Strassen’s Algorithm Reloaded

Jianyu Huang, Tyler M. Smith, Greg M. Henry, Robert A. van de Geijn

The University of Texas at Austin, Intel
Salt Lake City, UT
November 16th, 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 \( 4n \cdot \log_2 n \) arithmetical operations (all logarithms in this paper are for base 2. Thus \( \log_2 2.8 \approx 1 \): 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 \( n^{\log_2 5} \) arithmetical operations.

This fact should be compared with the result of Klyutev and Korovnichenko [6] 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 Winograd [7] 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. Bleiweg ris for inspiring discussions about the present subject and S. Cook and B. Fagley for encouraging me to write this paper.

2. We define algorithms \( \kappa_{n,k} \) which multiply matrices of order \( m \cdot 2^k \), by induction on \( k \): \( \kappa_{n,0} \) is the usual algorithm for matrix multiplication (requiring \( m^3 \) multiplications and \( m^2(m-1) \) additions). \( \kappa_{n,k} \) already known, define \( \kappa_{n,k+1} \) 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 the \( A_{ij}, B_{ij}, C_{ij} \) are matrices of order \( m \cdot 2^{k-1} \). Then compute

\[
\begin{align*}
I & = (A_{11} + A_{21}) (B_{11} + B_{21}), \\
II & = (A_{11} + A_{21}) B_{11}, \\
III & = (A_{12} + B_{12}) B_{11}, \\
IV & = A_{11} B_{11} + A_{21} B_{21}, \\
V & = (A_{11} + A_{21}) B_{22}, \\
VI & = (A_{11} + A_{21}) (B_{11} + B_{21}), \\
VII & = (A_{12} - A_{22}) (B_{12} + B_{22}).
\end{align*}
\]

* 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-2450).
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

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

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

**Direct Computation**

\[
\begin{align*}
C_{00} & := A_{00}B_{00} + A_{01}B_{10} + C_{00}; \\
C_{01} & := A_{00}B_{01} + A_{01}B_{11} + C_{01}; \\
C_{10} & := A_{10}B_{00} + A_{11}B_{10} + C_{10}; \\
C_{11} & := A_{10}B_{01} + A_{11}B_{11} + C_{11}; \\
\end{align*}
\]

8 multiplications, 8 additions

**Strassen’s Algorithm**

\[
\begin{align*}
M_0 & := (A_{00} + A_{11})(B_{00} + B_{11}); \\
M_1 & := (A_{10} + A_{11})B_{00}; \\
M_2 & := A_{00}(B_{01} - B_{11}); \\
M_3 & := A_{11}(B_{10} - B_{00}); \\
M_4 & := (A_{00} + A_{01})B_{11}; \\
M_5 & := (A_{10} - A_{00})(B_{00} + B_{01}); \\
M_6 & := (A_{01} - A_{11})(B_{10} + B_{11}); \\
C_{00} & := M_0 + M_3 - M_4 + M_7 + C_{00}; \\
C_{01} & := M_2 + M_4 + C_{01}; \\
C_{10} & := M_1 + M_3 + C_{10}; \\
C_{11} & := M_0 - M_1 + M_2 + M_5 + C_{11}.
\end{align*}
\]

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
\[
(7/8)^{\log_2(n)} 2n^3 = n^{\log_2(7/8)} 2n^3 \approx 2n^{2.807} \text{ flops.}
\]
Multi-level Strassen’s Algorithm (In theory)

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

- 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;

Speedup of Strassen over DGEMM for Square Matrices
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;

\[
M_0 := (A_{00} + A_{11})(B_{00} + B_{11});
\]
\[
M_1 := (A_{10} + A_{11})B_{00};
\]
\[
M_2 := A_{00}(B_{01} - B_{11});
\]
\[
M_3 := A_{11}(B_{10} - B_{00});
\]
\[
M_4 := (A_{00} + A_{01})B_{11};
\]
\[
M_5 := (A_{10} - A_{00})(B_{00} + B_{01});
\]
\[
M_6 := (A_{01} - A_{11})(B_{10} + B_{11});
\]
\[
C_{00} += M_0 + M_3 - M_4 + M_6
\]
\[
C_{01} += M_2 + M_4
\]
\[
C_{10} += M_1 + M_3
\]
\[
C_{11} += M_0 - M_1 + M_2 + M_5
\]
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;
- $d$-level Strassen ($n^3/n^{2.803}$ speedup)
  - Numerical unstable; Not achievable
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><strong>Matrix Size</strong></td>
<td>Must be large</td>
<td></td>
</tr>
<tr>
<td><strong>Matrix Shape</strong></td>
<td>Must be square</td>
<td></td>
</tr>
<tr>
<td><strong>No Additional Workspace</strong></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td><strong>Parallelism</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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><strong>Matrix Size</strong></td>
<td>Must be large</td>
<td></td>
</tr>
<tr>
<td><strong>Matrix Shape</strong></td>
<td>Must be square</td>
<td></td>
</tr>
<tr>
<td><strong>No Additional Workspace</strong></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td><strong>Parallelism</strong></td>
<td>Usually task parallelism</td>
<td></td>
</tr>
</tbody>
</table>
To achieve practical high performance of Strassen’s algorithm......

### Conventional Implementations
- Matrix Size: Must be large
- Matrix Shape: Must be square
- No Additional Workspace
- Parallelism: Usually task parallelism

### Our Implementations
- No Additional Workspace
- Parallelism: Can be data parallelism
Outline

- Standard Matrix-matrix multiplication
- Strassen’s Algorithm Reloaded
- Theoretical Model and Analysis
- Performance Experiments
- Conclusion
Level-3 BLAS Matrix-Matrix Multiplication (GEMM)

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

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

- 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.
  - Chenhan 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*.
  - Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn. “Strassen’s Algorithm Reloaded.” In *SC’16*.
GotoBLAS algorithm for GEMM in BLIS

\[
\begin{array}{c}
m \downarrow C + m \downarrow A \times k \downarrow B \\
\end{array}
\]

GotoBLAS algorithm for GEMM in BLIS

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

m & \begin{array}{c}
C \quad n \quad \leftarrow \\
A \quad k \quad \times \\
B \quad n \quad \rightarrow
\end{array}
\end{align*}
\]

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

GotoBLAS algorithm for GEMM in BLIS

\[
\begin{align*}
\text{Loop 5} & \quad \text{for } j_c = 0 : n - 1 \text{ steps of } n_c \\
& \quad J_c = j_c : j_c + n_c - 1 \\
\text{Loop 4} & \quad \text{for } p_c = 0 : k - 1 \text{ steps of } k_c \\
& \quad P_c = p_c : p_c + k_c - 1 \\
& \quad B(P_c, J_c) \rightarrow B_{p_c}
\end{align*}
\]

GotoBLAS algorithm for GEMM in BLIS

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 $k_c$
\[ P_c = p_c : p_c + k_c - 1 \]
\[ B(P_c, J_c) \rightarrow B_p \]

Loop 3  for $i_c = 0 : m - 1$ steps of $m_c$
\[ I_c = i_c : i_c + m_c - 1 \]
\[ A(I_c, P_c) \rightarrow A_i \]

endfor
endfor
endfor

GotoBLAS algorithm for GEMM in BLIS

\[
\begin{align*}
\text{Loop 5} & \quad \text{for} \ j_c = 0 : n - 1 \text{ steps of } n_c \\
J_c &= j_c : j_c + n_c - 1 \\
\text{Loop 4} & \quad \text{for} \ p_c = 0 : k - 1 \text{ steps of } k_c \\
P_c &= p_c : p_c + k_c - 1 \\
B(P_c, J_c) &\rightarrow B_p \\
\text{Loop 3} & \quad \text{for} \ i_c = 0 : m - 1 \text{ steps of } m_c \\
I_c &= i_c : i_c + m_c - 1 \\
A(I_c, P_c) &\rightarrow A_i \\
&\quad \text{// macro-kernel} \\
\text{Loop 2} & \quad \text{for} \ j_r = 0 : n_c - 1 \text{ steps of } n_r \\
J_r &= j_r : j_r + n_r - 1 \\
\end{align*}
\]

GotoBLAS algorithm for GEMM in BLIS

\[
\begin{align*}
C_i & \leftarrow \begin{array}{c}
\begin{array}{c}
\text{5th loop around micro-kernel}
\end{array}
\end{array} \\
\begin{array}{c}
A
\end{array} & \leftarrow \begin{array}{c}
\begin{array}{c}
\text{4th loop around micro-kernel}
\end{array}
\end{array} \\
B_j & \leftarrow \begin{array}{c}
\begin{array}{c}
\text{3rd loop around micro-kernel}
\end{array}
\end{array} \\
\begin{array}{c}
\text{Pack } B_p \rightarrow \tilde{B}_p
\end{array} & \leftarrow \begin{array}{c}
\begin{array}{c}
\text{2nd loop around micro-kernel}
\end{array}
\end{array} \\
\begin{array}{c}
\text{Pack } A_i \rightarrow \tilde{A}_i
\end{array} & \leftarrow \begin{array}{c}
\begin{array}{c}
\text{1st loop around micro-kernel}
\end{array}
\end{array} \\
\text{Update } C_{ij}
\end{align*}
\]

\[
\begin{align*}
C_i & = A_i \times k \times B_p \\
& = \begin{array}{c}
\text{Loop 1}
\end{array} \\
& \quad \text{for } i_r = 0 : m_r - 1 \text{ steps of } m_r \\
& \quad \begin{array}{c}
\text{Loop 2}
\end{array} \\
& \quad \quad \text{for } j_r = 0 : n_r - 1 \text{ steps of } n_r \\
& \quad \quad \quad \text{Loop 3} \\
& \quad \quad \quad \quad \text{for } i_c = 0 : m_c - 1 \text{ steps of } m_c \\
& \quad \quad \quad \quad \quad \text{Loop 4} \\
& \quad \quad \quad \quad \quad \quad \text{for } p_c = 0 : k - 1 \text{ steps of } k_c \\
& \quad \quad \quad \quad \quad \quad \quad \text{Loop 5} \\
& \quad \quad \quad \quad \quad \quad \quad \quad \text{for } j_c = 0 : n - 1 \text{ steps of } n_c \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{endfor}
\end{align*}
\]

\[\begin{array}{c}
\text{endfor}
\end{array} \quad \begin{array}{c}
\text{endfor}
\end{array} \quad \begin{array}{c}
\text{endfor}
\end{array} \quad \begin{array}{c}
\text{endfor}
\end{array} \quad \begin{array}{c}
\text{endfor}
\end{array}
\]

GotoBLAS algorithm for GEMM in BLIS

Loop 5 \( j_c = 0 : n - 1 \) steps of \( n_c \)
\[ J_c = j_c : j_c + n_c - 1 \]
Loop 4 \( p_c = 0 : k - 1 \) steps of \( k_c \)
\[ P_c = p_c : p_c + k_c - 1 \]
\[ B(P_c, J_c) \rightarrow \tilde{B}_p \]
Loop 3 \( i_c = 0 : m - 1 \) steps of \( m_c \)
\[ I_c = i_c : i_c + m_c - 1 \]
\[ A(I_c, P_c) \rightarrow \hat{A}_i \]
// macro-kernel
Loop 2 \( j_r = 0 : n_c - 1 \) steps of \( n_r \)
\[ J_r = j_r : j_r + n_r - 1 \]
Loop 1 \( i_r = 0 : m_c - 1 \) steps of \( m_r \)
\[ I_r = i_r : i_r + m_r - 1 \]
// micro-kernel
Loop 0 \( p_r = 0 : p_c - 1 \) steps of \( 1 \)
\[ C_C(I_r, J_r) = \alpha \hat{A}_i(I_r, p_r) \tilde{B}_p(p_r, J_r) \]
endfor
endfor
endfor
endfor
endfor

GotoBLAS algorithm for GEMM in BLIS

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

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

Loop 3 \( \text{for } i_c = 0 : m-1 \) steps of \( m_c \)
\( I_c = i_c : i_c + m_c - 1 \)
\( A(I_c, P_c) \rightarrow A_i \)
// macro-kernel

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

Loop 1 \( \text{for } i_r = 0 : m_c - 1 \) steps of \( m_r \)
\( I_r = i_r : i_r + m_r - 1 \)
//micro-kernel
\( C_c(I_r, J_r) \rightarrow \alpha A_i(I_r, p_r) \tilde{B}_p(p_r, J_r) \)
endfor
endfor
endfor
endfor
endfor
endfor

Outline

• Standard Matrix-matrix multiplication
• **Strassen’s Algorithm Reloaded**
• Theoretical Model and Analysis
• Performance Experiments
• Conclusion
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:

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

\[ M := \alpha(X + \delta Y)(V + \varepsilon W) \]
\[ C += \gamma_0 M; \quad D += \gamma_1 M; \]
\[ \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}. \]
High-performance implementation of the general operation?

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

*[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 \( n_c \)

\[ \mathcal{J}_c = j_c : j_c + n_c - 1 \]

*Jianyu Huang*, Tyler Smith, Greg Henry, and Robert van de Geijn.

“Strassen’s Algorithm Reloaded.” In *SC’16.*
$M := \alpha (X + \delta Y)(V + \varepsilon W); \quad C := \gamma_0 M; \quad D := \gamma_1 M; \quad \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}.$

Loop 5 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 $k_c$
\[ P_c = p_c : p_c + k_c - 1 \]
\[ V(P_c, J_c) + \varepsilon W(P_c, J_c) \rightarrow \tilde{B}_p \]

*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 $k_c$

$P_c = p_c : p_c + k_c - 1$

$V(P_c, J_c) + \epsilon W(P_c, J_c) \rightarrow \tilde{B}_p$

Loop 3: for $i_c = 0 : m - 1$ steps of $m_c$

$I_c = i_c : i_c + m_c - 1$

$X(I_c, P_c) + \delta Y(I_c, P_c) \rightarrow \tilde{A}_i$

\textit{endfor}

\textit{endfor}

\textit{endfor}

*Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn.

"Strassen's Algorithm Reloaded." In SC'16.
$M := \alpha(X + \delta Y)(V + \epsilon W)$; $C := \gamma_0 M$; $D := \gamma_1 M$; $\gamma_0, \gamma_1, \delta, \epsilon \in \{-1, 0, 1\}$.

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 $k_c$

\[ p_c = p_c : p_c + k_c - 1 \]

\[ V(P_c, J_c) + \epsilon W(P_c, J_c) \rightarrow \tilde{B}_p \]

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

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

\[ M_r(\mathcal{I}_r, \mathcal{J}_r) := \tilde{A}_i(\mathcal{I}_r, p_r) \tilde{B}_p(p_r, \mathcal{J}_r) \]
endfor

C(\mathcal{I}_r + i_c, \mathcal{J}_r + j_c) := \alpha \gamma_0 M_r(\mathcal{I}_r, \mathcal{J}_r)
D(\mathcal{I}_r + i_c, \mathcal{J}_r + j_c) := \alpha \gamma_1 M_r(\mathcal{I}_r, \mathcal{J}_r)

endfor

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

\[ \mathcal{I}_r = i_r : i_r + m_r - 1 \]

// micro-kernel

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

\[ \mathcal{J}_r = j_r : j_r + n_r - 1 \]

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

\[ \mathcal{I}_c = i_c : i_c + m_c - 1 \]

\[ X(\mathcal{I}_c, \mathcal{P}_c) + \delta Y(\mathcal{I}_c, \mathcal{P}_c) \rightarrow \tilde{A}_i \]
// macro-kernel

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

\[ \mathcal{P}_c = p_c : p_c + k_c - 1 \]

\[ V(\mathcal{P}_c, \mathcal{J}_c) + \varepsilon W(\mathcal{P}_c, \mathcal{J}_c) \rightarrow \tilde{B}_p \]

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

\[ \mathcal{J}_c = j_c : j_c + n_c - 1 \]

Update \( C_{ij}, D_{ij} \)

---

*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

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

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

(Continue)

| $M_0 := \alpha(A_{0,0} + A_{2,2} + A_{1,1} + A_{3,3})(B_0,0 + B_2,2 + B_1,1 + B_3,3)$ | $C_{0,0} += M_0$ | $C_{1,1} += M_0$ | $C_{2,2} += M_0$ | $C_{3,3} += M_0$ |
| $M_1 := \alpha(A_{1,0} + A_{3,2} + A_{1,1} + A_{3,3})(B_0,0 + B_2,2)$ | $C_{1,0} += M_1$ | $C_{1,1} -= M_1$ | $C_{3,2} += M_1$ | $C_{3,3} -= M_1$ |
| $M_2 := \alpha(A_{0,0} + A_{2,2})(B_0,1 + B_2,3 + B_1,1 + B_3,3)$ | $C_{0,1} += M_2$ | $C_{1,1} += M_2$ | $C_{2,3} += M_2$ | $C_{3,3} += M_2$ |
| $M_3 := \alpha(A_{1,1} + A_{3,3})(B_{1,0} + B_3,2 + B_0,0 + B_2,2)$ | $C_{0,0} += M_3$ | $C_{1,0} += M_3$ | $C_{2,2} += M_3$ | $C_{3,2} += M_3$ |
| $M_4 := \alpha(A_{0,0} + A_{2,2} + A_{0,1} + A_{2,3})(B_1,1 + B_3,3)$ | $C_{0,0} -= M_4$ | $C_{0,1} += M_4$ | $C_{2,2} -= M_4$ | $C_{2,3} += M_4$ |
| $M_5 := \alpha(A_{1,0} + A_{3,2} + A_{0,0} + A_{2,2})(B_0,0 + B_2,2 + B_0,1 + B_2,3)$ | $C_{1,1} += M_5$ | $C_{3,3} += M_5$ |
| $M_6 := \alpha(A_{0,1} + A_{2,3} + A_{1,1} + A_{3,3})(B_{1,0} + B_3,2 + B_1,1 + B_3,3)$ | $C_{0,0} += M_6$ | $C_{2,2} += M_6$ |
| $M_7 := \alpha(A_{2,0} + A_{2,2} + A_{3,1} + A_{3,3})(B_0,0 + B_1,1)$ | $C_{2,0} += M_7$ | $C_{3,1} += M_7$ | $C_{3,3} -= M_7$ |
| $M_8 := \alpha(A_{3,0} + A_{3,2} + A_{1,3} + A_{3,3})(B_0,0)$ | $C_{3,0} += M_8$ | $C_{3,2} -= M_8$ | $C_{3,3} += M_8$ |
| $M_9 := \alpha(A_{2,0} + A_{2,2})(B_0,1 + B_1,1)$ | $C_{2,1} += M_9$ | $C_{3,1} += M_9$ | $C_{3,3} -= M_9$ |
| $M_{10} := \alpha(A_{3,1} + A_{3,3})(B_{1,0} + B_0,0)$ | $C_{2,0} += M_{10}$ | $C_{3,0} += M_{10}$ | $C_{3,2} -= M_{10}$ |

General operation for two-level Strassen:

$M := \alpha(X_0 + X_1 + X_2 + X_3)(V + V_1 + V_2 + V_3)$

$\gamma_i, \delta_i, \epsilon_i \in \{-1, 0, 1\}$. 

$C_0 += \gamma_0 M$  
$C_1 += \gamma_1 M$  
$C_2 += \gamma_2 M$  
$C_3 += \gamma_3 M$
Additional Levels of Strassen Reloaded

• The general operation of one-level Strassen:

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

• The general operation of two-level Strassen:

\[ M := \alpha (X_0+\delta_1 X_1+\delta_2 X_2+\delta_3 X_3)(V+\varepsilon_1 V_1+\varepsilon_2 V_2+\varepsilon_3 V_3); \]
\[ C_0 += \gamma_0 M; \quad C_1 += \gamma_1 M; \quad C_2 += \gamma_2 M; \quad C_3 += \gamma_3 M; \]
\[ \gamma_i, \delta_i, \varepsilon_i \in \{-1, 0, 1\}. \]

• The general operation needed to integrate k levels of Strassen is given by

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

BLIS framework

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

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

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

Adapted to general operation

- 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 $C$. 

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

- **Naïve Strassen**
  - A traditional implementation with temporary buffers.
- **AB Strassen**
  - Integrate the addition of matrices into $\overline{A}_i$ and $\overline{B}_p$.
- **ABC Strassen**
  - Integrate the addition of matrices into $\overline{A}_i$ and $\overline{B}_p$.
  - Integrate the update of multiple submatrices of $C$ in the micro-kernel.
Parallelization

- 3rd loop (along $m_C$ direction)

- 2nd loop (along $n_R$ direction)

- both 3rd and 2nd loop

Outline

- Standard Matrix-matrix multiplication
- Strassen’s Algorithm Reloaded
- Theoretical Model and Analysis
- Performance Experiments
- 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

\[ T_a = T_{aX} + T_{aA^+} + T_{aB^+} + T_{aC^+} \]

- **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
  \[ T_a = \left( 7 \times 2^\frac{m}{2} \times \frac{n}{2} \times \frac{k}{2} + 5 \times 2^\frac{m}{2} \times \frac{n}{2} + 12 \times 2^\frac{m}{2} \times \frac{n}{2} \right) \cdot \tau_a \]

- **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
  \[ T_a = \left( 49 \times 2^\frac{m}{4} \times \frac{n}{4} + 95 \times 2^\frac{m}{4} \times \frac{n}{4} + 154 \times 2^\frac{m}{4} \times \frac{n}{4} \right) \cdot \tau_a \]
Memory Operations

\[ T_m = N_m^{A_\times} \cdot T_m^{A_\times} + N_m^{B_\times} \cdot T_m^{B_\times} + N_m^{C_\times} \cdot T_m^{C_\times} + N_m^{A_+} \cdot T_m^{A_+} + N_m^{B_+} \cdot T_m^{B_+} + N_m^{C_+} \cdot T_m^{C_+} \]

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

- **Two-level**
  - **ABC Strassen**
    \[ T_m = (194 \cdot \frac{m k}{4} \left\lceil \frac{n/4}{n_c} \right\rceil + 194 \cdot \frac{n k}{4} + 154 \cdot 2\lambda mn \left\lceil \frac{k/4}{k_c} \right\rceil) \cdot \tau_b \]
  - **AB Strassen**
    \[ T_m = (194 \cdot \frac{m k}{4} \left\lceil \frac{n/4}{n_c} \right\rceil + 194 \cdot \frac{n k}{4} + 49 \cdot 2\lambda mn \left\lceil \frac{k/4}{k_c} \right\rceil + 462 \cdot \frac{m n}{4}) \cdot \tau_b \]
  - **Naïve Strassen**
    \[ T_m = (49 \cdot \frac{m k}{4} \left\lceil \frac{n/4}{n_c} \right\rceil + 49 \cdot \frac{n k}{4} + 49 \cdot 2\lambda mn \left\lceil \frac{k/4}{k_c} \right\rceil + 293 \cdot \frac{m k}{4} + 293 \cdot \frac{n k}{4} + 462 \cdot \frac{m n}{4}) \cdot \tau_b \]
Modeled and Actual Performance on Single Core
Observation (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

Actual Performance
Observation **(Square Matrices)**

**Modeled Performance**

![Graph showing modeled performance](image1)

**Actual Performance**

![Graph showing actual performance](image2)
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?

\[ C \leftarrow A \times B \]
Observation \( (\text{Rank-k Update}) \)

- Importance of Rank-k update

\[
\begin{align*}
A_{21} & \quad A_{12} \\
A_{22} & \quad A_{12}
\end{align*}
\]

Blocked LU with partial pivoting (\texttt{getrf})

\[
\begin{align*}
\text{Algorithm: } [A, p] & := \text{LUPIV.BLK}(A) \\
\text{Partition: } A & \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}, \quad p \rightarrow \begin{pmatrix} p_T \\ p_B \end{pmatrix} \\
\text{where } & A_{TL} \text{ is } 0 \times 0, \text{ } p_T \text{ has } 0 \text{ elements} \\
\text{while } & n(A_{TL}) < n(A) \text{ do} \\
\text{Determine block size } b \\
\text{Repartition: } & \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}, \\
\text{where } & A_{11} = b \times b, \text{ } p_1 = b \times 1 \\
\text{endwhile}
\end{align*}
\]

\* 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 \( 4.7 \cdot n^{m+1} \) arithmetical operations (all logarithms in this paper are for base \( 2 \), thus \( \log 2 = 0.8 \); 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 \( const \cdot n^{m+1} \) arithmetical operations.

This fact should be compared with the result of Klyuev 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. Biebling for inspiring discussions about the present subject and St. Cook and B. Parlett for encouraging me to write this paper.

2. We define algorithms \( \alpha_{m,k} \) which multiply matrices of order \( m^2 \) by induction on \( k \); \( \alpha_{m,0} \) is the usual algorithm for matrix multiplication (requiring \( m^3 \) multiplications and \( m^2(m-1) \) additions), \( \alpha_{m,k} \) already being known, define \( \alpha_{m,k+1} \) as follows:

If \( A, B \) are matrices of order \( m^2 \times m^2 \) 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 the \( A_{ij}, B_{ij}, C_{ij} \) are matrices of order \( m \times m \). Then compute

\[
\begin{align*}
1 & = (A_{11} + A_{21})(B_{11} + B_{21}) \\
2 & = (A_{11} + A_{21})B_{11} \\
3 & = A_{11}(B_{11} - B_{21}) \\
4 & = A_{21}(-B_{11} + B_{21}) \\
5 & = (A_{11} + A_{21})B_{21} \\
6 & = (A_{11} + A_{21})(B_{12} + B_{22}) \\
7 & = (A_{12} - A_{22})(B_{21} + B_{22})
\end{align*}
\]

\* 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-7456).
Observation (Rank-k Update)

- Importance of Rank-k update

\[ A_{21} + A_{12} = A_{22} \times \]
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

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

Effective GFLOPS (2\(m\cdot n\cdot k/\text{time})$

$k \times 10^3$

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

Effective GFLOPS (2\(m\cdot n\cdot k/\text{time})$

$k \times 10^3$
**Observation (Rank-k Update)**

**Modeled Performance**

**Actual Performance**

---

**Graphs:**

- **Left Graph:**
  - Title: *m=n=16000, k varies, 1 core, modeled*
  - Data: *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*
  - Y-axis: Effective GFLOPS (2 m.n.k/time)
  - X-axis: k (x 10^3)

- **Right Graph:**
  - Title: *m=n=16000, k varies, 1 core*
  - Data: *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*
  - Y-axis: Effective GFLOPS (2 m.n.k/time)
  - X-axis: k (x 10^3)
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

\[ m=n=16000, \text{ } k \text{ varies, } \text{1 core, modeled} \]
**Observation** (Rank-k Update)

**Modeled Performance**

**Actual Performance**

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

- Standard Matrix-matrix multiplication
- Strassen’s Algorithm Reloaded
- Theoretical Model and Analysis
- Performance Experiments
- Conclusion
Single Node Experiment
Square Matrices

Rank-
k Update
Many-core Experiment
Intel® Xeon Phi™ coprocessor (KNC)
Distributed Memory Experiment
m=k=n=16000 \cdot N \text{ on } N \times N \text{ MPI mesh}

1 MPI process per socket

Effective GFLOPS (2 \cdot m \cdot n \cdot k/\text{time})/Socket
$m=k=n=16000 \cdot N$ on $N \times N$ MPI mesh
1 MPI process per socket

Effective GFLOPS $(2\cdot m\cdot n\cdot k$/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

• Standard Matrix-matrix multiplication
• Strassen’s Algorithm Reloaded
• Theoretical Model and Analysis
• Performance Experiments
• Conclusion
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><strong>Matrix Size</strong></td>
<td>Must be large</td>
<td>![Emoji Smiley]</td>
</tr>
<tr>
<td><strong>Matrix Shape</strong></td>
<td>Must be square</td>
<td>![Emoji Smiley]</td>
</tr>
<tr>
<td><strong>No Additional</strong></td>
<td>![Emoji Cross]</td>
<td>![Emoji Checkmark]</td>
</tr>
<tr>
<td><strong>Workspace</strong></td>
<td>![Emoji Sad]</td>
<td>![Emoji Smiley]</td>
</tr>
<tr>
<td><strong>Parallelism</strong></td>
<td>Usually task parallelism</td>
<td>Can be data parallelism</td>
</tr>
</tbody>
</table>
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!