中国科学院大学
夏季强化课程
2022

Fast Solvers for Large Algebraic Systems

Lecture 6. Communication hiding and avoiding

Chensong Zhang, AMSS
http://lsec.cc.ac.cn/~zhangcs

6
通信避免与隐藏

Table of Contents

Sources of Error in Simulation

Approximation: $$u(x) = U_h(x) + \mathcal{E}_{\text{dis}} + \mathcal{E}_{\text{alg}} + \mathcal{E}_{\text{fp}}$$

More refined mesh

Better discretizations

Better solvers

Better computers

Discretization Error	Algebraic Error	Floating-Point Error
Discretization Methods	Algebraic Solvers	Finite-Precision Arithmetic

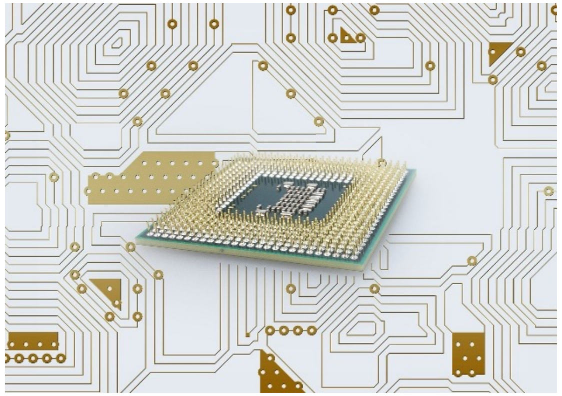
This Lecture

Introduction

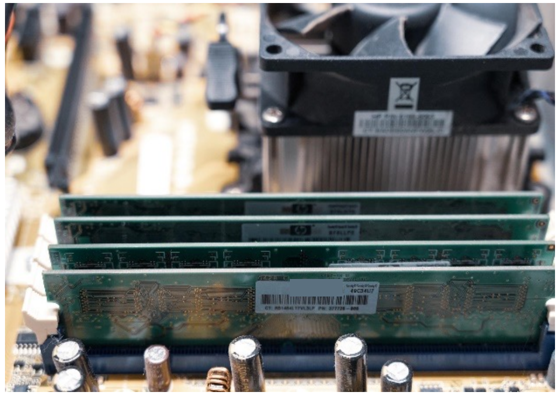
Why it is important to reduce data movement?

/01

Simplified Computer Architecture, Revisited



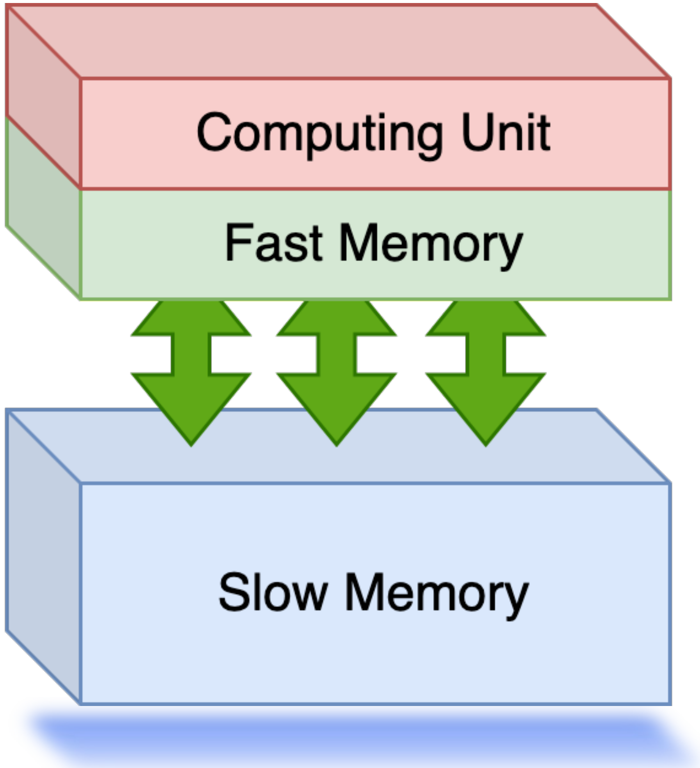
Hardware Architecture



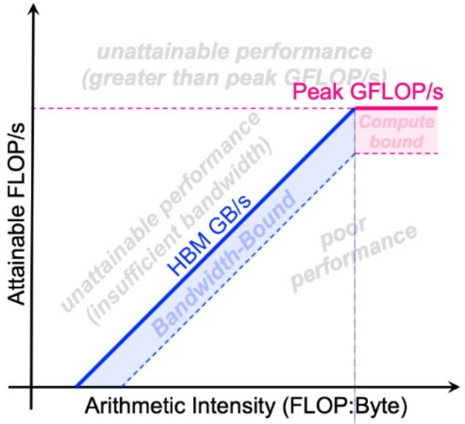
Ideal Model

Computation | GFLOP/s

Bandwidth | GB/s



- External Memory Model (computing is fast)
- Cache-Oblivious Model (not knowing cache size)



Roofline Model

Simplify the problem using two model parameters:

- Machine Balance
- $\text{AI} := \dfrac{\#\text{FP FLOP}}{\text{DataMove Byte}}$

Maximize Performance

- FP performance FLOPS (flops/sec) alone is misleading for modern computer's actual performance
- It's especially the case when we talk about large-scale simulation
- Review the roofline model; see Lecture 5 and Lecture by Prof. Wei Xue

$$\boxed{\begin{aligned}\text{Performance}(&\text{Num of Flops} \div \text{FLOPS},\\ &\text{Num of Bytes Moved} \div \text{Bandwidth},\\ &\text{Num of Messages} \times \text{Latency})\end{aligned}}$$

⟵

- Memory Wall
- Communication Wall

- Minimize volume of communication
- Minimize number of messages
- Complications: Multiple levels of memory and multiple types of parallelism
- Sometimes exchange memory/communication performance by redundant computation

Factor 1: Floating-Point Performance

- CISC (complex instruction set computing) architecture; Examples: Server and desktop CPUs
- RISC (reduced instruction set computing) architecture; Examples: Smartphones and tablets ARM CPUs. Less energy consumption!

How many floating-point calculations CPUs can do per sec at most? It becomes complicated.

FLOPS = Cores × Clock Speed × Num flops per cycle

= Cores × Clock Speed × Num SIMD Units × [(Num FMA units × 2) + Num Mul units]

Intel Core i9-9980HK (Coffee Lake)

- 2.3GHz 8-core, Turbo Boost up to 5.0GHz, AVX2 SIMD, FMA
- DP performance = 8 × 2.3 × 16 (AVX2 + FMA256) = 294GFLOPS (307)
- SP performance = 8 × 2.3 × 32 (AVX2 + FMA256) = 588GFLOPS

Factor 2: Communication Performance

Upon making reading requests, such as visiting a website, using an application, making a call, or downloading a file, users want to get quality responses as quickly as possible

Q: How to measure communication performance?

Bandwidth measures the amount of data that is able to pass (read and write) through a connection at a given time

Throughput refers to how much data can actually pass through, on average, over a specific period of time

- Data transmission performance ➔ Throughput is impacted by latency, so there may not be a linear relationship between bandwidth and throughput
- A network with high bandwidth may have components that process their various tasks slowly, while a lower-bandwidth network may have faster components, resulting in higher overall throughput

CAS and True Latencies

- Latency = Delay between when a “user” requires an action and when they get a “response”
- CAS (Column Access Strobe) latency is a measure of the clock cycles passing when the RAM module accesses a particular dataset in its column and making that data available after being instructed by a memory controller (source: Wiki)
- Example: RAM modules with a CAS latency of 17 will need (roughly) 17 clock cycles when a request is sent by the CPU and when the data is output by the RAM
- CAS latencies are an inaccurate indicator of memory performance: CAS latency (CL) and true latency

True Latency (ns) := (CAS Latency * Num of Data Trans / Clock Speed) *1000

Examples: DDR3-1600 CL11 vs DDR4-3200 CL22

➔➔➔ True = 11 *2 / 1600 * 1000 = 22 * 2 / 3200 * 1000 = 13.75ns

Sustainable Memory Bandwidth

STREAM: Memory Bandwidth Benchmark Test

STREAM Memory Bandwidth --- John D. McCalpin, mccalpin@cs.virginia.edu
Revised to Mon Apr 3 19:18:55 CDT 2017

