Communication-Optimal Parallel Algorithm for Strassen's Matrix Multiplication




Grey Ballard, James Demmel, Ben Lipshitz, Oded Schwartz

Summary

- Our algorithm is communication optimal
  - matches the recently proved communication lower bounds [1]
  - moves asymptotically less data than all existing algorithms
- Our implementation is faster
  - than any classical algorithm can be
  - than any Strassen implementation we are aware of
- With our algorithm, Strassen's matrix multiplication is faster than classical
  - not just computation, but also **communication**
  - not just in theory, but also **in practice**
  - not just sequentially, but also **in parallel**

Asymptotics: Lower Bounds and Algorithms

Classical	Flops	Bandwidth	Latency
Lower Bound [4]	$\frac{n^3}{P}$	$\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$	$\max\left\{\frac{n^3}{PM^{3/2}}, 1\right\}$
Cannon [2]	$\frac{n^3}{P}$	$\frac{n^2}{P^{1/2}}$	$P^{1/2}$
2.5D [6]	$\frac{n^3}{P}$	$\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$	$\frac{n^3}{PM^{3/2}} + \log P$

Strassen	Flops	Bandwidth	Latency
Lower Bound [1]	$\frac{n^\omega}{P}$	$\max\left\{\frac{n^\omega}{PM^{\omega/2-1}}, \frac{n^2}{P^{2/\omega}}\right\}$	$\max\left\{\frac{n^\omega}{PM^{\omega/2}}, 1\right\}$
Cannon-Strassen [5]	$\frac{n^\omega}{P^{(\omega-1)/2}}$	$\frac{n^2}{P^{1/2}}$	$P^{1/2}$
Strassen-Cannon [3, 5]	$\left(\frac{7}{8}\right)^\ell \frac{n^3}{P}$	$\left(\frac{7}{4}\right)^\ell \frac{n^2}{P^{1/2}}$	$7^\ell P^{1/2}$
New Algorithm	$\frac{n^\omega}{P}$	$\max\left\{\frac{n^\omega}{PM^{\omega/2-1}}, \frac{n^2}{P^{2/\omega}}\right\}$	$\max\left\{\frac{n^\omega}{PM^{\omega/2}}\log P, \log P\right\}$

n = Matrix dimension P = Number of processors
M = Local memory size ℓ = Number of Strassen steps taken
ω = Exponent of matrix multiplication. $\log_2 7 \approx 2.81$ for Strassen

- Architectural implications
  - Strassen reduces both computation and communication
  - To remain compute bound: $\beta \leq \gamma M^{\omega/2-1}$ vs. $\beta \leq \gamma M^{1/2}$ for classical
    - ◇ β is the inverse bandwidth, γ is time per flop

Strassen-Winograd Algorithm

- Requires 7 multiplies and 15 additions for 2×2 matrix multiplication
- Requires $O(n^\omega)$ flops for $n \times n$ matrix multiplication
- Hidden constant is better than Strassen's original algorithm

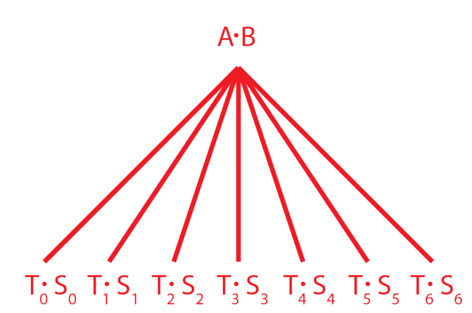
$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = A \cdot B = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

$$\begin{array}{llll}
T_0 = A_{11} & S_0 = B_{11} & Q_0 = T_0 \cdot S_0 & U_1 = Q_0 + Q_3 \\
T_1 = A_{12} & S_1 = B_{21} & Q_1 = T_1 \cdot S_1 & U_2 = U_1 + Q_4 \\
T_2 = A_{21} + A_{22} & S_2 = B_{12} + B_{11} & Q_2 = T_2 \cdot S_2 & U_3 = U_1 + Q_2 \\
T_3 = T_2 - A_{12} & S_3 = B_{22} - S_2 & Q_3 = T_3 \cdot S_3 & C_{11} = Q_0 + Q_1 \\
T_4 = A_{11} - A_{21} & S_4 = B_{22} - B_{12} & Q_4 = T_4 \cdot S_4 & C_{12} = U_3 + Q_5 \\
T_5 = A_{12} + T_3 & S_5 = B_{22} & Q_5 = T_5 \cdot S_5 & C_{21} = U_2 - Q_6 \\
T_6 = A_{22} & S_6 = S_3 - B_{21} & Q_6 = T_6 \cdot S_6 & C_{22} = U_2 + Q_2
\end{array}$$

