r + n_r - 1 \)

**Loop 1**  
for \( i_r = 0 : m_c - 1 \) steps of \( m_r \)  
\( I_r = i_r : i_r + m_r - 1 \)

**Loop 0**  
for \( p_r = 0 : p_c - 1 \) steps of \( 1 \)

\[ M_r (I_r, J_r) += \tilde{A}_i (I_r, p_r) \tilde{B}_p (p_r, J_r) \]

endfor

\[ C (I_r + i_c, J_r + j_c) += \alpha \gamma_0 M_r (I_r, J_r) \]

\[ D (I_r + i_c, J_r + j_c) += \alpha \gamma_1 M_r (I_r, J_r) \]

endfor

endfor

endfor

---

*Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn.*  
“Strassen’s Algorithm Reloaded.” In *SC’16.*
\[ M := \alpha (X + \delta Y)(V + \varepsilon W); \quad C += \gamma_0 M; \quad D += \gamma_1 M; \]
\[ \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}. \]

**Loop 5**  
for \( j_c = 0 : n - 1 \) steps of \( n_c \)
\[ J_c = j_c : j_c + n_c - 1 \]

**Loop 4**  
for \( p_c = 0 : k - 1 \) steps of \( k_c \)
\[ P_c = p_c : p_c + k_c - 1 \]
\[ V(P_c, J_c) + \varepsilon W(P_c, J_c) \rightarrow \tilde{B}_p \]

**Loop 3**  
for \( i_c = 0 : m - 1 \) steps of \( m_c \)
\[ I_c = i_c : i_c + m_c - 1 \]
\[ X(I_c, P_c) + \delta Y(I_c, P_c) \rightarrow \tilde{A}_i \]

// macro-kernel  
**Loop 2**  
for \( j_r = 0 : n_c - 1 \) steps of \( n_r \)
\[ J_r = j_r : j_r + n_r - 1 \]

**Loop 1**  
for \( i_r = 0 : m_c - 1 \) steps of \( m_r \)
\[ I_r = i_r : i_r + m_r - 1 \]

// micro-kernel  
**Loop 0**  
for \( p_r = 0 : p_c - 1 \) steps of \( 1 \)
\[ M_r(I_r, J_r) += \tilde{A}_i(I_r, p_r) \tilde{B}_p(p_r, J_r) \]
endfor  
\[ C(I_r + i_c, J_r + j_c) += \alpha \gamma_0 M_r(I_r, J_r) \]
\[ D(I_r + i_c, J_r + j_c) += \alpha \gamma_1 M_r(I_r, J_r) \]
endfor  
endfor  
endfor  
endfor

---

*Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn.*  
“Strassen’s Algorithm Reloaded.” In *SC’16.*
Two-level Strassen’s Algorithm Reloaded

Assume $m$, $n$, and $k$ are all multiples of 4. Letting

$$
C = \begin{pmatrix}
C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\
C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\
C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\
C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3}
\end{pmatrix},
A = \begin{pmatrix}
A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\
A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix},
B = \begin{pmatrix}
B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\
B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\
B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\
B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3}
\end{pmatrix},
$$

where $C_{i,j}$ is $\frac{m}{4} \times \frac{n}{4}$, $A_{i,p}$ is $\frac{m}{4} \times \frac{k}{4}$, and $B_{p,j}$ is $\frac{k}{4} \times \frac{n}{4}$.
## Two-level Strassen’s Algorithm Reloaded

### General operation for two-level Strassen:

\[
M := \alpha(x_0 + x_1 + x_2 + x_3)(V + V_1 + V_2 + V_3);
\]

\[
\begin{align*}
M_0 &:= \alpha(a_{0,0} + a_{2,2} + a_{1,1} + a_{3,3})(B_{0,0} + B_{2,2} + B_{1,1} + B_{3,3}); \\
C_{0,0} &= M_0; \\
C_{1,1} &= M_0; \\
C_{2,2} &= M_0; \\
C_{3,3} &= M_0;
\end{align*}
\]

\[
\begin{align*}
M_1 &:= \alpha(a_{1,0} + a_{3,2} + a_{1,1} + a_{3,3})(B_{0,0} + B_{2,2}); \\
C_{1,0} &= M_1; \\
C_{1,1} &= M_1; \\
C_{2,2} &= M_1; \\
C_{3,3} &= M_1;
\end{align*}
\]