All results are in MB/s --- 1 MB=10^6 B, *not* 2^20 B

Sub. Date	Machine ID	ncpus	COPY	SCALE	ADD	TRIAD	
2015.07.10	SGI_UV_3000	3072	12799304.0	12815808.0	13838195.0	13826185.0	data
2012.08.14	SGI_Altix_UV_2000	2048	6591669.0	6592082.0	7128484.0	7139690.0	data
2016.01.13	ScaleMP_Xeon_E5-2680v3_64B	1534	5741247.0	5775190.0	6318785.0	6367015.0	data
2011.04.05	SGI_Altix_UV_1000	2048	5321074.0	5346667.0	5823380.0	5859367.0	data
2006.07.10	SGI_Altix_4700	1024	3661963.0	3677482.0	4385585.0	4350166.0	data
2013.03.26	Fujitsu_SPARC_M10-4S	1024	3474998.0	3500800.0	3956102.0	4002703.0	data
2011.06.06	ScaleMP_Xeon_X6560_64B	768	1493963.0	2112630.0	2252598.0	2259709.0	data
2017.04.04	Fujitsu_SPARC_M12-2S	192	1322423.0	1299737.0	1479182.0	1530865.0	data
2004.12.22	SGI_Altix_3700_Bx2	512	906388.0	870211.0	1055179.0	1119913.0	data
2003.11.13	SGI_Altix_3000	512	854062.0	854338.0	1008594.0	1007828.0	data
2003.10.02	NEC_SX-7	32	876174.7	865144.1	869179.2	872259.1	data
2008.04.07	IBM_Power_595	64	679207.2	624707.8	777334.8	805804.6	data
2013.09.12	Oracle_SPARC_T5-8	128	604648.0	611264.0	622572.0	642884.0	data
1999.12.07	NEC_SX-5-16A	16	607492.0	590390.0	607412.0	583069.0	data
2009.08.10	ScaleMP_XeonX5570_vSMP_16B	128	437571.0	431726.0	442722.0	445869.0	data
1997.06.10	NEC_SX-4	32	434784.0	432886.0	437358.0	436954.0	data
2004.08.11	HP_AlphaServer_GS1280-1300	64	407351.0	400142.0	437010.0	431450.0	data
1996.11.21	Cray_T932_321024-3E	32	310721.0	302182.0	359841.0	359270.0	data
2014.04.24	Oracle_Sun_Server_X4-4	60	221370.0	221944.0	244588.0	245068.0	data
2007.04.17	Fujitsu/Sun_Enterprise_M9000	128	224401.0	223113.0	224271.0	227059.0	data

Source: https://www.cs.virginia.edu/stream/new.html

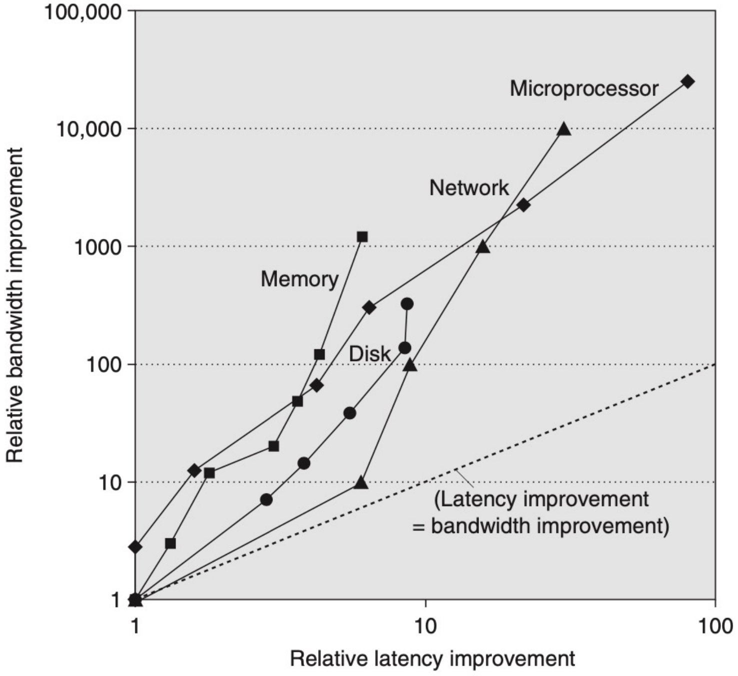


Figure 1.9 Log–log plot of bandwidth and latency milestones from Figure 1.10 relative to the first milestone. Note that latency improved 6X to 80X while bandwidth improved about 300X to 25,000X. Updated from Patterson [2004].

Source: Hennessy, John L., and David A. Patterson. "Computer architecture: a quantitative approach". Elsevier, 2011

Some Test Results from BSCC

2011年
成立
北京市和中科院共建重大项目
在“怀柔科学城”挂牌成立

2012年
SCP超级云计算平台上线
填补国内超算云服务领域空白

2014
“元”超级计算机上线
实现中国超算从百万亿次到
千万亿次的跨越

2018年
超级云计算服务平台发布
中国科技云·超算云发布
人工智能平台发布

2019年
启动超级云计算建设模式
通用X86计算资源4000服务器+
累计服务客户20000+

2020年
计算能力实现10PFlops
获2020 HPC TOP100榜单第三
通用超算算力第一

服务能力

北京超级云计算中心总核心数共27万核，服务用户（机构用户）数超过30000 家。在持续扩容计算资源的同时， 上线计算性能达百PFlops的国产服务器资源，满足大规模并行计算需求，可根据用户的计算量、应用程序及业务场景， 提供随需供应、不排队、省心省时的高品质VIP计算服务。

Reducing Communication Cost

Fundamental tools for analyzing/reducing communication cost

/02

Communication Performance Model

Single Communication Time = Latency + Num of Bytes Moved ÷ Bandwidth

Fast-Slow Memory Model

Not big enough to store the whole problem	Serial Case	Distributed Case
Fast Memory	Cache	Local
Communication ↕	Memory access ↕	Network access ↕
Sufficiently large but slow memory (assume it can't be used for FP arithmetic)	DRAM or HBM	Remote

Ref: CS267 lecture notes on J. Demmel's webpage: https://people.eecs.berkeley.edu/~demmel/

Basic Ideas on Reducing Communication

So communication could be more costly compared to computation. How can we get around?

Num of messages

Reduce total number of messages needed by better organizing algorithms, combining messages, ……

Size of messages

Reduce amount of data that need to be moved by better iterative algorithms, better partitioning, …….

Comm. hiding

Hide communication behind computation by aligning them in a smart way, ……

Better network

Use better inter-connecting network with high throughput, ……

Matrix Multiplication Algorithms

Algorithm 16: Naïve matrix multiplication

```
%% Given two matrices A, B ∈ ℝ^{n×n};
for i = 1 : n
    for j = 1 : n
        for k = 1 : n
            C(i,j) ← C(i,j) + A(i,k) * B(k,j);
        end
    end
end
```

- Naïve matrix-multiplication cost $O(n^3)$ operations. Strassen 1969 $O(n^{2.8074})$, Williams 2011 $O(n^{2.3728642})$, Allan & Williams 2020 $O(n^{2.3728596})$... Efforts to reduce this bound to $O(n^{2+\varepsilon})$
- Forward error of matrix-multiplications (depends on size of the inner loop)

$$\text{Compute } C = AB, A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k} \Longrightarrow |\hat{C} - C| \le \gamma_n |A||B| \qquad \gamma_n := \frac{nu}{1-nu}$$

Take Memory into Account

Algorithm 17: Naïve matrix multiplication with data movement

```
%% Given two matrices A, B ∈ R^{n×n};
for i = 1 : n
    %% Read A(i,:) into fast memory, n × n times in total
    for j = 1 : n
        %% Read B(:,j) into fast memory, n^2 × n times in total
        %% Read C(i,j) into fast memory, n^2 × 1 times in total
        for k = 1 : n
            C(i,j) ← C(i,j) + A(i,k) * B(k,j);
        end
        %% Write C(i,j) back to slow memory, n^2 × 1 times in total
    end
end
```

- Floating Point Arithmetic = $2n^3$, Data Movement = $n^3 + 3n^2$, AI or CI $\approx$ 2!
- Floating Point with FMA = n^3, Data Movement = $n^3 + 3n^2$, AI or CI $\approx$ 1!

Memory bound!

Blocked (Tiled) GEMM Algorithm

Divide-and-Conquer

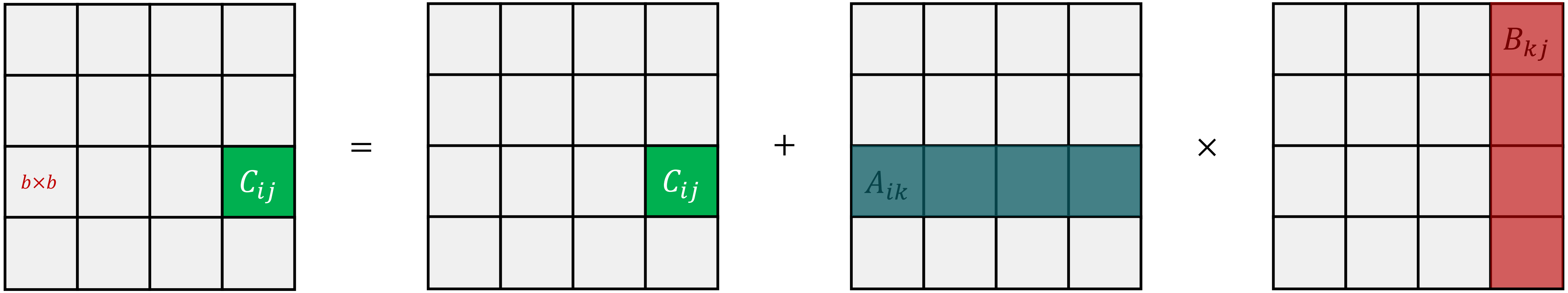
Algorithm 18: Blocked matrix multiplication

```
%% Given two matrices A, B ∈ R^{n×n};
for i = 1 : n/b
    for j = 1 : n/b
        %% Read C(i,j) into fast memory, (n/b)^2 × b^2 = n^2 times in total
        for k = 1 : n/b
            %% Read A(i,k) into fast memory, (n/b)^3 × b^2 = n^3/b times in total
            %% Read B(k,j) into fast memory, (n/b)^3 × b^2 = n^3/b times in total
            C(i,j) ← C(i,j) + A(i,k) * B(k,j);
        end
        %% Write C(i,j) back to slow memory, (n/b)^2 × b^2 = n^2 times in total
    end
end
```

- Floating Point Arithmetic = $2n^3$, Data Movement = $2\frac{n^3}{b} + 2n^2$, AI or CI $\approx b$!

CPU bound!

Arithmetic Intensity of Blocked GEMM



- So we want to make block size as large as possible! But there is a constraint …
- Assume that $3\ b\times b$ blocks can fit into the fast memory (of size M) ➔ Hence $b \le \sqrt{M/3}$
- Total data movement $\approx 2\frac{n^3}{b} = \Omega(n^3/\sqrt{M})$
- Q: Can we make this cache-oblivious?
- Q: Can this be further improved? Make less data movement?

Lower Bounds on Communication

- Serial MatMul problem [Hong and Kung 1981]

  Number of Words Moved $\geq \Omega\left(\text{Num of operations}/\sqrt{\text{Size of fast memory}}\right) = \Omega(n^3/\sqrt{M})$

- Attainable by using blocked implementation, like in BLAS
- Parallel MatMul (load balanced version) problem [Irony, Toledo, and Tiskin 2004]

  Number of Words Moved $\geq \Omega\left(\text{Num of operations per proc}/\sqrt{\text{Size of fast memory per proc}}\right)$

  $= \Omega(\frac{n^3/P}{\sqrt{M/P}})$ … $= \Omega(n^2/\sqrt{P})$ if n^2/P for each processor

- How about the SUMMA algorithm? Almost there! For each $k = 1{:}n/b$, we get

  Number of messages $= 2log(\sqrt{P})\sqrt{P},$ Number of entries moved $= 2log(\sqrt{P})n^2/\sqrt{P}$

- Attainable by using the Cannon's algorithm, 2.5D SUMMA, Parallel Strassen, …

Strassen's Algorithm

- Naïve matrix-multiplication (8 mul + 4 add)

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{bmatrix}$$

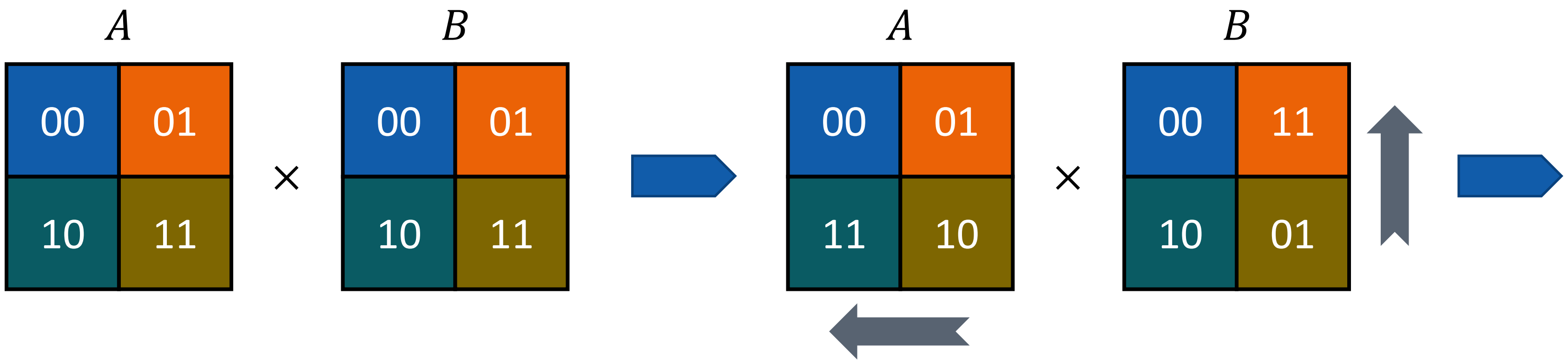
- Strassen's matrix-multiplication (7 mul + 18 add)

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

$$\begin{aligned} M_1 &= (A_{11} + A_{22}) * (B_{11} + B_{22}) & M_5 &= (A_{11} + A_{12}) * B_{22} \\ M_2 &= (A_{21} + A_{22}) * B_{11} & M_6 &= (A_{21} - A_{11}) * (B_{11} + B_{12}) \\ M_3 &= A_{11} * (B_{12} - B_{22}) & M_7 &= (A_{12} - A_{22}) * (B_{21} + B_{22}) \\ M_4 &= A_{22} * (B_{21} - B_{11}) \end{aligned}$$

- Strassen's algorithm has an asymptotic complexity $O(n^{log_2 7})$!

Cannon's Algorithm for MatMul



$\sqrt{P} \times \sqrt{P}$ processor layout

Initial alignment

Sub-block MatMul

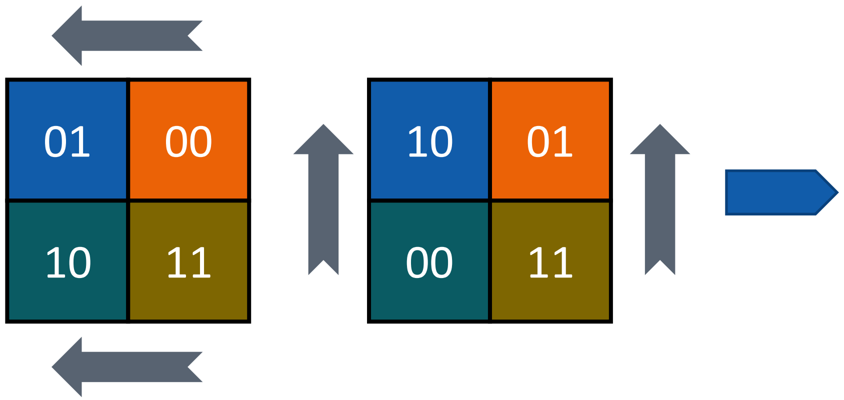
$A_{00}B_{00}$ $A_{01}B_{11}$

$A_{11}B_{10}$ $A_{10}B_{01}$

Number of entries moved:

$$\sqrt{P} \times 2\left(\frac{n}{\sqrt{P}}\right)^2 = \frac{2n^2}{\sqrt{P}}$$

Lower bound on communication!



After first shift

$+A_{01}B_{10}$ $+A_{00}B_{01}$

$+A_{10}B_{00}$ $+A_{11}B_{11}$

Lynn Cannon 1969

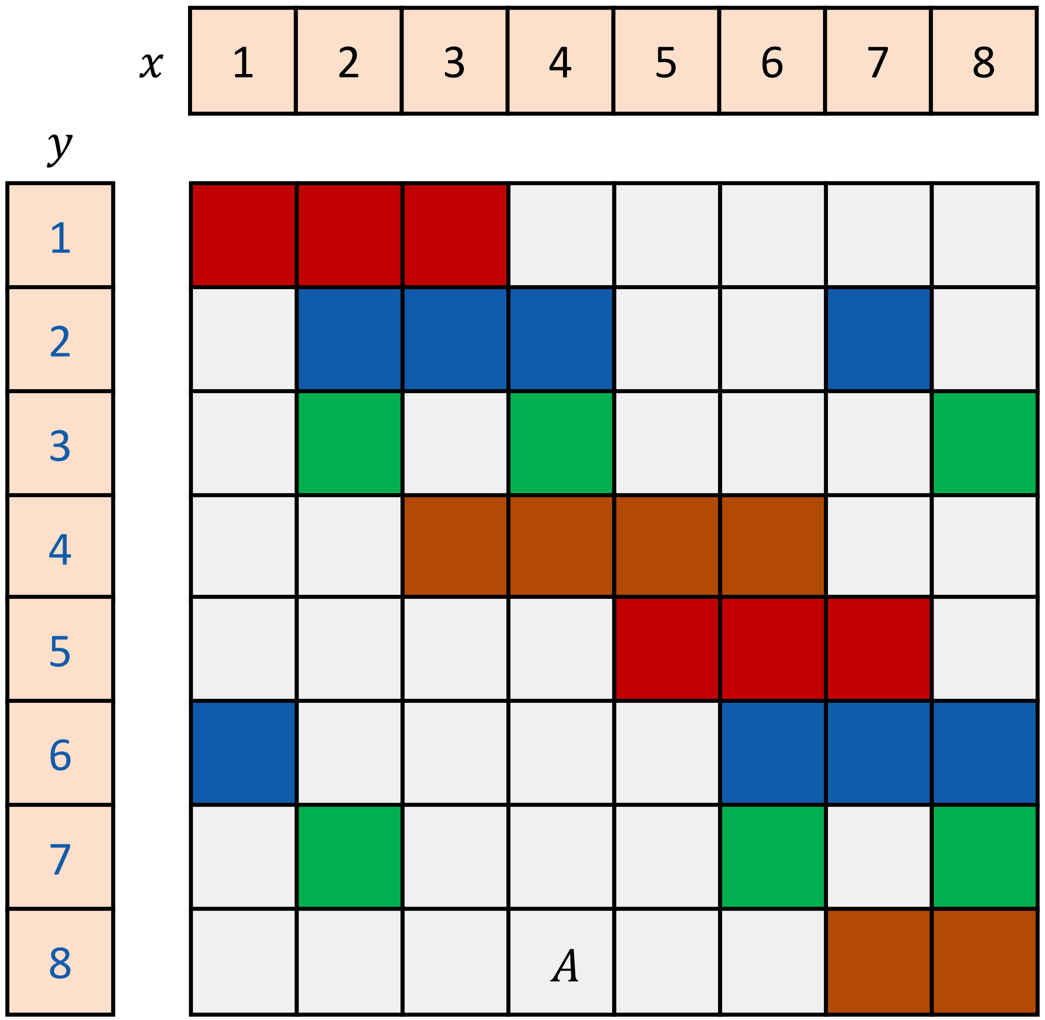
Optimizing Communication

- In the fast matrix-multiplication example, we not only reduced communication …
- We can actually minimized data movement in some cases!
- Q: Can this be done in general?

- Algorithms involving three embedded loops can be optimized in a similar way
- BLAS3 (GEMM, triangular solve)
- Cholesky, LDL^T , LU, QR (Regular LU with partial pivoting does not attain the bound)
- Eigenvalue and SVD
- Graph algorithms (shortest paths between all pairs)
- The lower bound can also be applied to sparse matrices

Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz, Minimizing Communication in Numerical Linear Algebra, SIAM Journal on Matrix Analysis and Applications 2011 32:3, 866-901

A More Challenging Case: SpMV



$y = y + Ax$

Fast Memory

A	1,1	1,2	1,3	2,2
x	1	2	3	4
y	1	2	3	4

Read X 3

A	2,3	2,4	2,7	3,2
x	1	2	3	4
y	1	2	3	4

Read X 1

A	2,3	2,4	2,7	3,2
x	5	6	7	8
y	1	2	3	4

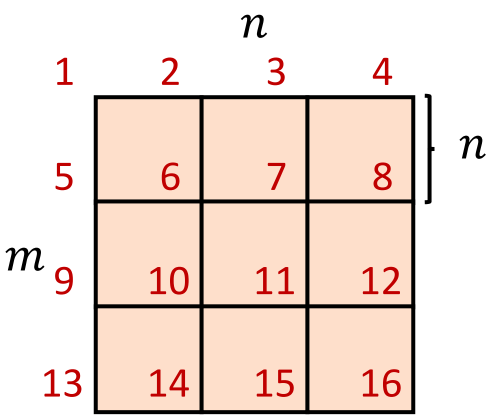
Read X 1

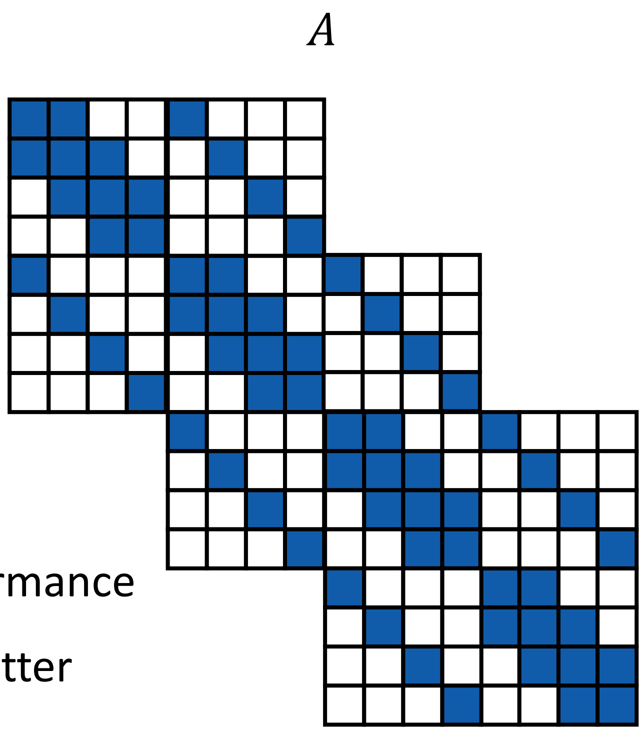
A	2,3	2,4	2,7	3,2
x	1	2	3	4
y	1	2	3	4

Read X 1

Reordering Sparse Matrices

- Sparse matrices are a lot more difficult to deal with and to analyze
- Use different data structures and optimized lib (Review Lecture by Weifeng Liu)
- Use different ordering to improve efficiency (depends on what solver you will use)

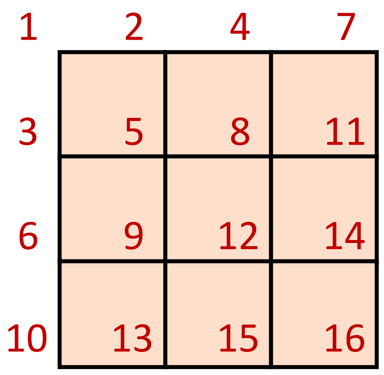


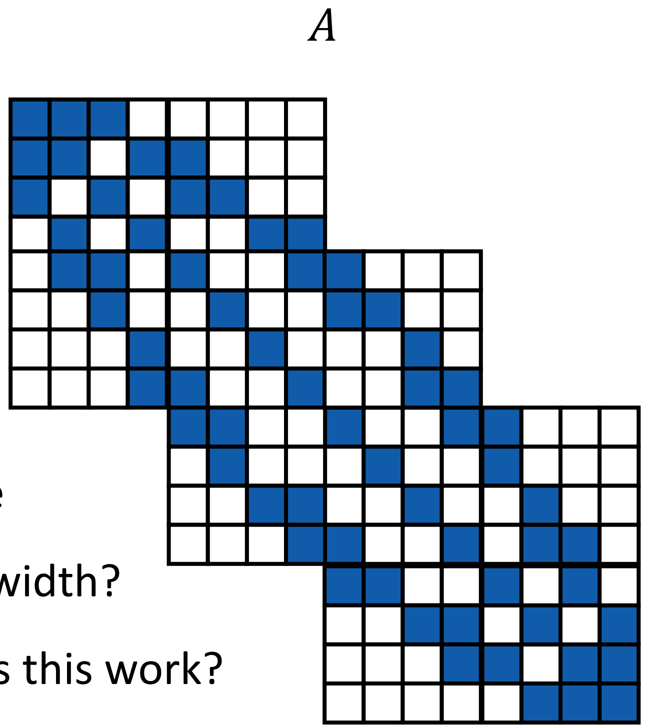


Bandwidth = n

Locality? ➔ Cache performance

Stencil computation is better





BFS ➔ Cuthill-McKee

➔ Reduce bandwidth?

➔ Q: When does this work?

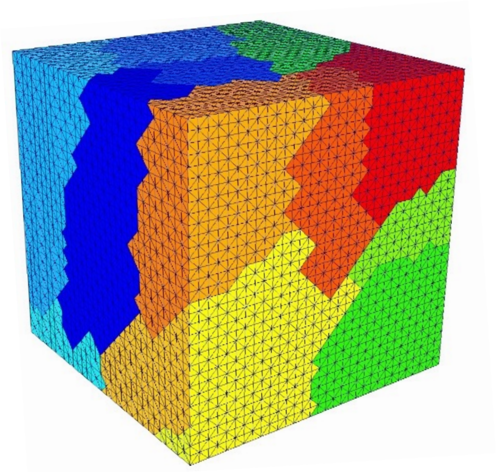
Considerations on Iterative Solvers

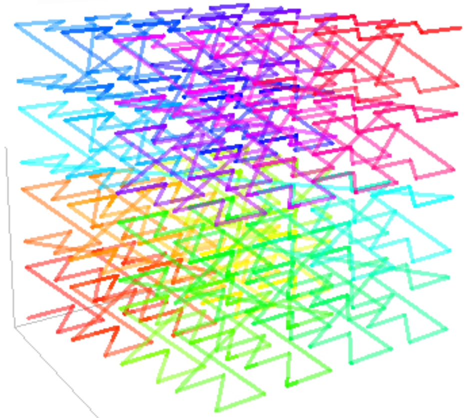
Reducing communication cost of iterative solvers

/03

Parallel Iterative Solvers, Revisited

初始网格处理 → 空间填充曲线 → 负载平衡剖分 → 子区域求解





网格划分

1	3	5	7
2	4	6	8

数据分发

进程1

1	3	5
2	4	6

局部网格

Ghost区
需要进行
数据传递

进程2

3	5	7
4	6	8

局部计算

局部网格

多目标优化问题
每个部分的工作量的变化极小，
界面面积极小，
裂缝和井等特征不跨区域……

GMRES Method, Revisited

- The generalized minimum residual (GMRES) method finds:

$$\min_{e \in \mathcal{K}_m(A,r)} \|r - Ae\|_0$$

in the Krylov subspace

$$\mathcal{K}_m(A,r) := \mathrm{span}\{r, Ar, A^2r, \ldots, A^{m-1}r\}$$

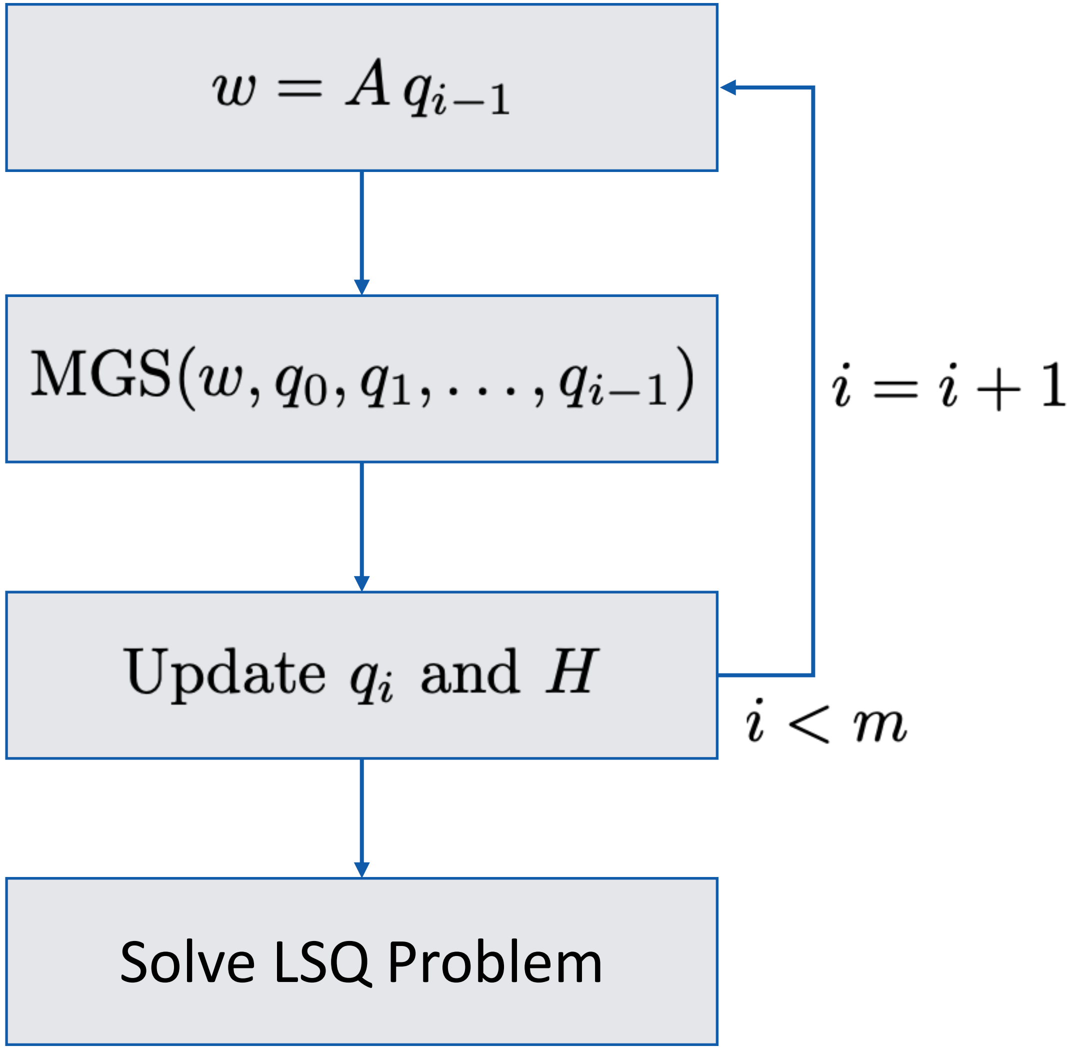
- We form an orthonormal basis of the Krylov subspace

$$\mathcal{K}_m := \mathrm{span}\{q_1, q_2, \ldots, q_m\}$$

- By applying the modified Gram-Schmidt (MGS) algorithm, we form H and then solve the least squares (LSQ) problem with H

Q: Remember why we use this implementation? We tried to: (1) avoid numerical instability; (2) use iterative procedure to stop at any iteration. But communication was not concerned!

Data Movement in GMRES



- Analyzing data movement is difficult
  - Parallel architectures
  - Parallel data layout
  - Parallel algorithm

- SpMV   No chance for data reuse
  - Words moved $\sim O(m \cdot nnz)$
  - Number of messages $\sim O(m)$

- MGS   Iterative
  - Words moved $\sim O(m^2 \cdot n)$
  - Number of messages $\sim O(m^2 \cdot \log P)$

Communication-Avoiding GMRES

$$W = [Aq_0, A^2q_0, \ldots, A^mq_0]$$

$$[Q, R] = \mathrm{TSQR}(W)$$

Build H

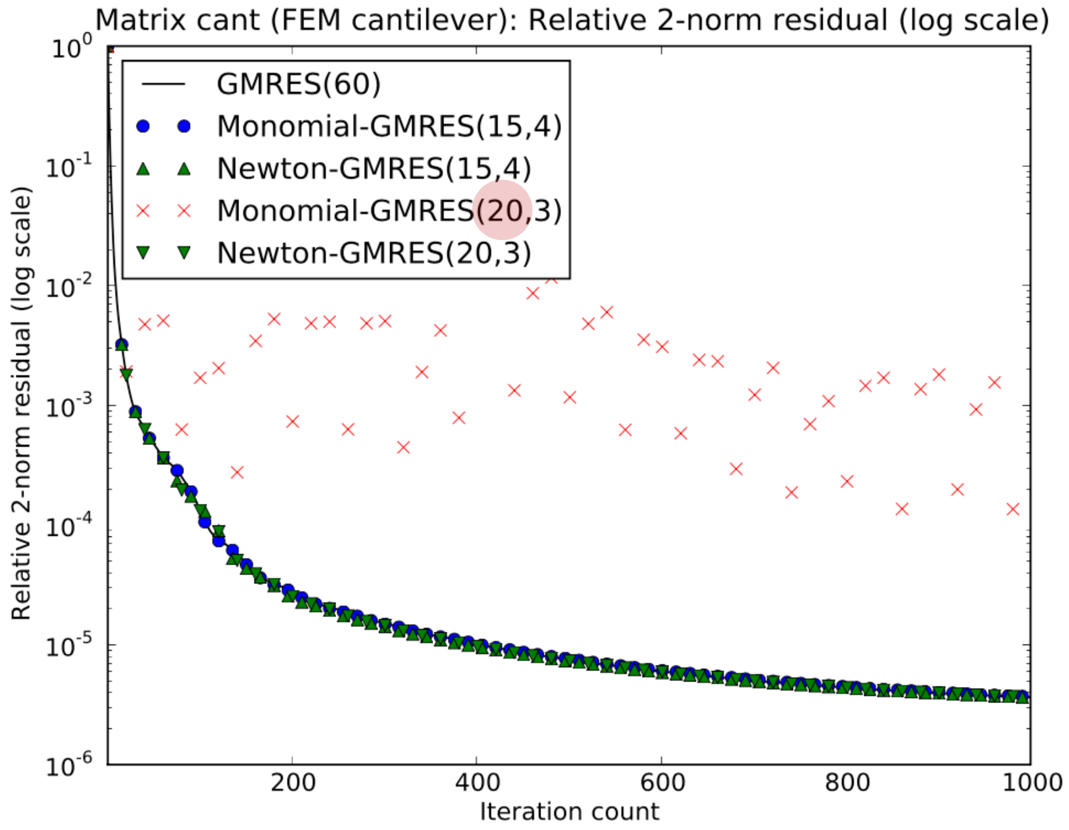
Solve LSQ Problem

- Reorganize the algorithm
  - Identical mathematical method (with exact FP)
  - Use the matrix powers kernel
  - Use QR factorization instead of MGS

- Matrix Powers
  - Words moved $\sim O(nnz)$
  - Number of messages $\sim O(1)$

- TSQR
  - Words moved $\sim O(m \cdot n)$
  - Number of messages $\sim O(\log P)$

Performance of CA-GMRES



- The "easy" implementation of CA-GMRES is not always stable because the matrix powers kernel may produce linearly dependent vectors
- Use the Newton basis (shifted polynomials based on the eigenvalues of the upper Hessenberg matrix) proposed by Bai, Hu, and Reichel, 1994

$$W = \left[(A - \lambda_1 I)q_0, \ldots, \Pi_{j=1}^{m}(A - \lambda_j I)q_0\right]$$

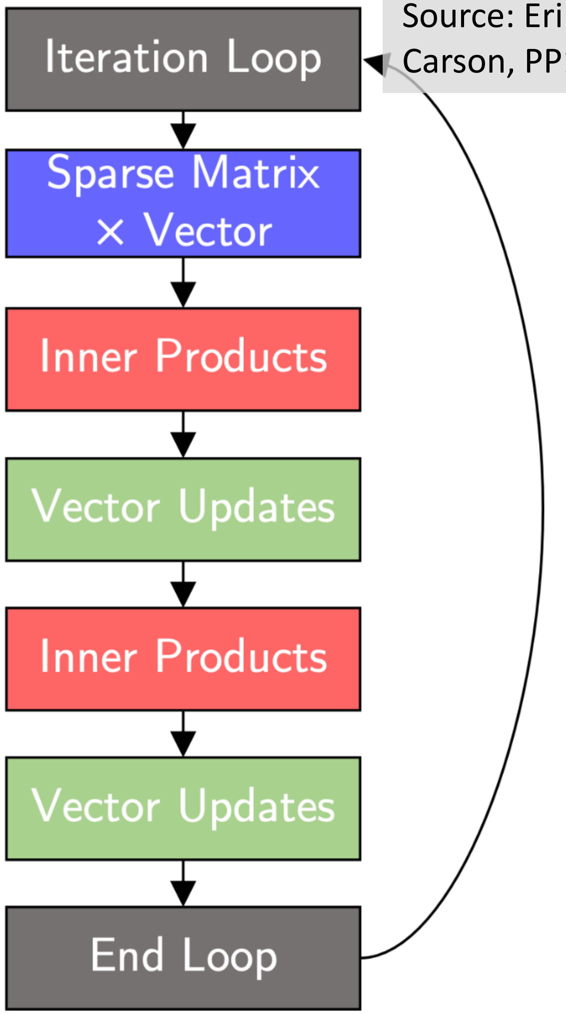
Source: Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. 2009. Minimizing communication in sparse matrix solvers. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09).

Conjugate Gradient Method, Revisited

SpMV is the most expensive part: more floating-point calculations required and not cache-friendly (memory-bound)

Algorithm 2: Conjugate gradient method

%% Given an initial guess u and a tolerance ε;
$r \leftarrow f - \mathcal{A}u,\ p \leftarrow r$;
while $\|r\| > \varepsilon$
  $\alpha \leftarrow (r,r)/(\mathcal{A}p,p)$;
  $\tilde{u} \leftarrow u + \alpha p$;
  $\tilde{r} \leftarrow r - \alpha \mathcal{A}p$;
  $\beta \leftarrow (\tilde{r},\tilde{r})/(r,r)$;
  $\tilde{p} \leftarrow \tilde{r} + \beta p$;
  Update: $u \leftarrow \tilde{u},\ r \leftarrow \tilde{r},\ p \leftarrow \tilde{p}$;
end



Source: Erin Carson, PP18

Inner products are expensive for communication: not much computation cost but global all-reduce is necessary (communication-bound)

Three-Term Recurrence CG

M. Hoemmen 2010

- Based on the three-term recurrence formulation for residuals

$$r_{k+1} = \rho_k(r_k - \gamma_k A r_k) + (1 - \rho_k) r_{k-1}$$

and the residuals are orthogonal to each other, we have

$$\gamma_k = \frac{(r_k, r_k)}{(A r_k, r_k)}$$

$$\rho_k = \left(1 - \frac{\gamma_k}{\gamma_{k-1}} \frac{(r_k, r_k)}{(r_{k-1}, r_{k-1})} \frac{1}{\rho_{k-1}}\right)^{-1}$$

- We can derive a new recurrence relation

$$x_{k+1} = \rho_k(x_k + \gamma_k r_k) + (1 - \rho_k) x_{k-1}$$

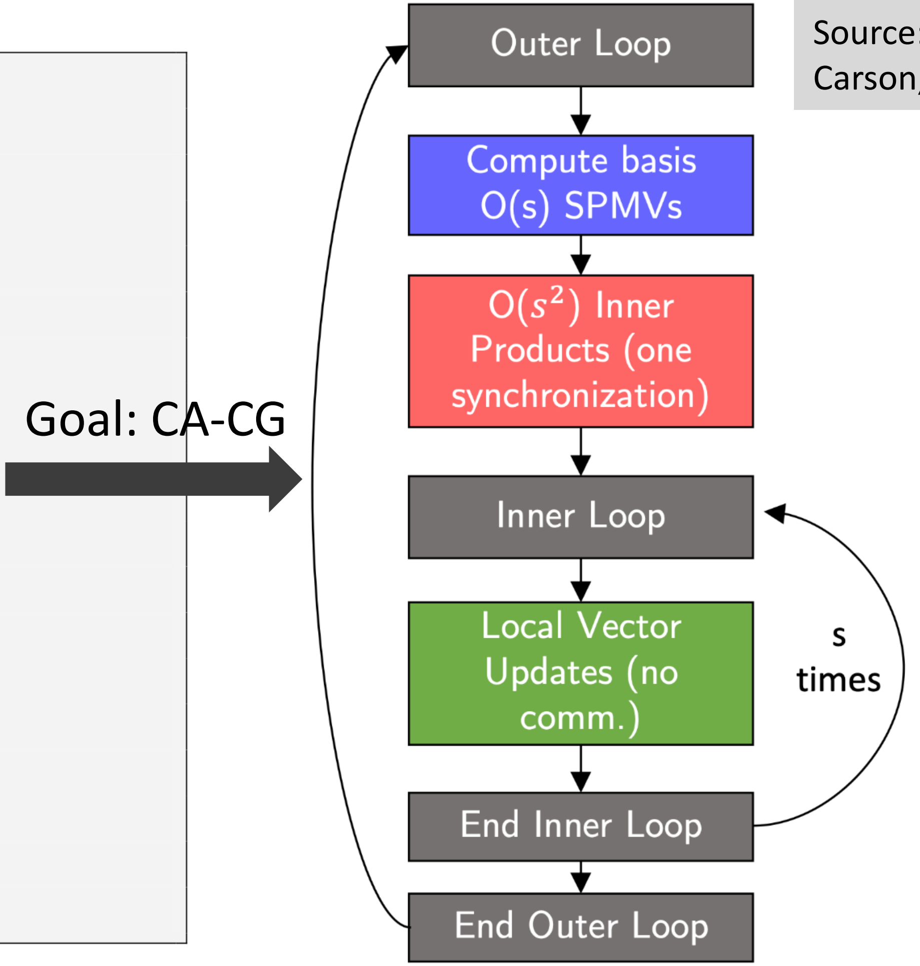
This formulation (usually named as CG3) does not involve the conjugate directions. This is desirable for deriving s-step CG algorithms.

Ref: Y. Saad, “Iterative Methods for Sparse Linear Systems”, SIAM, Philadelphia, Second Ed., 2003

s-Step CG3 Method

Algorithm 19: s-Step conjugate gradient method

%% Given an initial guess u_1 and a tolerance ε;
$r_1 \leftarrow f - \mathcal{A}u_1$;
for k = 0:MaxIter
    for j = 1:s
        $w_{sk+j} \leftarrow \mathcal{A}r_{sk+j}$; %% P2P communication
        $\mu_{sk+j} \leftarrow (r_{sk+j}, r_{sk+j})$; %% Test for convergence; All_reduce
        $\nu_{sk+j} \leftarrow (w_{sk+j}, r_{sk+j})$;
        $\gamma_{sk+j} \leftarrow \mu_{sk+j}/\nu_{sk+j}$;
        if $sk + j == 1$
            $\rho_{sk+j} \leftarrow 1$;
        else
            $\xi_1 \leftarrow \gamma_{sk+j}/\gamma_{sk+j-1}$;
            $\xi_2 \leftarrow \mu_{sk+j}/\mu_{sk+j-1}$;
            $\rho_{sk+j} \leftarrow (1 - \xi_1\xi_2/\rho_{sk+j-1})^{-1}$;
        end
        $x_{sk+j+1} \leftarrow \rho_{sk+j}(x_{sk+j} + \gamma_{sk+j}r_{sk+j}) + (1 - \rho_{sk+j})x_{sk+j-1}$;
        $r_{sk+j+1} \leftarrow \rho_{sk+j}(r_{sk+j} - \gamma_{sk+j}w_{sk+j}) + (1 - \rho_{sk+j})r_{sk+j-1}$;
    end
end



Source: Erin Carson, PP18

Ref: Mark F. Hoemmen, Communication-avoiding Krylov subspace methods, Ph.D. thesis, 2010

Communication-Avoiding CG

M. Hoemmen 2010

- We have the recurrence relation for residual

$$r_{sk+j+1} = \rho_{sk+j}\left(r_{sk+j} - \gamma_{sk+j} A r_{sk+j}\right) + (1-\rho_{sk+j}) r_{sk+j-1}$$

- Rearrange the terms as follows:

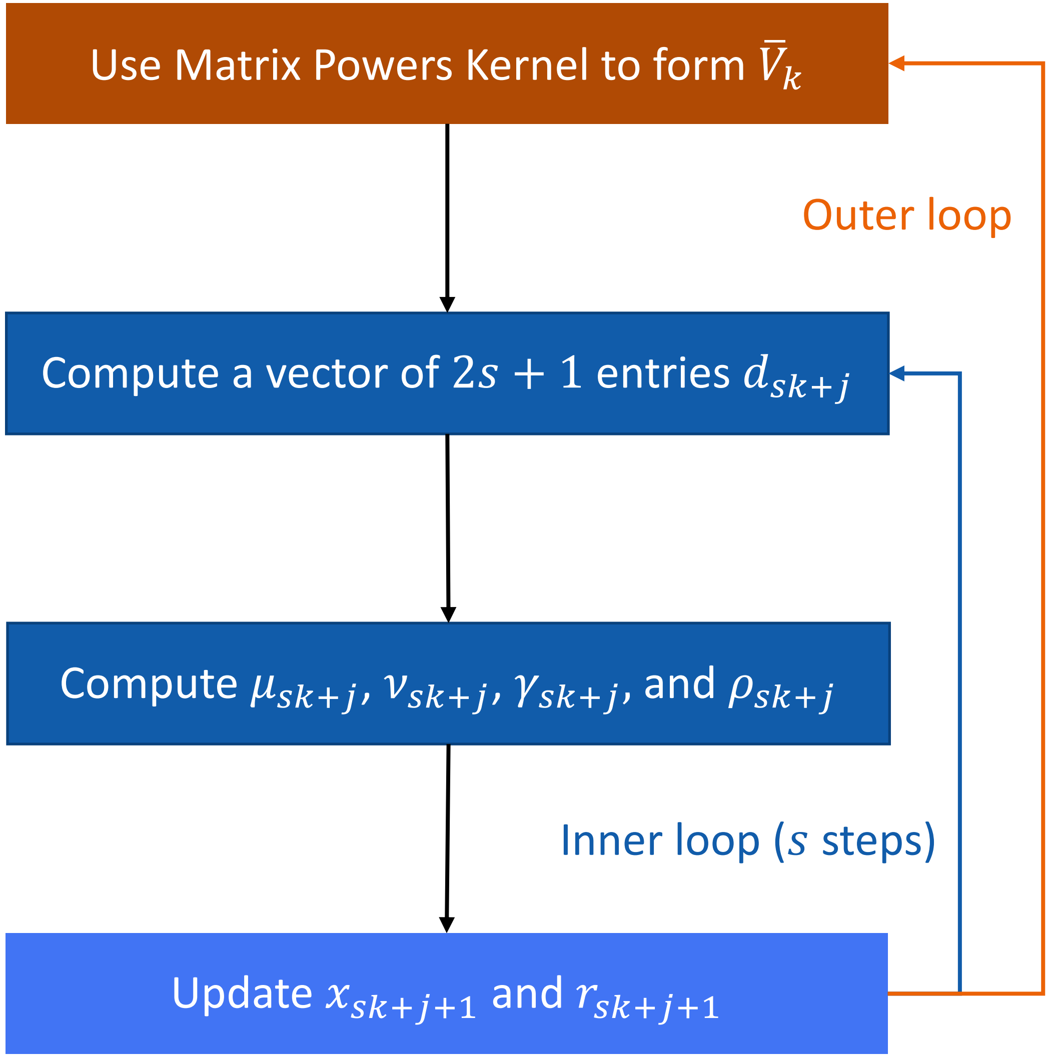
$$A r_{sk+j} = \frac{1-\rho_{sk+j}}{\rho_{sk+j}\gamma_{sk+j}} r_{sk+j-1} + \frac{1}{\rho_{sk+j}\gamma_{sk+j}} r_{sk+j} - \frac{1}{\rho_{sk+j}\gamma_{sk+j}} r_{sk+j+1}$$

- Write the recurrence in terms of matrix form:

$$\boxed{A\underbrace{\left[r_{sk+1},\ldots,r_{sk+s}\right]}_{R_k} = \frac{1-\rho_{sk}}{\rho_{sk}\gamma_{sk}} r_{sk} e_1^T + \underbrace{\left[r_{sk+1},\ldots,r_{sk+s+1}\right]}_{\bar{R}_k}\bar{T}_k}$$

where $\bar{T}_k$ is a $(s+1)\times s$ tridiagonal matrix in terms of $\rho_{sk+1},\ldots,\rho_{sk+s},\gamma_{sk+1},\ldots,\gamma_{sk+s}$

From s-Step CG To CA-CG



$$\bar{V}_k := [v_{sk+1}, \ldots, v_{sk+s}]$$

$$\{v_{sk+i}\}_{i=1:s+1} = \mathrm{span}\{r_{sk+1}, Ar_{sk+1}, \ldots, A^s r_{sk+1}\}$$

is a basis of the Krylov subspace

$$\boxed{w_{sk+j} := A\, r_{sk+1} = [R_{k-1}, \bar{V}_k]\, d_{sk+j}}$$

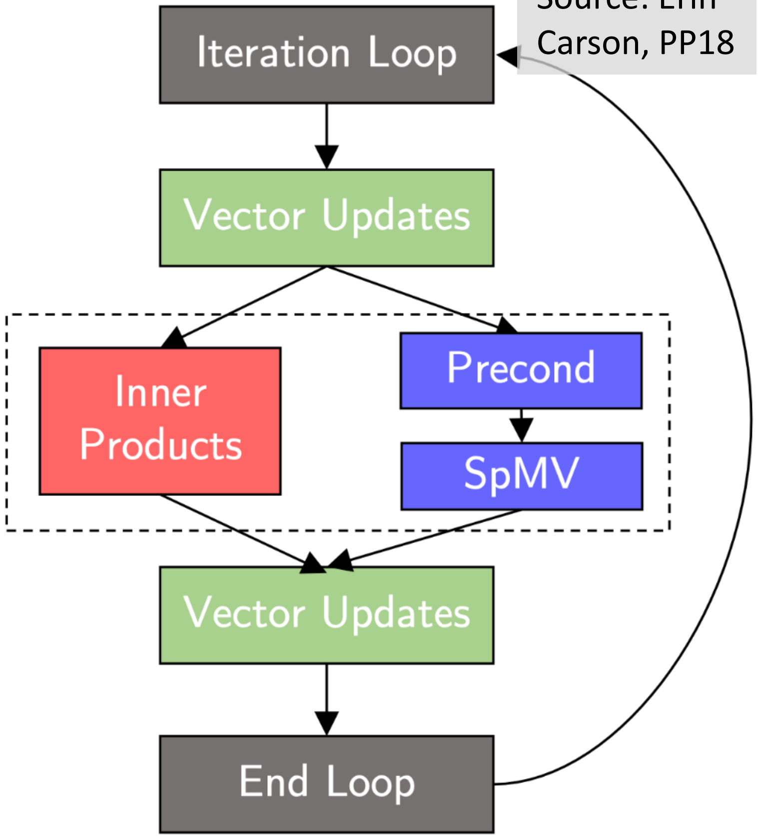
- The matrix powers kernel needs to load the coefficient matrix once
- In exact arithmetic, the algorithm produces the same results as the standard CG
- Further improvement by using a inner product coalescing kernel Sec 5.4.4, M. Hoemmen 2010

Pipelined Conjugate Gradient Method

Algorithm 20: Pipelined conjugate gradient method

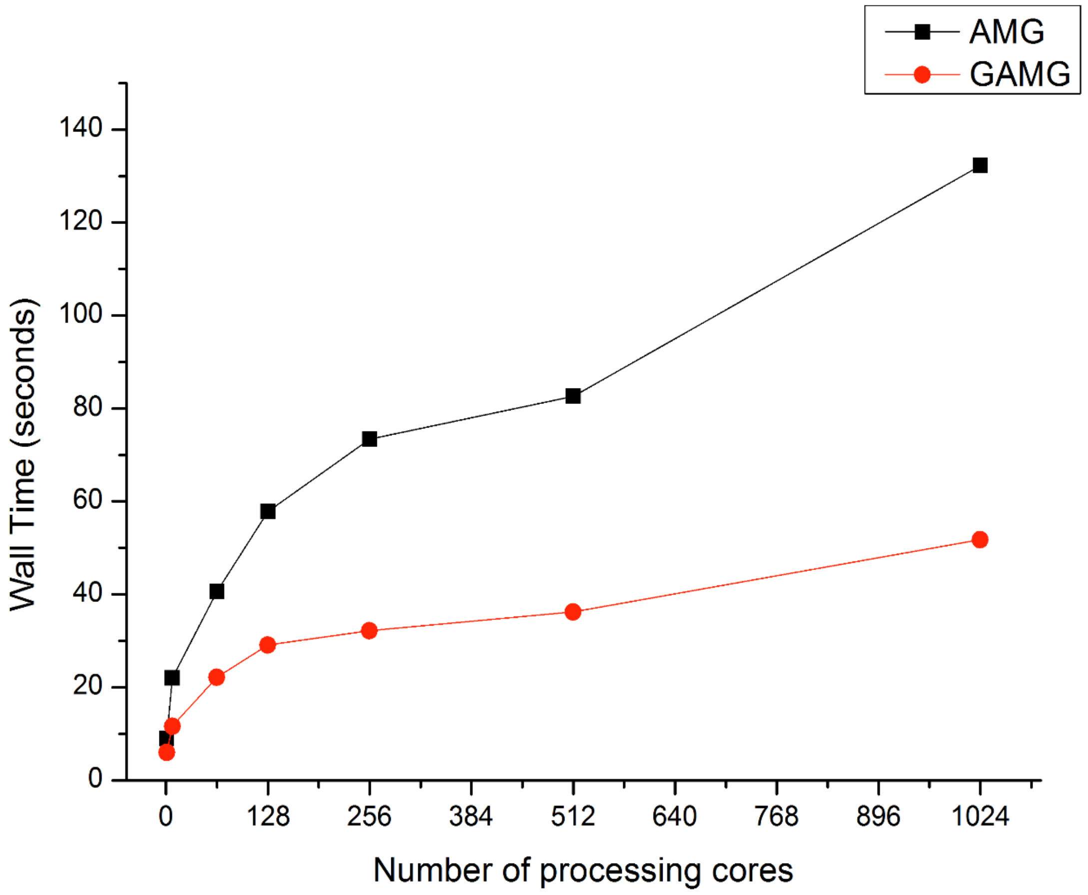
%% Given an initial guess u and a tolerance ε;
$r \leftarrow f - \mathcal{A}u, \ p \leftarrow r$;
$s \leftarrow \mathcal{A}p, \ w \leftarrow \mathcal{A}r, \ z \leftarrow \mathcal{A}w$;
$\alpha \leftarrow (r, r)/(p, s)$;
while $\|r\| > \varepsilon$
    $\tilde{u} \leftarrow u + \alpha p$;
    $\tilde{r} \leftarrow r - \alpha s$;
    $\tilde{w} \leftarrow w - \alpha z$;
    $\tilde{z} \leftarrow \mathcal{A}\tilde{w}$; %% SpMV, asymc
    $\beta \leftarrow (\tilde{r}, \tilde{r})/(r, r)$;
    $\alpha \leftarrow \frac{(\tilde{r},\tilde{r})}{(w,r)-(\beta/\alpha)(\tilde{r},\tilde{r})}$;
    $\tilde{p} \leftarrow \tilde{r} + \beta p$;
    $\tilde{s} \leftarrow \tilde{w} + \beta s$;
    $\tilde{z} \leftarrow \tilde{z} + \beta z$; %% Results of SpMV needed here!
    Update: $u \leftarrow \tilde{u}, \ r \leftarrow \tilde{r}, \ p \leftarrow \tilde{p}, \ w \leftarrow \tilde{w}, \ s \leftarrow \tilde{s}, \ z \leftarrow \tilde{z}$;
end

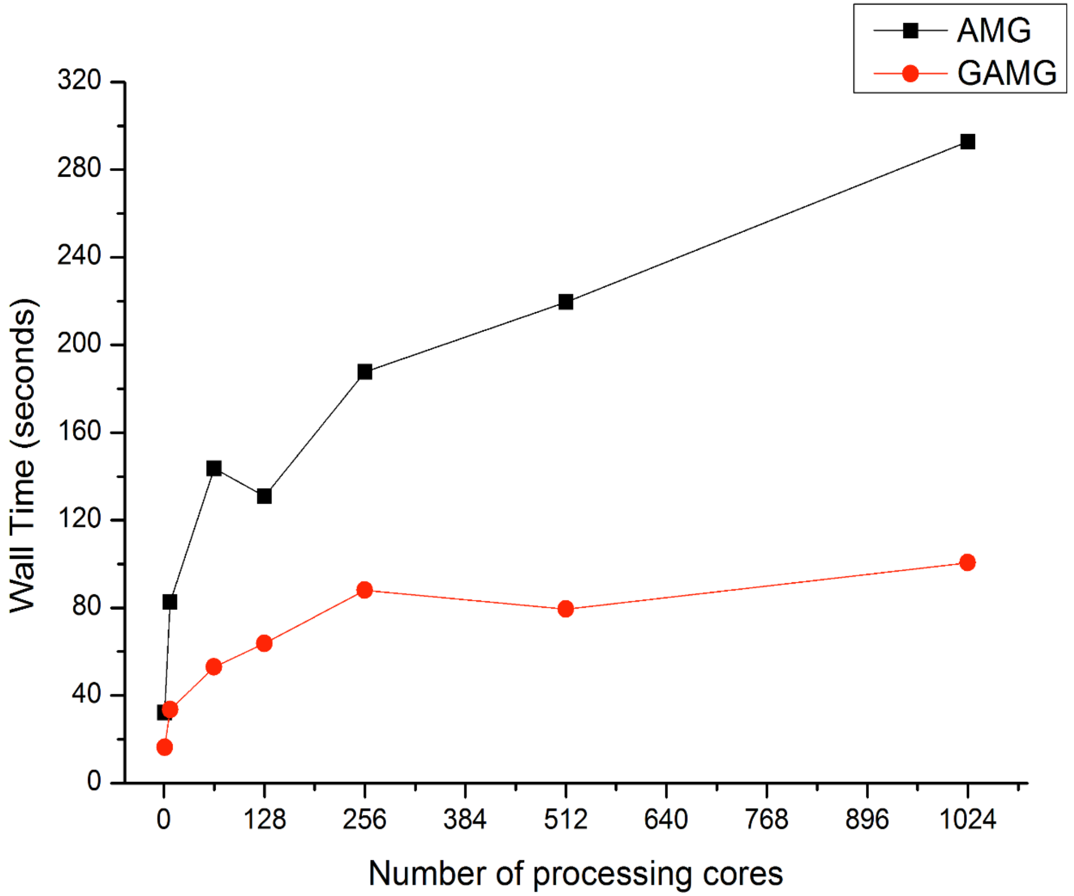
Source: Erin Carson, PP18



Ref: Ghysels, Pieter and Wim Vanroose. "Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm." Parallel Comput. 40 (2014): 224-238.

Taking Preconditioning Into Account





Source: A stable and scalable hybrid solver for rate-type non-Newtonian fluid models, Y.-J. Lee, W. Leng, and C.-S. Zhang, Journal of Computational and Applied Mathematics, 300, 103–118 (07/2016).

Multigrid Methods, Revisited

Algorithm (Setup step for multigrid methods)

For a given sparse matrix $A \in \mathbb{R}^{N \times N}$, we apply the following steps:

1. Obtain a suitable matrix for coarsening $A_f \in \mathbb{R}^{N_f \times N_f}$ (for example, $A_f = A_{\text{sym}}$);
2. Define a coarse space with N_c variables (C/F splitting or aggregation);
3. Construct a prolongation (usually an interpolation) $P \in \mathbb{R}^{N_f \times N_c}$:
   - 3.1. Give a sparsity pattern for the interpolation P;
   - 3.2. Determine weights of the interpolation P;
4. Construct a restriction $R \in \mathbb{R}^{N_c \times N_f}$ (for example, $R = P^T$);
5. Form a coarse-level coefficient matrix (for example, $A_c = RA_f P$);
6. Give a sparse approximation of A_c whenever necessary.

Optimizing Parallel Multigrid

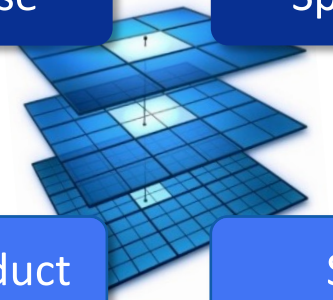
Permutation

Transpose

SpGEMM

RAP

P’AP



Inner Product

SpMV

Prolongation

Aggregation

Restriction

Injection

Smoother

Coarsest solver

- Scalable
- Low complexity
- Fast convergence

Better algorithms

Autotuning data structures

Predicting number of nnz

Software prefetching

Fusing operations

Avoid branching

Comm. avoiding / hiding

Mixed precisions

Reading and Thinking

NCMIS

SIAM J. MATRIX ANAL. & APPL.
Vol. 32, No. 3, pp. 866–901

MINIMIZING COMMUNICATION IN NUMERICAL LINEAR ALGEBRA*

GREY BALLARD[†], JAMES DEMMEL[‡], OLGA HOLTZ[§], AND ODED SCHWARTZ[¶]

Abstract. In 1981 Hong and Kung proved a lower bound on the amount of communication (amount of data moved between a small, fast memory and large, slow memory) needed to perform dense, n-by-n matrix multiplication using the conventional $O(n^3)$ algorithm, where the input matrices were too large to fit in the small, fast memory. In 2004 Irony, Toledo, and Tiskin gave a new proof of this result and extended it to the parallel case (where communication means the amount of data moved between processors). In both cases the lower bound may be expressed as $\Omega(\#\text{arithmetic\_operations}/\sqrt{M})$, where M is the size of the fast memory (or local memory in the parallel case). Here we generalize these results to a much wider variety of algorithms, including LU factorization, Cholesky factorization, LDL^T factorization, QR factorization, the Gram–Schmidt algorithm, and algorithms for eigenvalues and singular values, i.e., essentially all direct methods of linear algebra. The proof works for dense or sparse matrices and for sequential or parallel algorithms. In addition to lower bounds on the amount of data moved (bandwidth cost), we get lower bounds on the number of messages required to move it (latency cost). We extend our lower bound technique to compositions of linear algebra operations (like computing powers of a matrix) to decide whether it is enough to call a sequence of simpler optimal algorithms (like matrix multiplication) to minimize communication, or whether we can do better. We give examples of both. We also show how to extend our lower bounds to certain graph-theoretic problems. We point out recently designed algorithms that attain many of these lower bounds.

Key words. linear algebra algorithms, bandwidth, latency, communication-avoiding, lower bound

AMS subject classifications. 68Q25, 68W10, 68W15, 68W40, 65Y05, 65Y10, 65Y20, 65F30

DOI. 10.1137/090769156

- Do you do extremely large simulation now? In the future?
- How do you analyze the performance of your parallel code?
- Is the code communication-bound?
- Have you applied any communication avoiding or hiding algorithms?
- Do you think CA and CH algorithms will help? Why?

Contact Me

- Office hours: Mon 14:00—15:00
- Walk-in or online with appointment
- zhangcs@lsec.cc.ac.cn
- http://lsec.cc.ac.cn/~zhangcs

My sincere gratitude to:

Wei Xue, Bin Dai

中国科学院大学
夏季强化课程
2022

Fast Solvers for Large Algebraic Systems

THANKS

Chensong Zhang, AMSS

http://lsec.cc.ac.cn/~zhangcs

Release version 2022.07.02