The Parallel Algorithm

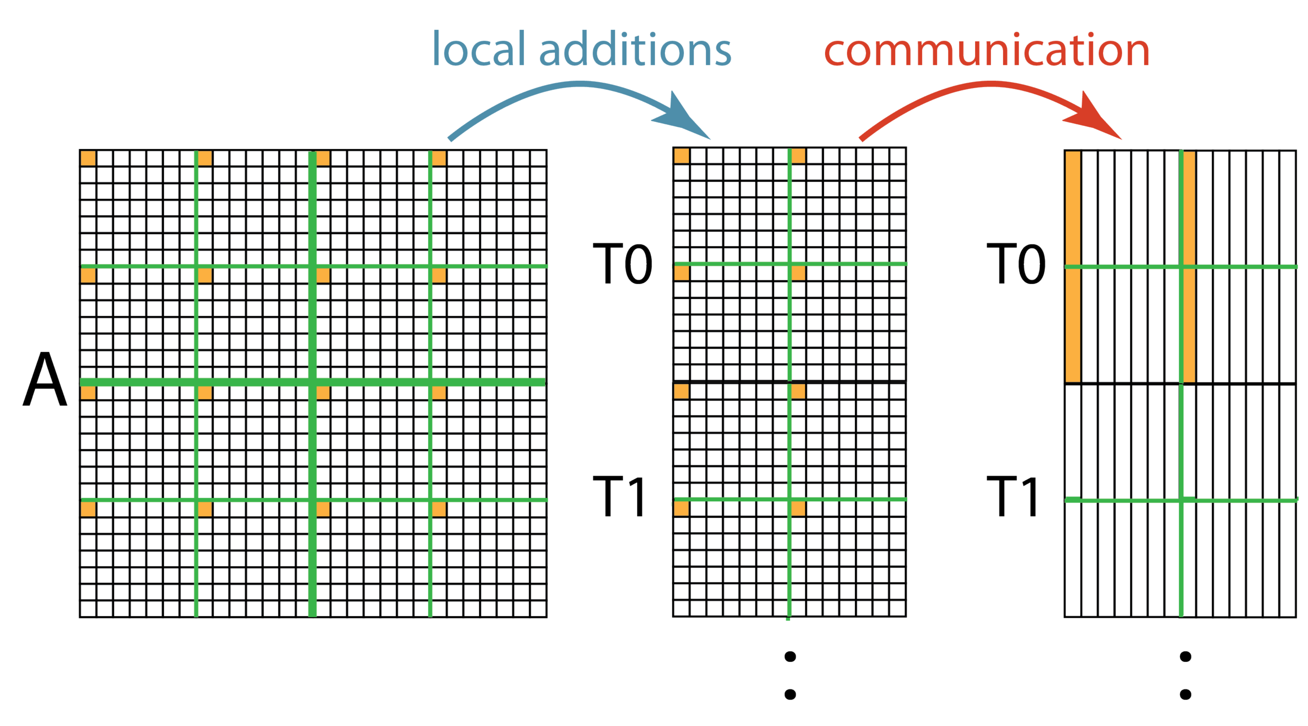
- Key parallelization decision is to choose how to compute 7 subproblems
  - breadth first search ordering or depth first search ordering
- Independent decision can be made at each level of recursion tree

Breadth First Search (BFS) Step

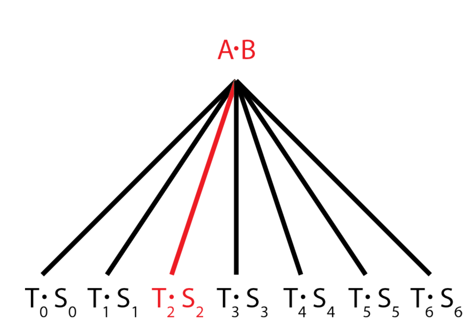


A BFS level of the recursion tree

- Runs all 7 multiplies in parallel
  - each uses $P/7$ processors
- Requires $7/4$ as much extra memory
- Requires communication
- Reduces future communication