\[
\begin{align*}
M_2 &:= \alpha(a_{0,0} + a_{2,2})(B_{0,0} + B_{2,2} + B_{1,1} + B_{3,3}); \\
C_{0,1} &= M_2; \\
C_{1,1} &= M_2; \\
C_{2,2} &= M_2; \\
C_{3,3} &= M_2;
\end{align*}
\]

\[
\begin{align*}
M_3 &:= \alpha(a_{1,1} + a_{3,3})(B_{1,0} + B_{3,2} + B_{0,0} + B_{2,2}); \\
C_{0,0} &= M_3; \\
C_{1,0} &= M_3; \\
C_{2,2} &= M_3; \\
C_{3,2} &= M_3;
\end{align*}
\]

\[
\begin{align*}
M_4 &:= \alpha(a_{0,0} + a_{2,2} + a_{0,1} + a_{2,3})(B_{1,1} + B_{3,3}); \\
C_{0,0} &= M_4; \\
C_{1,0} &= M_4; \\
C_{2,2} &= M_4; \\
C_{3,3} &= M_4;
\end{align*}
\]

\[
\begin{align*}
M_5 &:= \alpha(a_{1,0} + a_{3,2} + a_{0,0} + a_{2,2})(B_{0,0} + B_{2,2} + B_{0,1} + B_{2,3}); \\
C_{1,1} &= M_5; \\
C_{3,2} &= M_5; \\
C_{3,3} &= M_5; \\
C_{3,3} &= M_5;
\end{align*}
\]

\[
\begin{align*}
M_6 &:= \alpha(a_{0,1} + a_{2,3} + a_{1,1} + a_{3,3})(B_{1,0} + B_{3,2} + B_{1,1} + B_{3,3}); \\
C_{0,0} &= M_6; \\
C_{2,2} &= M_6; \\
C_{3,3} &= M_6; \\
C_{3,3} &= M_6;
\end{align*}
\]

\[
\begin{align*}
M_7 &:= \alpha(a_{2,0} + a_{2,2} + a_{3,1} + a_{3,3})(B_{0,0} + B_{1,1}); \\
C_{2,0} &= M_7; \\
C_{3,1} &= M_7; \\
C_{3,3} &= M_7; \\
C_{3,3} &= M_7;
\end{align*}
\]

\[
\begin{align*}
M_8 &:= \alpha(a_{3,0} + a_{3,2} + a_{3,1} + a_{3,3})(B_{0,0}); \\
C_{2,1} &= M_8; \\
C_{3,1} &= M_8; \\
C_{3,3} &= M_8; \\
C_{3,3} &= M_8;
\end{align*}
\]

\[
\begin{align*}
M_9 &:= \alpha(a_{2,0} + a_{2,2})(B_{0,1} + B_{1,1}); \\
C_{2,0} &= M_9; \\
C_{3,1} &= M_9; \\
C_{3,3} &= M_9; \\
C_{3,3} &= M_9;
\end{align*}
\]

\[
\begin{align*}
M_{10} &:= \alpha(a_{3,1} + a_{3,3})(B_{1,0} + B_{0,0}); \\
C_{2,0} &= M_{10}; \\
C_{3,0} &= M_{10}; \\
C_{3,2} &= M_{10}; \\
C_{3,2} &= M_{10};
\end{align*}
\]

\[
\begin{align*}
M_{40} &:= \alpha(a_{3,0} + a_{1,0} + a_{2,0} + a_{0,0})(B_{0,0} + B_{0,2} + B_{0,1} + B_{0,3}); \\
C_{3,3} &= M_{40}; \\
C_{2,2} &= M_{41}; \\
C_{2,2} &= M_{42}; \\
C_{1,1} &= M_{43};
\end{align*}
\]

\[
\begin{align*}
M_{41} &:= \alpha(a_{2,1} + a_{0,1} + a_{3,1} + a_{1,1})(B_{1,0} + B_{1,2} + B_{1,1} + B_{1,3}); \\
C_{0,0} &= M_{42}; \\
C_{1,0} &= M_{43}; \\
C_{1,1} &= M_{44};
\end{align*}
\]

\[
\begin{align*}
M_{42} &:= \alpha(a_{0,2} + a_{2,2} + a_{1,3} + a_{3,3})(B_{2,0} + B_{2,2} + B_{3,1} + B_{3,3}); \\
C_{0,0} &= M_{44}; \\
C_{1,0} &= M_{45}; \\
C_{0,0} &= M_{46};
\end{align*}
\]

\[
\begin{align*}
M_{43} &:= \alpha(a_{1,2} + a_{3,2} + a_{1,3} + a_{3,3})(B_{2,0} + B_{2,2}); \\
C_{0,0} &= M_{46}; \\
C_{0,0} &= M_{47};
\end{align*}
\]

\[
\begin{align*}
M_{44} &:= \alpha(a_{0,2} + a_{2,2})(B_{2,1} + B_{2,3} + B_{3,1} + B_{3,3}); \\
C_{1,1} &= M_{47}; \\
C_{0,0} &= M_{48};
\end{align*}
\]

\[
\begin{align*}
M_{45} &:= \alpha(a_{1,3} + a_{3,3})(B_{3,0} + B_{3,2} + B_{2,0} + B_{2,2}); \\
C_{1,1} &= M_{48}; \\
C_{0,0} &= M_{50};
\end{align*}
\]

\[
\begin{align*}
M_{46} &:= \alpha(a_{0,2} + a_{2,2} + a_{0,3} + a_{2,3})(B_{3,1} + B_{3,3}); \\
C_{1,1} &= M_{50}; \\
C_{0,0} &= M_{50};
\end{align*}
\]

\[
\begin{align*}
M_{47} &:= \alpha(a_{1,2} + a_{3,2} + a_{0,2} + a_{2,2})(B_{2,0} + B_{2,2} + B_{2,1} + B_{2,3}); \\
C_{1,1} &= M_{50}; \\
C_{0,0} &= M_{50};
\end{align*}
\]

\[
\begin{align*}
M_{48} &:= \alpha(a_{0,3} + a_{2,3} + a_{1,3} + a_{3,3})(B_{3,0} + B_{3,2} + B_{3,1} + B_{3,3}); \\
C_{0,0} &= M_{50}; \\
C_{0,0} &= M_{50};
\end{align*}
\]
Additional Levels of Strassen Reloaded

- The general operation of one-level Strassen:

\[ M := \alpha(X+\delta Y)(V+\epsilon W); \quad C := \gamma_0 M; \quad D := \gamma_1 M; \]
\[ \gamma_0, \gamma_1, \delta, \epsilon \in \{-1, 0, 1\}. \]

- The general operation of two-level Strassen:

\[ M := \alpha(X_0+\delta_1 X_1+\delta_2 X_2+\delta_3 X_3)(V+\epsilon_1 V_1+\epsilon_2 V_2+\epsilon_3 V_3); \]
\[ C_0 := \gamma_0 M; \quad C_1 := \gamma_1 M; \quad C_2 := \gamma_2 M; \quad C_3 := \gamma_3 M; \]
\[ \gamma_r, \delta_r, \epsilon_r \in \{-1, 0, 1\}. \]

- The general operation needed to integrate \( k \) levels of Strassen is given by

\[ M := \alpha \left( \sum_{s=0}^{l_x-1} \delta_s X_s \right) \left( \sum_{t=0}^{l_y-1} \epsilon_t V_t \right); \]
\[ C_r := \gamma_r M \quad \text{for} \quad r = 0, \ldots, l_C - 1; \]
\[ \delta_i, \epsilon_i, \gamma_i \in \{-1, 0, 1\}. \]
Building blocks

BLIS framework

- A routine for packing $B_p$ into $\bar{B}_p$
  - C/Intel intrinsics

- A routine for packing $A_i$ into $\bar{A}_i$
  - C/Intel intrinsics

- A micro-kernel for updating an $m_R \times n_R$ submatrix of $C$.
  - SIMD assembly (AVX, AVX512, etc)

Adapted to general operation

- Integrate the update of multiple submatrices $C_r$ of $C$.

- Integrate the addition of multiple matrices $V_t$ into $\bar{B}_p$

- Integrate the addition of multiple matrices $X_s$ into $\bar{A}_i$

- Integrate the update of multiple submatrices of $C$. 

\[ M := \alpha \left( \sum_{s=0}^{l_X-1} \delta_s X_s \right) \left( \sum_{t=0}^{l_V-1} \epsilon_t V_t \right) ; \]
\[ C_r += \gamma_r M \text{ for } r = 0, \ldots, l_C - 1; \]
\[ \delta_i, \epsilon_i, \gamma_i \in \{-1, 0, 1\}. \]
Variations on a theme

• Naïve Strassen
  ➢ A traditional implementation with temporary buffers.

• AB Strassen
  ➢ Integrate the addition of matrices into $\vec{A}_i$ and $\vec{B}_p$.