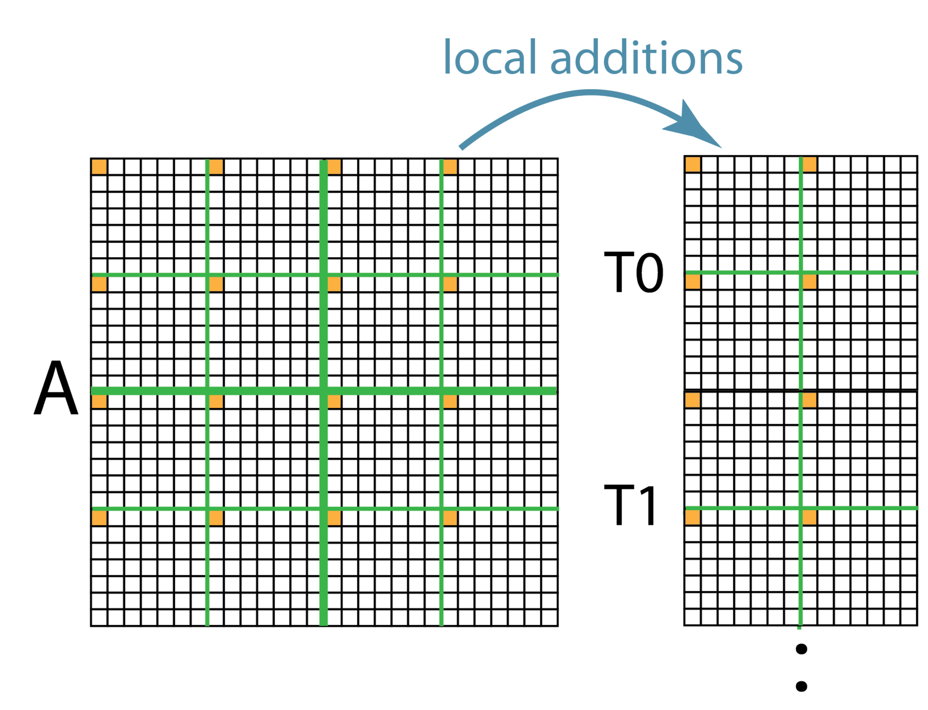


Depth First Search (DFS) Step



A DFS level of the recursion tree

- Runs all 7 multiplies sequentially
  - each uses all P processors
- Requires $1/4$ as much extra memory
- No immediate communication
- Increases future communication



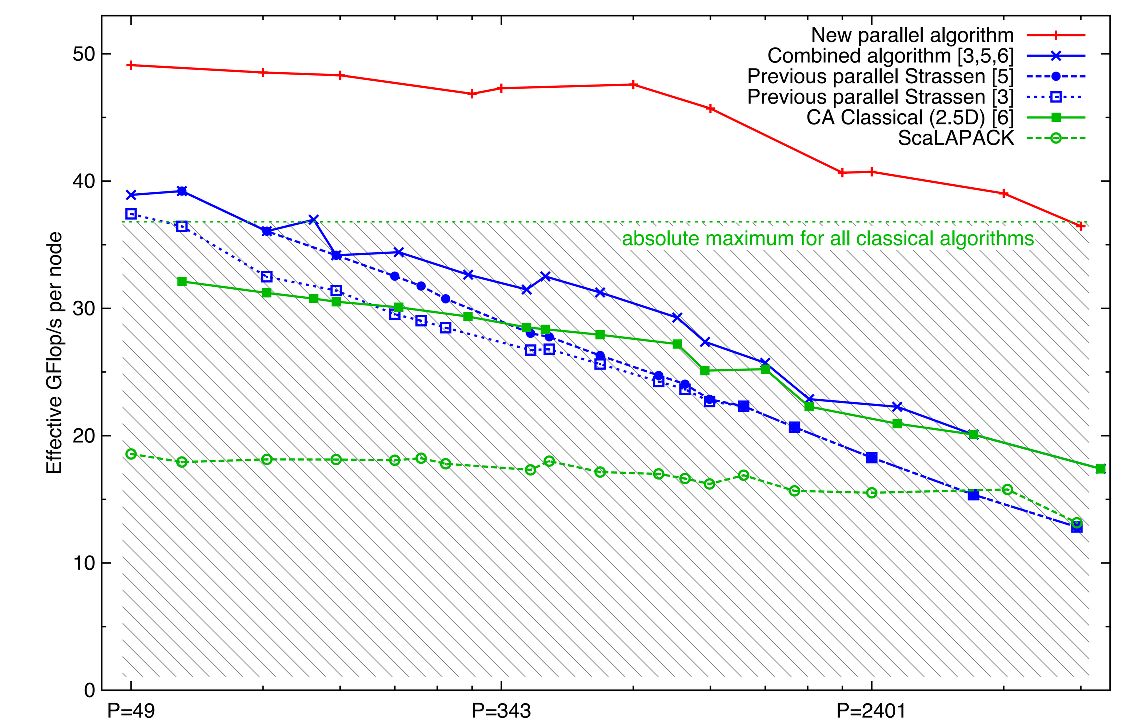
Global Schemes

- Unlimited memory: do $k = \log_7 P$ BFS steps, then local computation
  - requires $O\left(\frac{n^2}{P^{2/\omega}}\right)$ local memory footprint
- Limited memory: do $\ell = \log_2 \frac{3n}{P^{1/\omega}M^{1/2}}$ DFS steps, then k BFS steps, then local computation
  - requires $O(M)$ local memory footprint