• ABC Strassen
  ➢ Integrate the addition of matrices into $\vec{A}_i$ and $\vec{B}_p$.
  ➢ Integrate the update of multiple submatrices of $C$ in the micro-kernel.
Parallelization

- **3rd loop (along $m_C$ direction)**
  
  - Pack $V_p + εW_p → B_p$

- **2nd loop (along $n_R$ direction)**
  
  - Pack $X_i + δY_I → A_i$

- **both 3rd and 2nd loop**
  
  - $Y_J W_j X_j V_j C_j n_C n_C$

Outline

• Review of State-of-the-art GEMM in BLIS
• Strassen’s Algorithm Reloaded
• Theoretical Model and Practical Performance
• Extension to Other BLAS-3 Operation
• Extension to Other Fast Matrix Multiplication
• Conclusion
Performance Model

- Performance Metric

\[
\text{Effective GFLOPS} = \frac{2 \cdot m \cdot n \cdot k}{\text{time (in seconds)}} \cdot 10^{-9}
\]

- Total Time Breakdown

\[
T = T_a + T_m
\]

- Arithmetic Operations

- Memory Operations
Arithmetic Operations

\[ T_a = T_a^x + T_a^{A+} + T_a^{B+} + T_a^{C+} \]

- **DGEMM**
  - No extra additions
  \[ T_a = 2mnk \cdot \tau_a \]

- **One-level Strassen (ABC, AB, Naïve)**
  - 7 submatrix multiplications
  - 5 extra additions of submatrices of A and B
  - 12 extra additions of submatrices of C
  \[ T_a = \left( 7 \times 2 \frac{m \times n \times k}{2} + 5 \times 2 \frac{m \times k}{2} + 5 \times 2 \frac{k \times n}{2} + 12 \times 2 \frac{m \times n}{2} \right) \cdot \tau_a \]

- **Two-level Strassen (ABC, AB, Naïve)**
  - 49 submatrix multiplications
  - 95 extra additions of submatrices of A and B
  - 154 extra additions of submatrices of C
  \[ T_a = \left( 49 \times 2 \frac{m \times n \times k}{4} + 95 \times 2 \frac{m \times k}{4} + 95 \times 2 \frac{k \times n}{4} + 154 \times 2 \frac{m \times n}{4} \right) \cdot \tau_a \]
Memory Operations

\[ T_m = N_m^{A_x} \cdot T_m^{A_x} + N_m^{B_x} \cdot T_m^{B_x} + N_m^{C_x} \cdot T_m^{C_x} + N_m^{A_+} \cdot T_m^{A_+} + N_m^{B_+} \cdot T_m^{B_+} + N_m^{C_+} \cdot T_m^{C_+} \]

• DGEMM
  \[ T_m = (1 \cdot mk \left\lceil \frac{n}{n_c} \right\rceil + 1 \cdot nk + 1 \cdot 2\lambda mn \left\lceil \frac{k}{k_c} \right\rceil ) \cdot \tau_b \]

• One-level
  - ABC Strassen
    \[ T_m = (12 \cdot \frac{m k}{2} \cdot \left\lceil \frac{n/2}{n_c} \right\rceil + 12 \cdot \frac{n k}{2} + 12 \cdot 2\lambda \frac{m n}{2} \cdot \left\lceil \frac{k/2}{k_c} \right\rceil ) \cdot \tau_b \]
  - AB Strassen
    \[ T_m = (12 \cdot \frac{m k}{2} \cdot \left\lceil \frac{n/2}{n_c} \right\rceil + 12 \cdot \frac{n k}{2} + 7 \cdot 2\lambda \frac{m n}{2} \cdot \left\lceil \frac{k/2}{k_c} \right\rceil + 36 \cdot \frac{m n}{2} ) \cdot \tau_b \]
  - Naïve Strassen
    \[ T_m = (7 \cdot \frac{m k}{2} \cdot \left\lceil \frac{n/2}{n_c} \right\rceil + 7 \cdot \frac{n k}{2} + 7 \cdot 2\lambda \frac{m n}{2} \cdot \left\lceil \frac{k/2}{k_c} \right\rceil + 19 \cdot \frac{m k}{2} + 19 \cdot \frac{n k}{2} + 36 \cdot \frac{m n}{2} ) \cdot \tau_b \]

• Two-level
  - ABC Strassen
    \[ T_m = (194 \cdot \frac{m k}{4} \cdot \left\lceil \frac{n/4}{n_c} \right\rceil + 194 \cdot \frac{n k}{4} \cdot \frac{m n}{4} \cdot \left\lceil \frac{k/4}{k_c} \right\rceil ) \cdot \tau_b \]
  - AB Strassen
    \[ T_m = (194 \cdot \frac{m k}{4} \cdot \left\lceil \frac{n/4}{n_c} \right\rceil + 194 \cdot \frac{n k}{4} + 49 \cdot 2\lambda \frac{m n}{4} \cdot \left\lceil \frac{k/4}{k_c} \right\rceil + 462 \cdot \frac{m n}{4} ) \cdot \tau_b \]
  - Naïve Strassen
    \[ T_m = (49 \cdot \frac{m k}{4} \cdot \left\lceil \frac{n/4}{n_c} \right\rceil + 49 \cdot \frac{n k}{4} + 49 \cdot 2\lambda \frac{m n}{4} \cdot \left\lceil \frac{k/4}{k_c} \right\rceil + 293 \cdot \frac{m k}{4} + 293 \cdot \frac{n k}{4} + 462 \cdot \frac{m n}{4} ) \cdot \tau_b \]
Modeled and Actual Performance on Single Core
Observation \textbf{(Square Matrices)}

Modeled Performance

Actual Performance
Observation (Square Matrices)

Modeled Performance

Actual Performance
Observation (Square Matrices)

Modeled Performance

Actual Performance
Observation (Square Matrices)

Modeled Performance

![Graph showing modeled performance with various algorithms and their performance levels.]

Actual Performance

![Graph showing actual performance with various algorithms and their performance levels.]

Legend:
- Modeled DGEMM
- Modeled One-level ABC Strassen
- Modeled Two-level ABC Strassen
- Modeled One-level AB Strassen
- Modeled Two-level AB Strassen
- Modeled One-level Naive Strassen
- Modeled Two-level Naive Strassen

Potential Activities:
- Analyze the differences in performance between modeled and actual scenarios.
- Compare the efficiency of different algorithms in various performance metrics.
- Investigate the scalability of the algorithms as the scale of the matrices increases.

Key Points:
- The modeled performance closely follows the actual performance, indicating a high level of accuracy in the model.
- The one-level and two-level ABC Strassen algorithms show a higher performance gain compared to the AB Strassen algorithms.
- Naive Strassen algorithms are significantly less efficient than their ABC counterparts.

Next Steps:
- Further analysis could include the impact of hardware characteristics on performance.
- Exploring hybrid algorithms that combine different strategies for improved efficiency.
- Testing in more complex scenarios to assess robustness.

Overall Implications:
- The model can be refined to include factors not currently accounted for, improving its predictive power.
- Continued research in this area could lead to significant advancements in computational efficiency for large-scale matrix operations.
Observation (Square Matrices)

Modeled Performance

Actual Performance
Observation (Square Matrices)

Modeled Performance

Actual Performance

Theoretical Speedup over DGEMM

- One-level Strassen (1+14.3% speedup)
  - 8 multiplications → 7 multiplications;
- Two-level Strassen (1+30.6% speedup)
  - 64 multiplications → 49 multiplications;
Observation (Square Matrices)

- Both one-level and two-level
  - For small square matrices, **ABC Strassen** outperforms **AB Strassen**
  - For larger square matrices, this trend reverses

- Reason
  - **ABC Strassen** avoids storing M (M resides in the register)
  - **ABC Strassen** increases the number of times for updating submatrices of \( C \)
Observation (Square Matrices)

- Both one-level and two-level
  - For small square matrices, ABC Strassen outperforms AB Strassen
  - For larger square matrices, this trend reverses
- Reason
  - ABC Strassen avoids storing $M$ ($M$ resides in the register)
  - ABC Strassen increases the number of times for updating submatrices of $C$
Observation (Rank-k Update)

- What is Rank-k update?
Observation (Rank-k Update)

• Importance of Rank-k update

Importance of Rank-k update

Volker Strassen

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices $A$ and $B$ of order $n$ from the coefficients of $A$ and $B$ with less than $4.7 \cdot n^{\log_2 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order $n$, solving a system of $n$ linear equations in $n$ unknowns, computing a determinant of order $n$ etc. all requiring less than const $n^{\log_2 7}$ arithmetical operations.

This fact should be compared with the result of Krylov and Korovkin-Scherbak [1] that Gaussian elimination for solving a system of linear equations is optimal if one restricts oneself to operations upon rows and columns as a whole. We also note that Vinograd [2] modifies the usual algorithms for matrix multiplication and inversion and for solving systems of linear equations, trading roughly half of the multiplications for additions and subtractions.

It is a pleasure to thank D. Bollinger for inspiring discussions about the present subject and St. Cook and B. Parlett for encouraging me to write this paper.

2. We define algorithms $\sigma_{k,1}$ which multiply matrices of order $m \cdot 2^k$, by induction on $k$: $\sigma_{m,1}$ is the usual algorithm for matrix multiplication (requiring $m^2$ multiplications and $m^2(m-1)$ additions), $\sigma_{m,k}$ already being known, define $\sigma_{m,k+1}$ as follows.

If $A$, $B$ are matrices of order $m \cdot 2^k$ to be multiplied, write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

where $A_{1k}, B_{1k}, C_{1k}$ are matrices of order $m \cdot 2^k$. Then compute

$$\begin{align*}
I & = (A_{11}+A_{22})(B_{11}+B_{22}), \\
II & = (A_{11}+A_{22})B_{11}, \\
III & = A_{11}(B_{11}-B_{22}), \\
IV & = A_{22}(-B_{11}+B_{22}), \\
V & = (A_{11}+A_{22})B_{11}, \\
VI & = (A_{11}+A_{22})(B_{11}+B_{22}), \\
VII & = (A_{11}-A_{22})(B_{11}+B_{22}).
\end{align*}$$

3. Blocked LU with partial pivoting (getrf)

Algorithm: $[A, p] := \text{LUPIV\_BLK}(A)$

Partition

$$A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}, \quad p \rightarrow \begin{pmatrix} p_T \\ p_B \end{pmatrix}$$

where $A_{TL}$ is $0 \times 0$, $p_T$ has 0 elements

while $n(A_{TL}) < n(A)$ do

Determine block size $b$

Repartition

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix},$$

$$\begin{pmatrix} p_T \\ p_B \end{pmatrix} \rightarrow \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix}$$

where $A_{11}$ is $b \times b$, $p_1$ is $b \times 1$

$$\begin{pmatrix} (A_{11})_{p_1} \\ (A_{21})_{p_1} \end{pmatrix} := \text{LUPIV\_UNB} \left( \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} \right),$$

$$\begin{pmatrix} (A_{10})_{p_1} & (A_{12})_{p_1} \end{pmatrix} := \text{PIV} \left( p_1, \begin{pmatrix} (A_{10})_{p_1} & (A_{12})_{p_1} \\ A_{20} & A_{22} \end{pmatrix} \right).$$

$A_{22} := A_{22} - A_{21}A_{12}$

Continue with

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix},$$

$$\begin{pmatrix} p_T \\ p_B \end{pmatrix} \leftarrow \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix}$$

endwhile

Gaussian Elimination is not Optimal

* The results have been found while the author was at the Department of Statistics of the University of California, Berkeley. The author wishes to thank the National Science Foundation for their support (NSF GP-7454).
Observation (Rank-k Update)

- Importance of Rank-k update

\[
A_{21} A_{12} = A_{22}
\]
Observation (Rank-k Update)

Modeled Performance

Actual Performance

$m=n=16000$, $k$ varies, 1 core, modeled

$m=n=16000$, $k$ varies, 1 core
Observation (Rank-k Update)

Modeled Performance

Actual Performance

m=n=16000, k varies, 1 core, modeled

m=n=16000, k varies, 1 core

Effective GFLOPS (2 m n k/time)
Observation (Rank-k Update)

Modeled Performance

Actual Performance
Observation (Rank-k Update)

Modeled Performance

Actual Performance

\[ m=n=16000, \text{ } k \text{ varies, 1 core, modeled} \]

Effective GFLOPS (2m,n,k/time)

\[ k \times 10^3 \]

- Modeled DGEMM
- Modeled One-level ABC Strassen
- Modeled Two-level ABC Strassen
- Modeled One-level AB Strassen
- Modeled Two-level AB Strassen
- Modeled One-level Naive Strassen
- Modeled Two-level Naive Strassen

\[ m=n=16000, \text{ } k \text{ varies, 1 core} \]

Effective GFLOPS (2m,n,k/time)

\[ k \times 10^3 \]

- BLIS DGEMM
- MKL DGEMM
- One-level ABC Strassen
- Two-level ABC Strassen
- One-level AB Strassen
- Two-level AB Strassen
- One-level Naive Strassen
- Two-level Naive Strassen
Observation (Rank-k Update)

Modeled Performance

Actual Performance

\[ m=n=16000, \text{ k varies, 1 core, modeled} \]

\[ m=n=16000, \text{ k varies, 1 core} \]
Modeled Performance

Actual Performance

- **Reason:**
  - **ABC Strassen** avoids forming the temporary matrix $M$ explicitly in the memory ($M$ resides in register), especially important when $m, n >> k$. 

Single Node Experiment
Square Matrices

Rank-k Update

Effective GFLOPS (2-m-n-K-time)

1 core

m=k=n, 1 core

5 core

m=k=n, 5 core

10 core

m=k=n, 10 core

m=n=16000, k varies, 1 core

m=n=16000, k varies, 5 core

m=n=16000, k varies, 10 core

1 core

5 core

10 core
Many-core Experiment
Intel® Xeon Phi™ coprocessor (KNC)

m=n=15120, k varies

Effective GFLOPS (2-m.n.k/time)

- BLIS DGEMM
- MKL DGEMM
- One-level ABC Strassen

k × 10^3
Distributed Memory Experiment
m=k=n=16000 · N on N × N MPI mesh
1 MPI process per per socket

Effective GFLOPS (2·m·n·k/time)/Socket

BLIS DGEMM
MKL DGEMM
One-level ABC Strassen
Two-level ABC Strassen
One-level AB Strassen
Two-level AB Strassen
One-level Naive Strassen
Two-level Naive Strassen
Outline

• Review of State-of-the-art GEMM in BLIS
• Strassen’s Algorithm Reloaded
• Theoretical Model and Practical Performance
• Extension to Other BLAS-3 Operation
• Extension to Other Fast Matrix Multiplication
• Conclusion
Level-3 BLAS Symmetric Matrix-Matrix Multiplication (SYMM)

- Symmetric matrix-matrix multiplication (SYMM) is supported in the level-3 BLAS* interface as
  
  \[
  \text{dsymm}( \text{side}, \text{uplo}, m, n, \\
  \text{alpha}, A, \text{lda}, B, \text{ldb}, \\
  \text{beta}, C, \text{ldc} )
  \]

- SYMM computes
  
  \[
  C := \alpha AB + \beta C;
  \]

Level-3 BLAS Symmetric Matrix-Matrix Multiplication (SYMM)

\[ C := \alpha AB + \beta C; \]

- \( A \) is a symmetric matrix (square and equal to its transpose);
- Assume only the lower triangular half is stored;
- Follow the same algorithm we used for GEMM by modifying the packing routine for matrix \( A \) to account for the symmetric nature of \( A \).

**Example Matrix** \( A \)

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{bmatrix}
\]

is stored as

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 4 & 5 & 0 & 0 & 0 & 0 & 0 \\
4 & 5 & 6 & 7 & 0 & 0 & 0 & 0 \\
5 & 6 & 7 & 8 & 9 & 0 & 0 & 0 \\
6 & 7 & 8 & 9 & 10 & 11 & 0 & 0 \\
7 & 8 & 9 & 10 & 11 & 12 & 13 & 0 \\
8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{bmatrix}
\]
SYMM with One-Level Strassen

- When partitioning $A$ for one-level Strassen operations, we integrate the symmetric nature of $A$. 

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
3 & 4 & 5 & 0 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10 \\
8 & 9 & 10 & 11
\end{bmatrix}
\] 

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
3 & 4 & 5 & 0 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10 \\
8 & 9 & 10 & 11
\end{bmatrix}
\] 

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
3 & 4 & 5 & 0 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10 \\
8 & 9 & 10 & 11
\end{bmatrix}
\] 

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
3 & 4 & 5 & 0 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10 \\
8 & 9 & 10 & 11
\end{bmatrix}
\] 

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
3 & 4 & 5 & 0 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10 \\
8 & 9 & 10 & 11
\end{bmatrix}
\] 

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
3 & 4 & 5 & 0 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10 \\
8 & 9 & 10 & 11
\end{bmatrix}
\]
SYMM Implementation Platform: BLISlab*

What is BLISlab?

- A Sandbox for Optimizing GEMM that mimics implementation in BLIS
- A set of exercises that use GEMM to show how high performance can be attained on modern CPUs with hierarchical memories
- Used in Prof. Robert van de Geijn’s CS 383C Numerical Linear Algebra

Contributions

- Added DSYMM
- Applied Strassen’s algorithm to DGEMM and DSYMM
- 3 versions: Naive, AB, ABC
- Strassen improves performance in both DGEMM and DSYMM

* [https://github.com/flame/blislab](https://github.com/flame/blislab)
DSYMM Results in BLISlab

![Graphs showing DSYMM results for 1 core and 8 cores.](image)
Outline

• Review of State-of-the-art GEMM in BLIS
• Strassen’s Algorithm Reloaded
• Theoretical Model and Practical Performance
• Extension to Other BLAS-3 Operation
• Extension to Other Fast Matrix Multiplication
• Conclusion
(2,2,2) Strassen Algorithm

\[
M_0 := (A_{00} + A_{11})(B_{00} + B_{11}); \quad C_{00} += M_0; \quad C_{11} += M_0;
M_1 := (A_{10} + A_{11})B_{00}; \quad C_{10} += M_1; \quad C_{11} -= M_1;
M_2 := A_{00}(B_{01} - B_{11}); \quad C_{01} += M_2; \quad C_{11} += M_2;
M_3 := A_{11}(B_{10} - B_{00}); \quad C_{00} += M_3; \quad C_{10} += M_3;
M_4 := (A_{00} + A_{01})B_{11}; \quad C_{01} += M_4; \quad C_{00} -= M_4;
M_5 := (A_{10} - A_{00})(B_{00} + B_{01}); \quad C_{11} += M_5;
M_6 := (A_{01} - A_{11})(B_{10} + B_{11}); \quad C_{00} += M_6;
\]

\[
\begin{bmatrix}
1 & 0 & 1 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & -1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & -1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\]
(3,2,3) Fast Matrix Multiplication

\[
M_0 := (A_{00} + A_{11} + A_{22})(B_{11} + B_{12});
M_1 := (A_{00} + A_{10} + A_{21})(B_{00} - B_{12});
M_2 := (A_{00} + A_{10} - B_{01} - B_{11});
M_3 := (A_{00} - A_{10} + A_{21})(B_{02} - B_{10} + B_{12});
M_4 := (A_{00} + A_{10} + A_{11})(B_{00} + B_{01} + B_{11});
M_5 := (A_{10})(B_{00});
M_6 := (A_{10} + A_{21} + A_{20} + A_{21})(B_{10} - B_{12});
M_7 := (A_{11})(B_{10});
M_8 := (A_{10} + A_{10} + A_{20})(B_{01} + B_{02});
M_9 := (A_{10} + A_{01})(-B_{00} + B_{02});
M_{10} := (A_{21})(B_{02} + B_{12});
M_{11} := (A_{20} - A_{21})(B_{02});
M_{12} := (A_{01})(B_{00} + B_{01} + B_{10} + B_{11});
M_{13} := (A_{10} - A_{20})(B_{00} - B_{02} + B_{10} - B_{12});
M_{14} := (A_{00} + A_{01} + A_{10} - A_{11})(B_{11});
\]

\[
C_{02} := M_0^2;
C_{02} := M_1^2; C_{21} := M_2^2;
C_{00} := M_3^2; C_{01} := M_4^2; C_{11} := M_5^2;
C_{00} := M_6^2; C_{01} := M_7^2; C_{11} := M_8^2;
C_{00} := M_9^2; C_{01} := M_{10}; C_{11} := M_{11};
\]

Outline

• Review of State-of-the-art GEMM in BLIS
• Strassen’s Algorithm Reloaded
• Theoretical Model and Practical Performance
• Extension to Other BLAS-3 Operation
• Extension to Other Fast Matrix Multiplication
• Conclusion
To achieve practical high performance of Strassen’s algorithm......

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Conventional Implementations</th>
<th>Must be large</th>
<th>Our Implementations</th>
<th>Usually task parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Shape</td>
<td>Must be square</td>
<td></td>
<td></td>
<td>Can be data parallelism</td>
</tr>
</tbody>
</table>

Parallelism

No Additional Workspace

Workspace

Conventional Implementations

Our Implementations
Acknowledgement

- NSF grants ACI-1148125/1340293, CCF-1218483.
- Intel Corporation through an Intel Parallel Computing Center (IPCC).
- Access to the Maverick and Stampede supercomputers administered by TACC.

We thank Field Van Zee, Chenhan Yu, Devin Matthews, and the rest of the SHPC team (http://shpc.ices.utexas.edu) for their supports.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
Thank you!