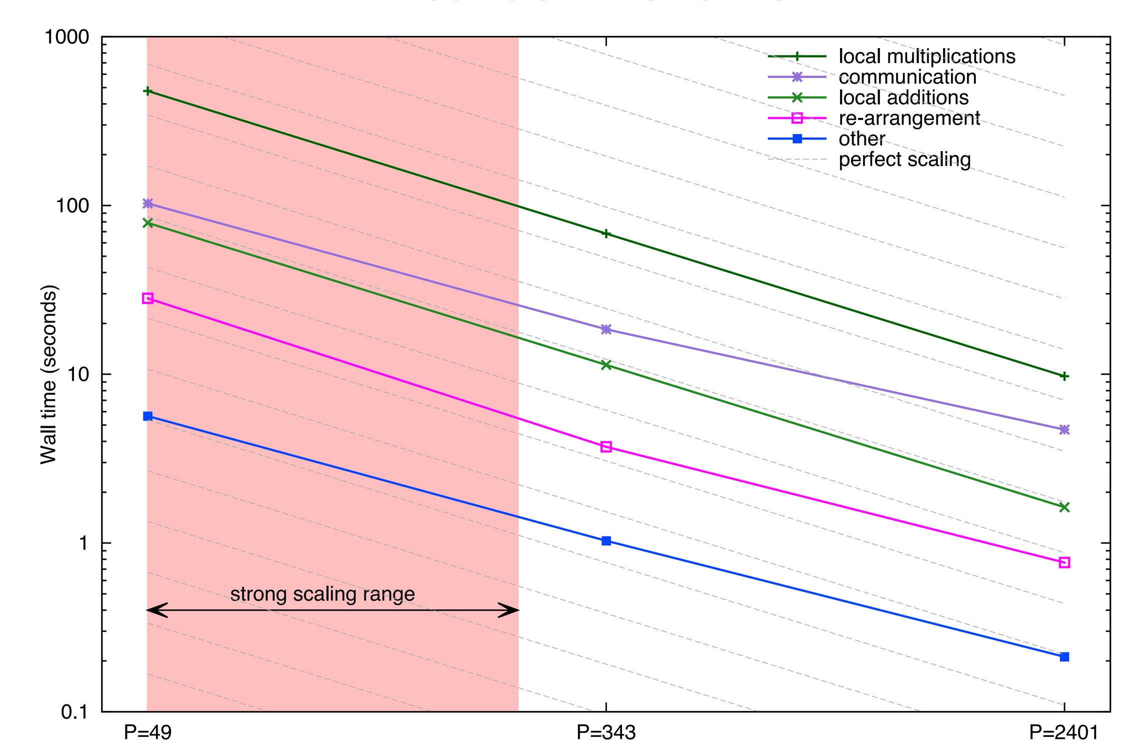
Performance Data

Performance on Franklin (Cray XT4)

$n = 94080$



Breakdown of time



Perfect Strong Scaling Range

- Within range, both flops and communication scale linearly with P
- For largest problem that fits on P_0 processors, ranges up to $P_0^{\omega/2}$ processors
- For $P > P_0^{\omega/2}$, communication can no longer scale perfectly

Implementation Details

- Interleaving BFS and DFS
- Data Layout
- Local shared memory Strassen
- Running on $P = m \cdot 7^k$
- Hybrid BFS steps for $m > 1$
- Hiding communication

Open Problems

- Analyze contention and optimize for it
- An efficient algorithm for arbitrary number of processors
- Fast parallel dense linear algebra: LU, QR, etc.
- Other practical fast matrix multiplication algorithms

References

[1] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 1–12, New York, NY, USA, 2011. ACM.

[2] L. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, Bozeman, MN, 1969.

[3] B. Grayson, A. Shah, and R. van de Geijn. A high performance parallel Strassen implementation. In *Parallel Processing Letters, Vol 6*, pages 3–12, 1995.

[4] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Par. Dist. Comp.*, 64(9):1017–1026, 2004.

[5] Q. Luo and J. Drake. A scalable parallel strassen's matrix multiplication algorithm for distributed-memory computers. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, SAC '95, pages 221–226, New York, NY, USA, 1995. ACM.

[6] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Euro-Par 2011, Part II*, volume 6853 of *Lecture Notes in Computer Science*, pages 90–109. Springer, 2011.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung.