High-Performance Variants of Krylov Subspace Methods: I/II

Erin C. Carson

Katedra numerické matematiky, Matematicko-fyzikální fakulta, Univerzita Karlova

SNA '19

January 21-25, 2019

This research was partially supported by OP RDE project No. CZ.02.2.69/0.0/0.0/16_027/0008495

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

Lecture Outline

- Parallel computers and performance modeling
  - Architecture trends
- Krylov subspace methods
  - Properties
  - Performance bottlenecks at scale
- High-performance variants of Krylov subspace methods
  - Early approaches
  - Pipelined methods
  - s-step methods
- Practical implementation issues and challenges

Computational and Data Science at Scale

- Why are we interested in solving larger and larger problems?
- Enables new frontiers in computational science and engineering

  $\Rightarrow$ Finer-grained simulation, over longer time scales, processing huge amounts of available data

Computational and Data Science at Scale

- Why are we interested in solving larger and larger problems?
- Enables new frontiers in computational science and engineering

  ⇒ Finer-grained simulation, over longer time scales, processing huge amounts of available data

  - Atmosphere, Earth, Environment
  - Physics - applied, nuclear, particle, fusion, photonics
  - Bioscience, Biotechnology, Genetics
  - Chemistry, Molecular Sciences
  - Geology, Seismology
  - Electrical Engineering, Circuit Design, Microelectronics
  - Mechanical Engineering - from prosthetics to spacecraft

Computational and Data Science at Scale

- Why are we interested in solving larger and larger problems?
- Enables new frontiers in computational science and engineering

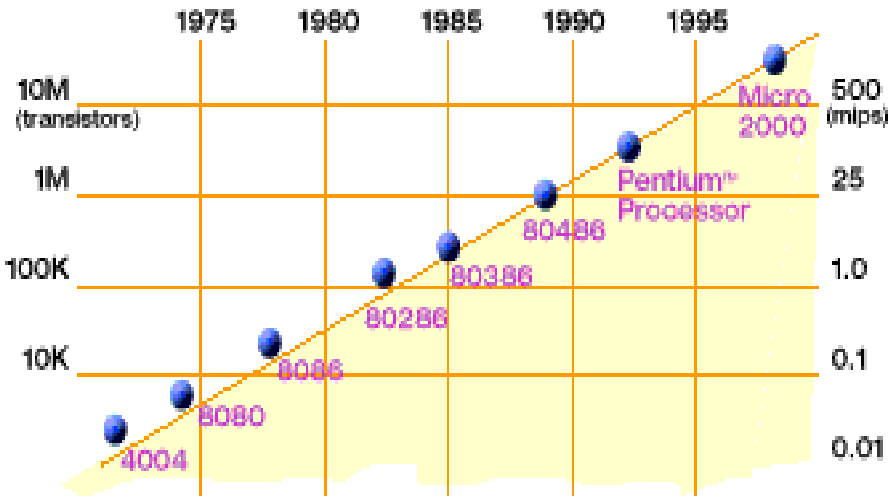
  ⇒ Finer-grained simulation, over longer time scales, processing huge amounts of available data

  - Atmosphere, Earth, Environment
  - Physics - applied, nuclear, particle, fusion, photonics
  - Bioscience, Biotechnology, Genetics
  - Chemistry, Molecular Sciences
  - Geology, Seismology
  - Electrical Engineering, Circuit Design, Microelectronics
  - Mechanical Engineering - from prosthetics to spacecraft

- Also industrial and commercial interests

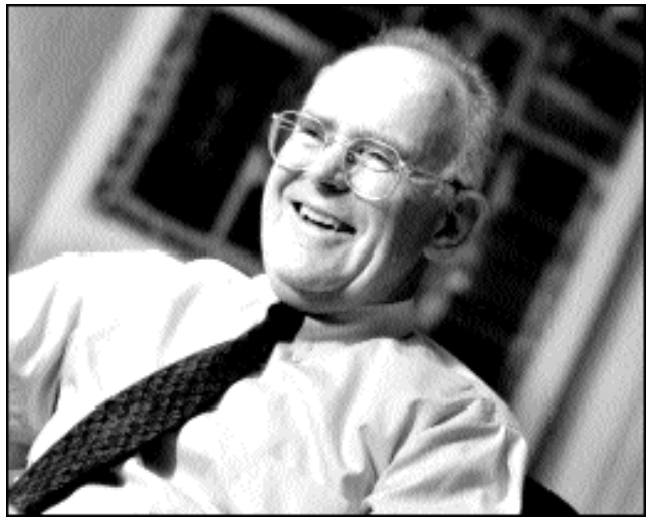
  - "Big Data", databases, data mining
  - Artificial Intelligence (AI)
  - Medical imaging and diagnosis
  - Pharmaceutical design
  - Financial and economic modeling
  - Advanced graphics and virtual reality
  - Oil exploration

Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years
"Moore's Law"

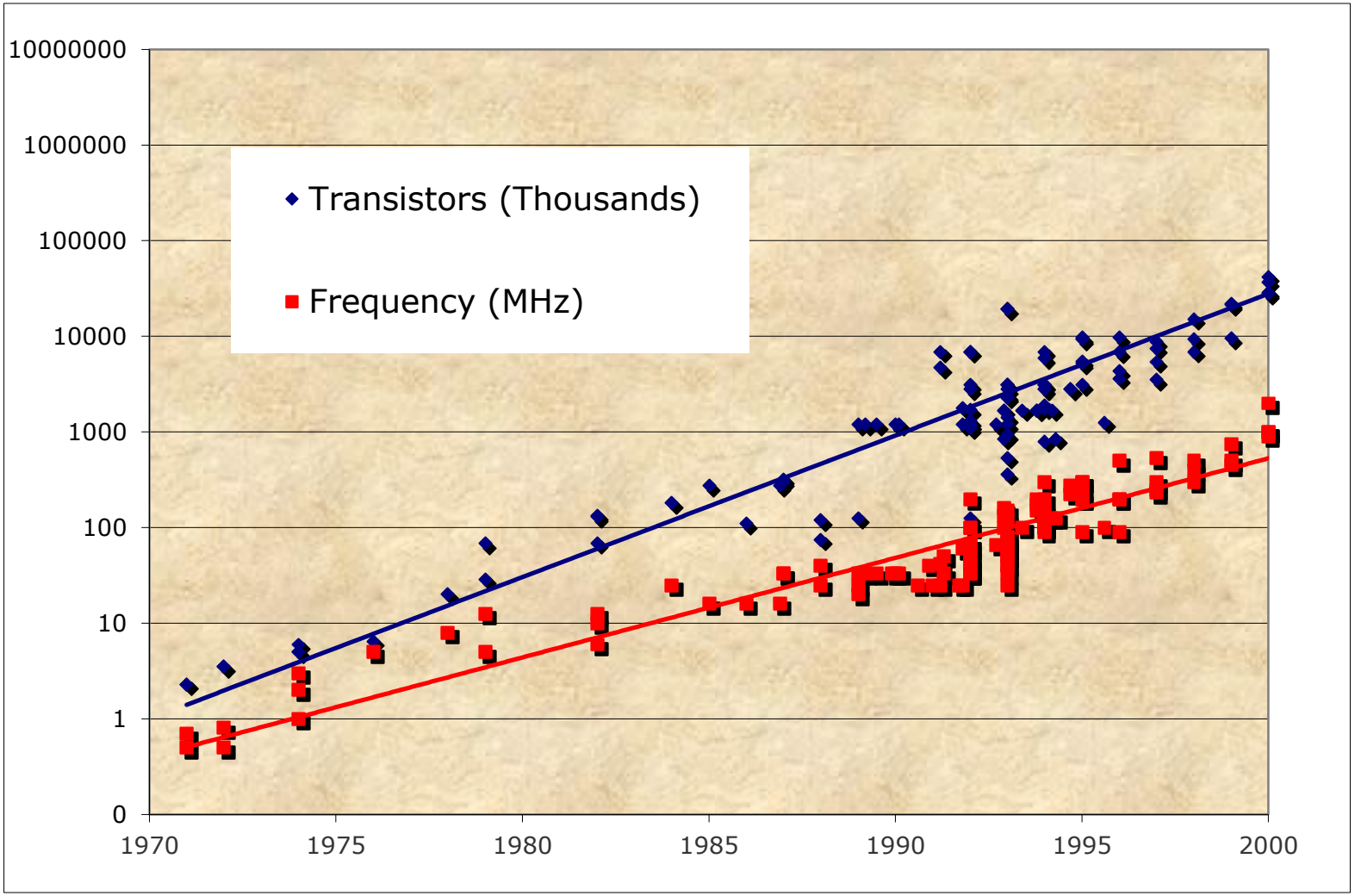
Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

Microprocessor Transistors / Clock (1970-2000)



Slide source: Kathy Yelick

Historical Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x?

Slide source: Kathy Yelick

Historical Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x?
- Clock rate goes up by x because wires are shorter
  - actually less than x, because of power consumption

Slide source: Kathy Yelick

Historical Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x?
- Clock rate goes up by x because wires are shorter
  - actually less than x, because of power consumption
- Transistors per unit area goes up by x^2

Slide source: Kathy Yelick

Historical Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x?
- Clock rate goes up by x because wires are shorter
  - actually less than x, because of power consumption
- Transistors per unit area goes up by x^2
- Die size has also increased
  - typically another factor of $\sim x$

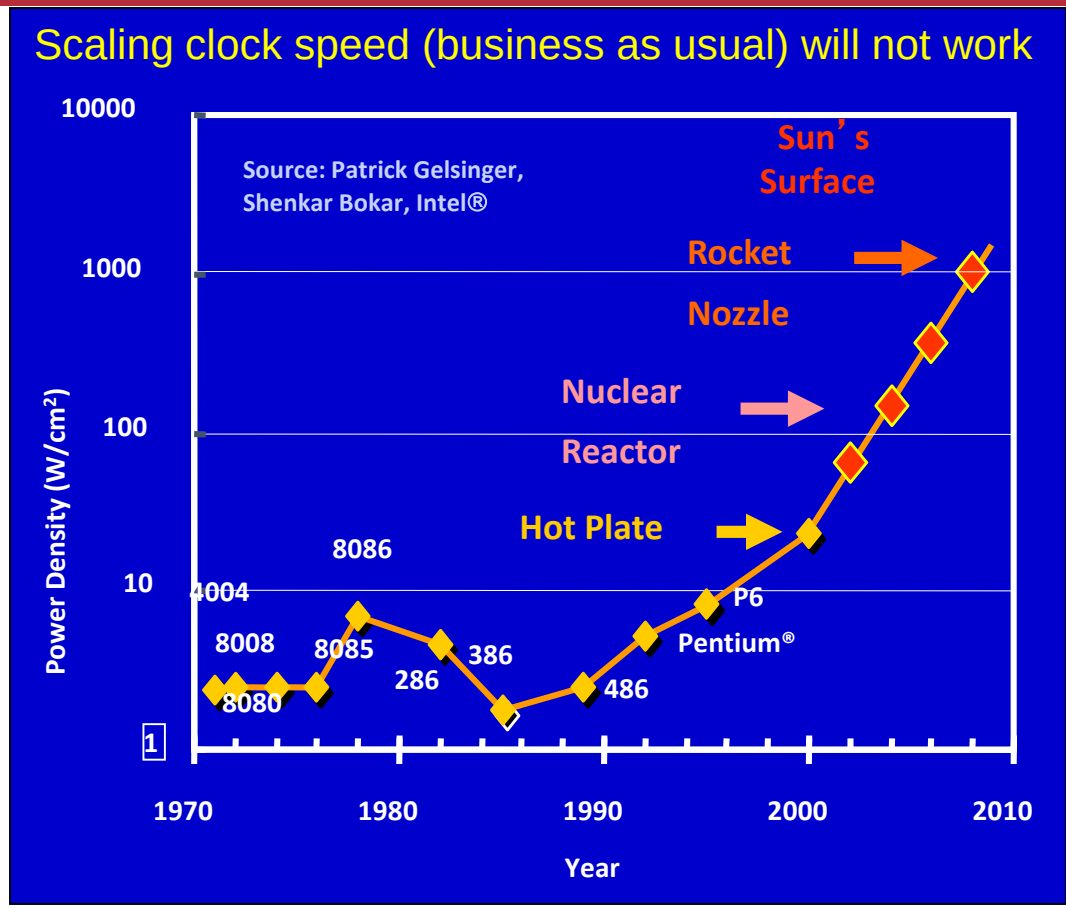
Slide source: Kathy Yelick

Historical Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x?
- Clock rate goes up by x because wires are shorter
  - actually less than x, because of power consumption
- Transistors per unit area goes up by x^2
- Die size has also increased
  - typically another factor of $\sim x$
- Raw computing power of the chip goes up by $\sim x^4$!
  - typically x^3 is devoted to either on-chip
    - parallelism: hidden parallelism such as ILP
    - locality: caches
- So most programs x^3 times faster, without changing them

Slide source: Kathy Yelick

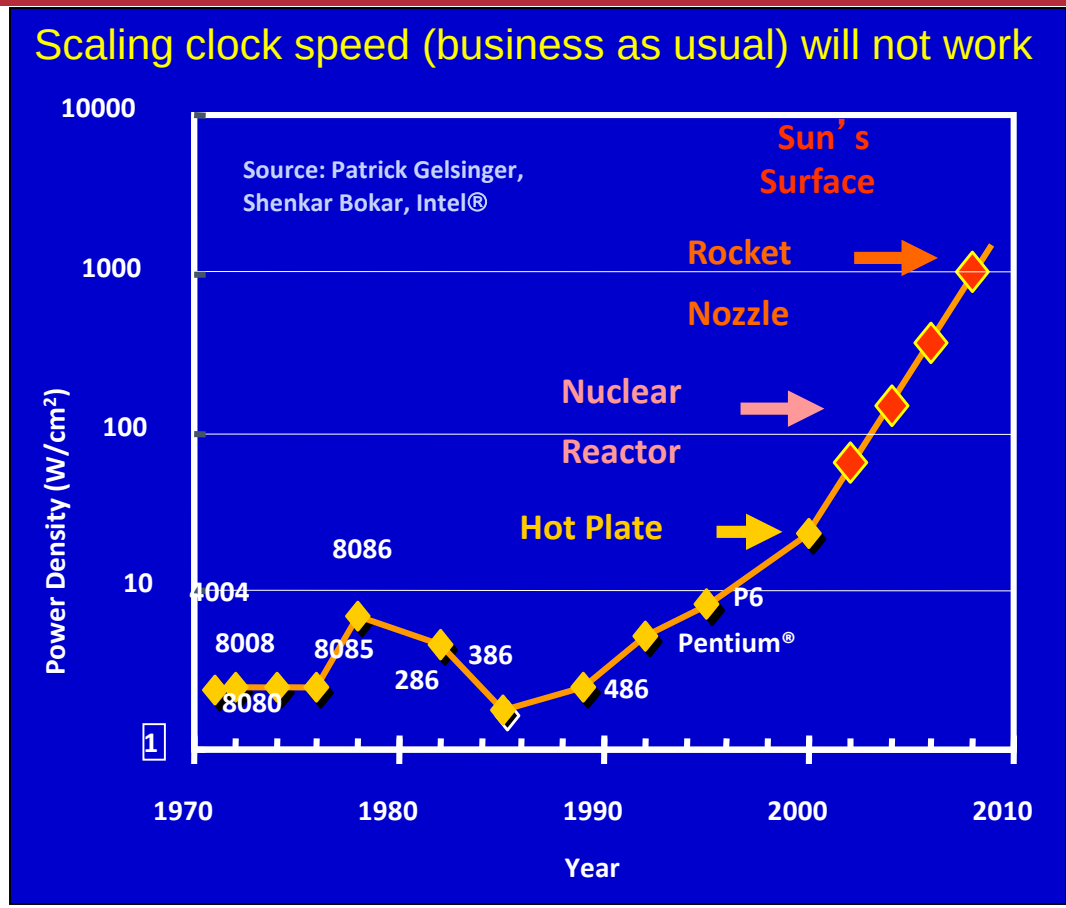
Power Density Limits Serial Performance



Slide source: Kathy Yelick

Power Density Limits Serial Performance

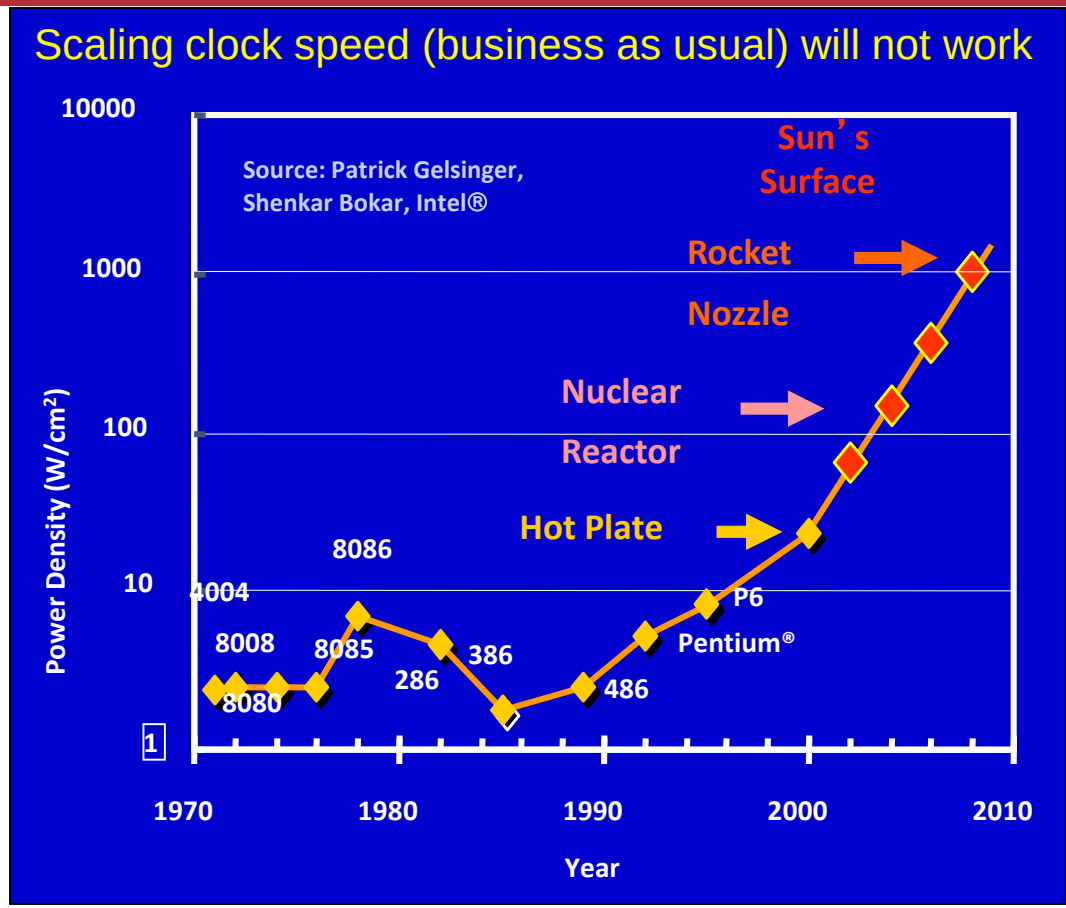
- Concurrent systems are more power efficient
  - Dynamic power is proportional to $V^2 f C$
  - Increasing frequency (f) also increases supply voltage (V) → cubic effect
  - Increasing cores increases capacitance (C) but only linearly
  - Save power by lowering clock speed



Slide source: Kathy Yelick

Power Density Limits Serial Performance

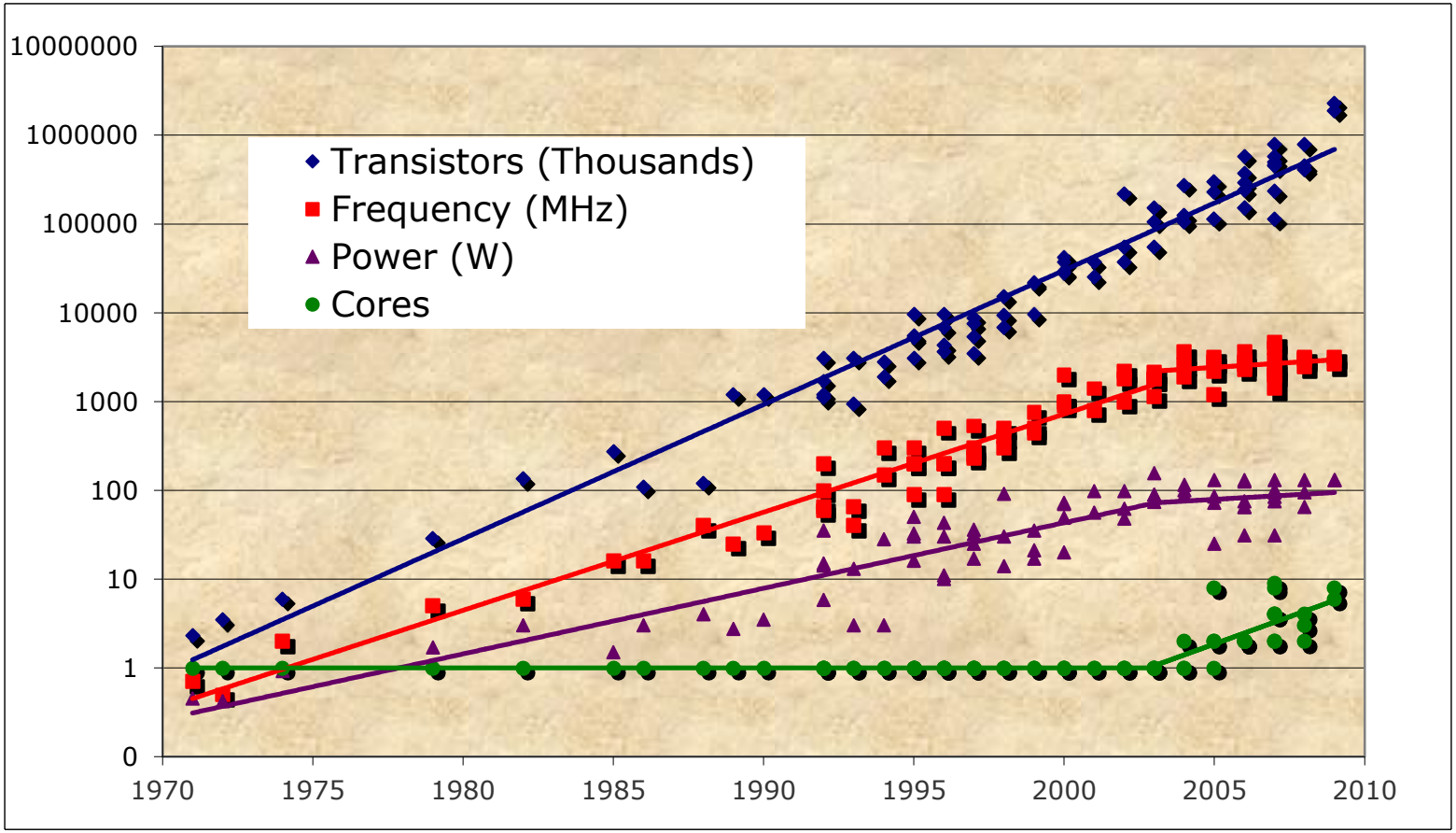
- Concurrent systems are more power efficient
  - Dynamic power is proportional to $V^2 f C$
  - Increasing frequency (f) also increases supply voltage (V) → cubic effect
  - Increasing cores increases capacitance (C) but only linearly
  - Save power by lowering clock speed



- High performance serial processors waste power
  - Speculation, dynamic dependence checking, etc. burn power
  - Implicit parallelism discovery
- More transistors, but not faster serial processors

Slide source: Kathy Yelick

Revolution in Processors



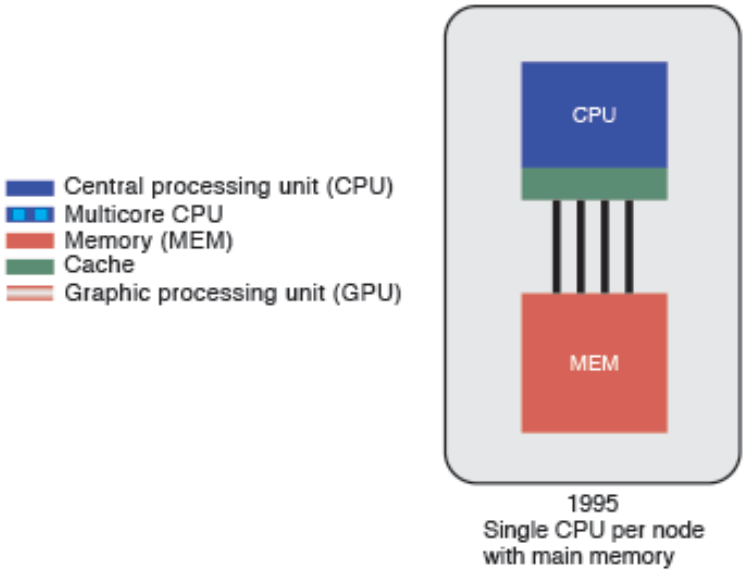
- Chip density is continuing increase ~2x every 2 years
- Clock speed is not
- Number of processor cores may double instead
- Power is under control, no longer growing

Parallel Computer Architectures

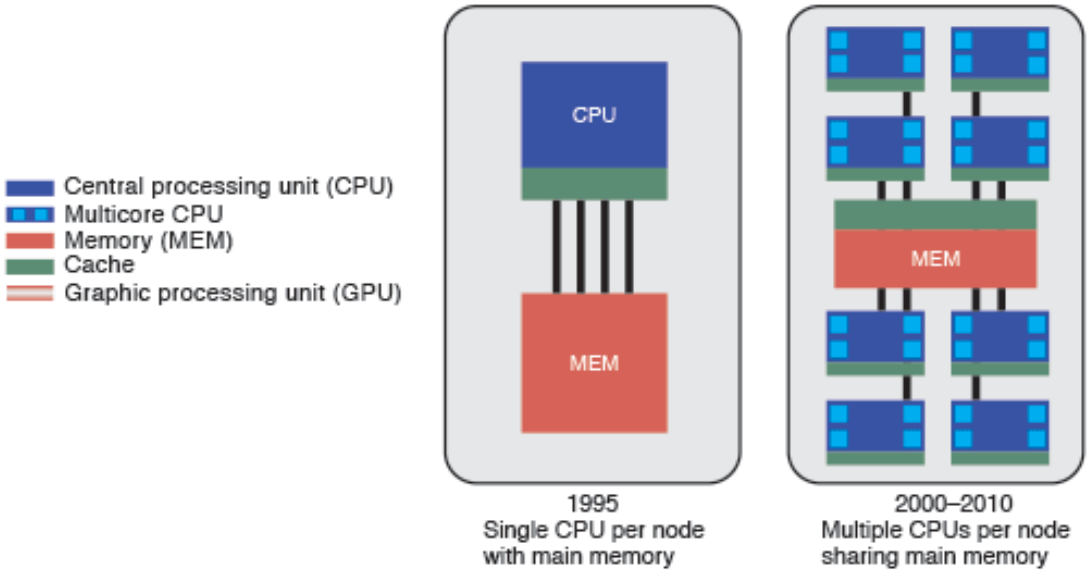
- Takeaway*: all* programs that need to run faster will have to become parallel programs
- Since mid 2000s - not only are fastest computers parallel, but nearly *all* computers are parallel

Evolution of HPC Nodes

https://str.llnl.gov/march-2015/still



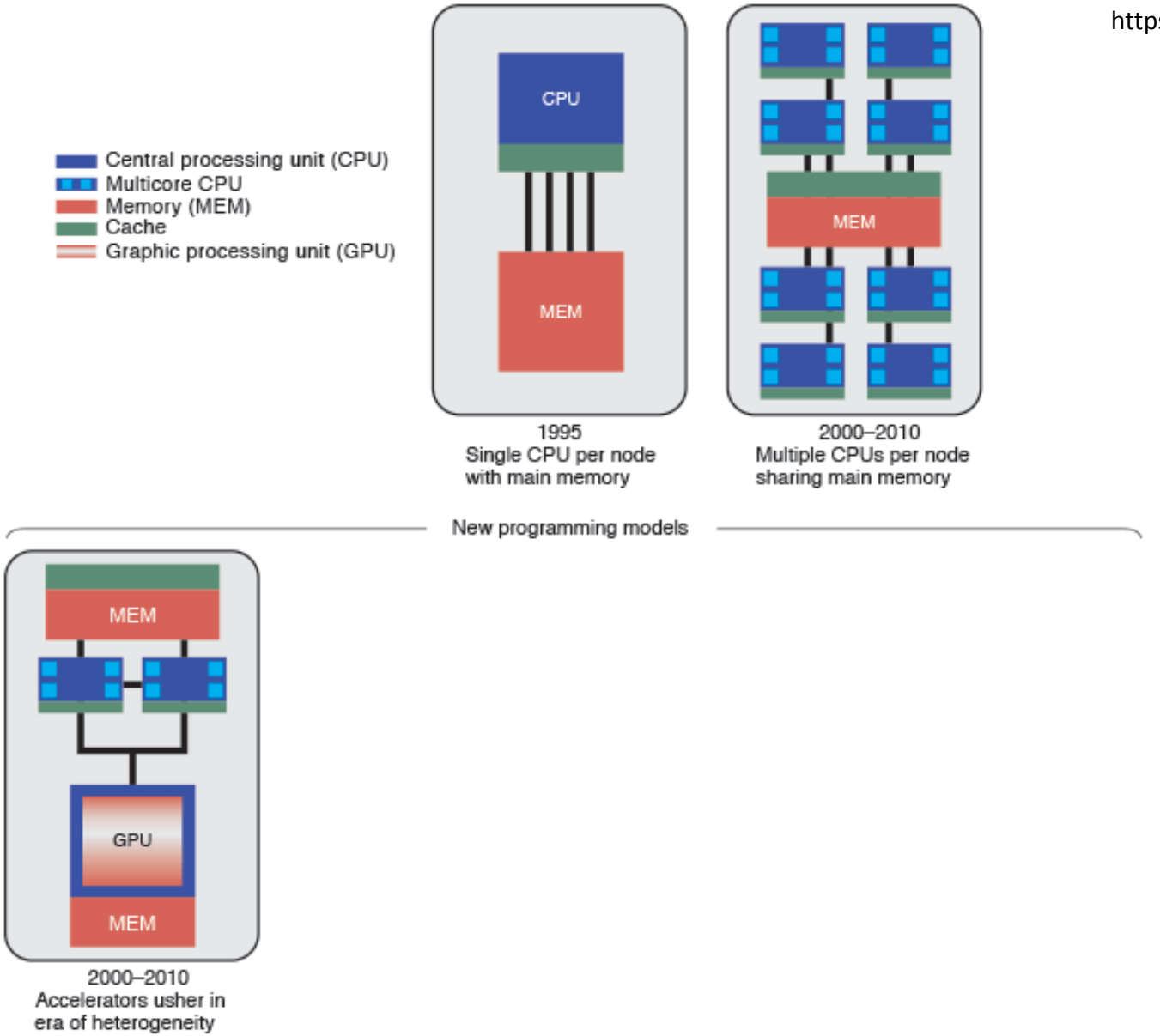
Evolution of HPC Nodes



https://str.llnl.gov/march-2015/still

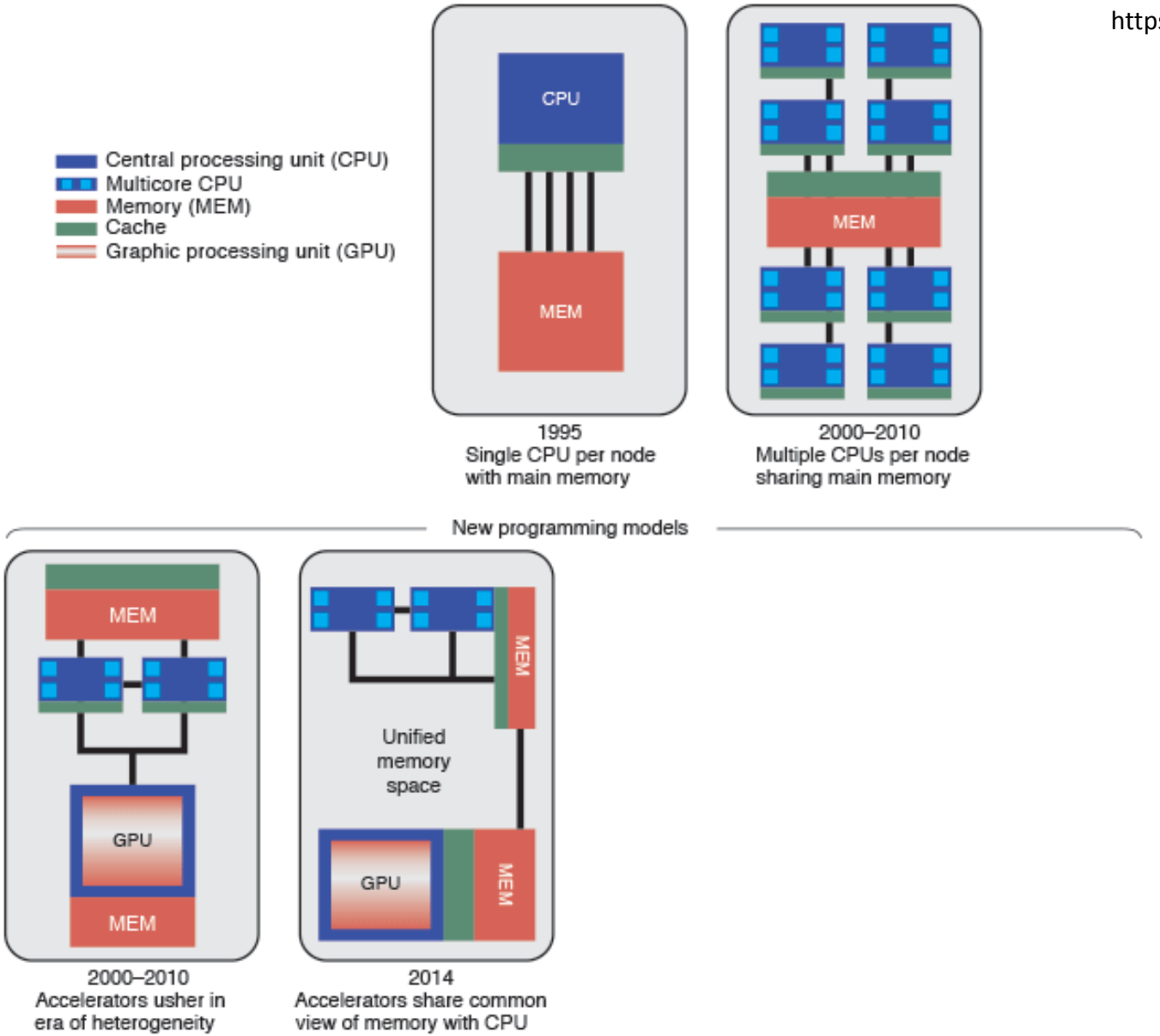
Evolution of HPC Nodes

https://str.llnl.gov/march-2015/still



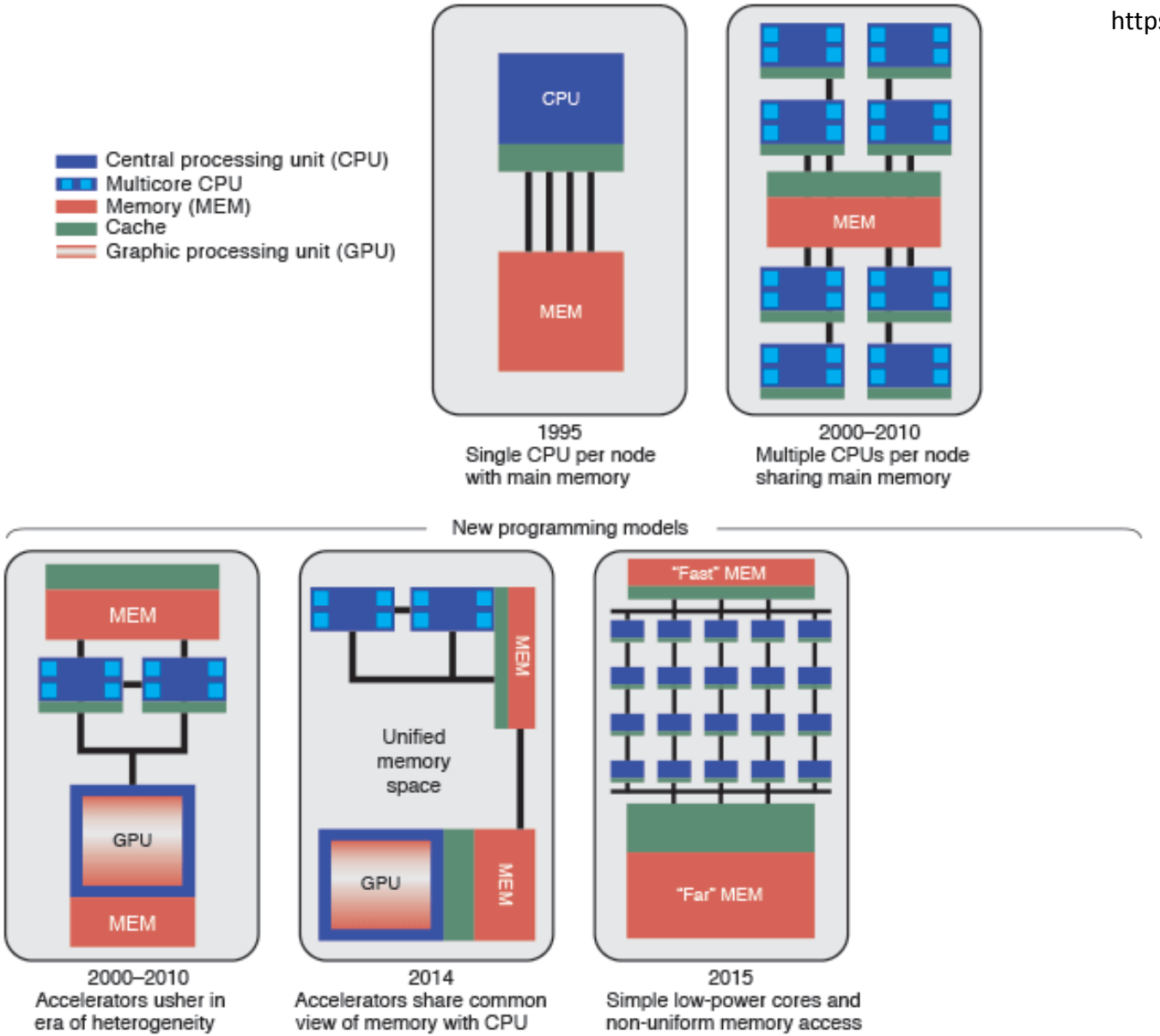
Evolution of HPC Nodes

https://str.llnl.gov/march-2015/still



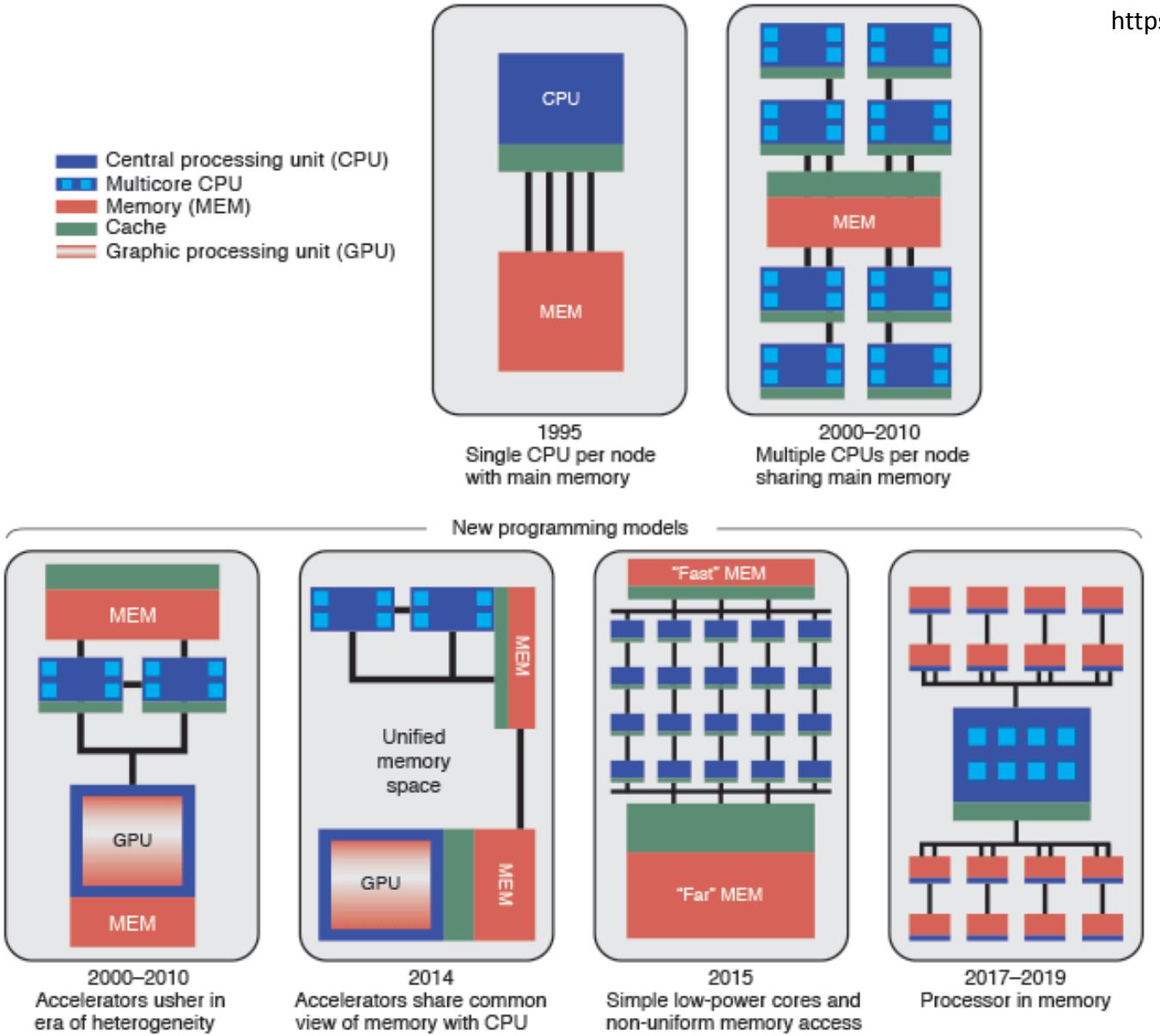
Evolution of HPC Nodes

https://str.llnl.gov/march-2015/still



Evolution of HPC Nodes

https://str.llnl.gov/march-2015/still



HPC Architectures Today

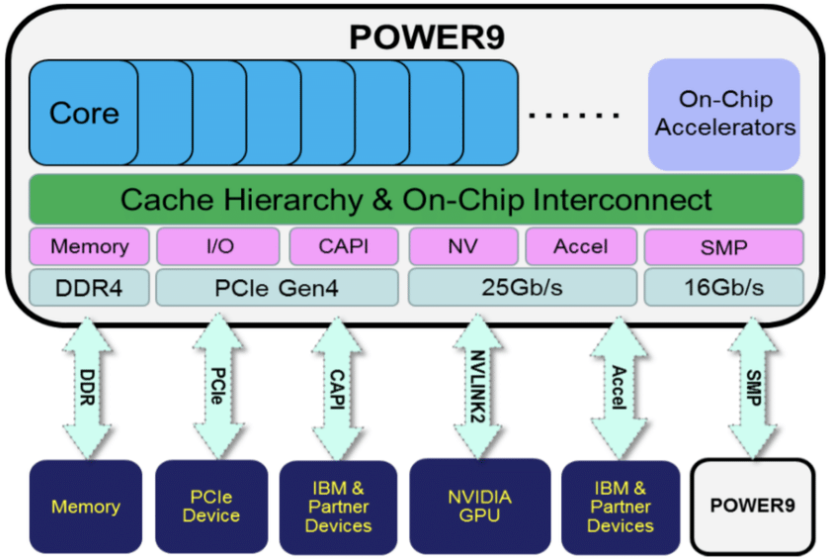
Summit (Oak Ridge National Lab, Tennessee)

- current #1 on the TOP500

HPC Architectures Today

One Processor: 22 SIMD processing cores, on-chip accelerators

- Each core supports 4 hardware threads
- Each core has separate L1 cache; pairs of cores share L2 and L3 cache



https://www.olcf.ornl.gov/wp-content/uploads/2018/12/summit_workshop_thompto.pdf

HPC Architectures Today

One GPU (NVIDIA V100): 80 streaming multiprocessors (SMs), 16 GB of high-bandwidth memory (HBM2), 6 MB L2 cache shared by SMs

https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide/#nvidia-v100-gpus

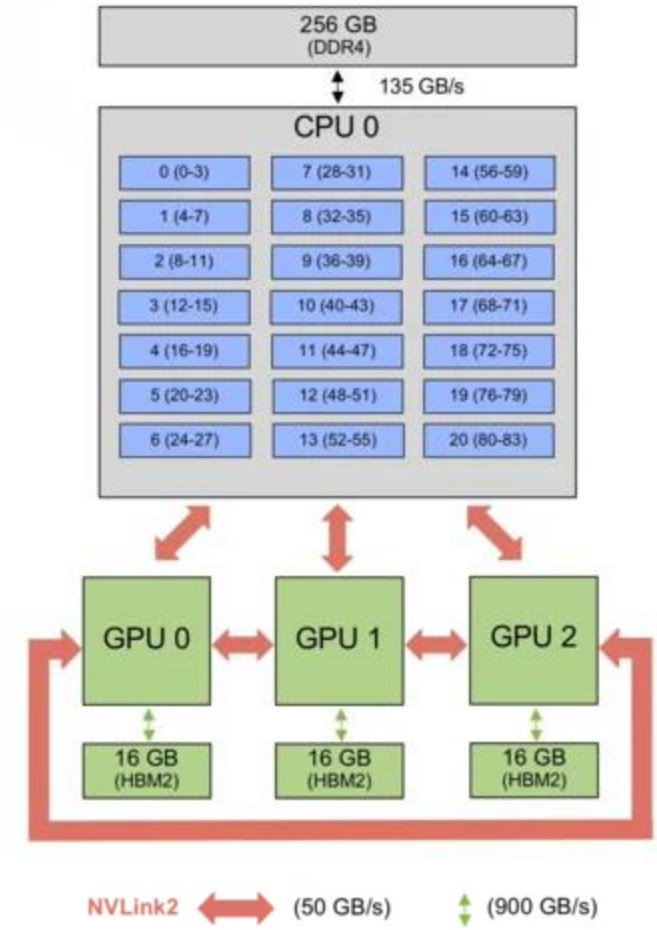
HPC Architectures Today

One SM:
32 FP64 (double-precision) cores,
64 FP32 (single-precision) cores,
64 INT32 cores,
8 tensor cores,
128-KB shared memory/L1 cache

https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide/#nvidia-v100-gpus

HPC Architectures Today

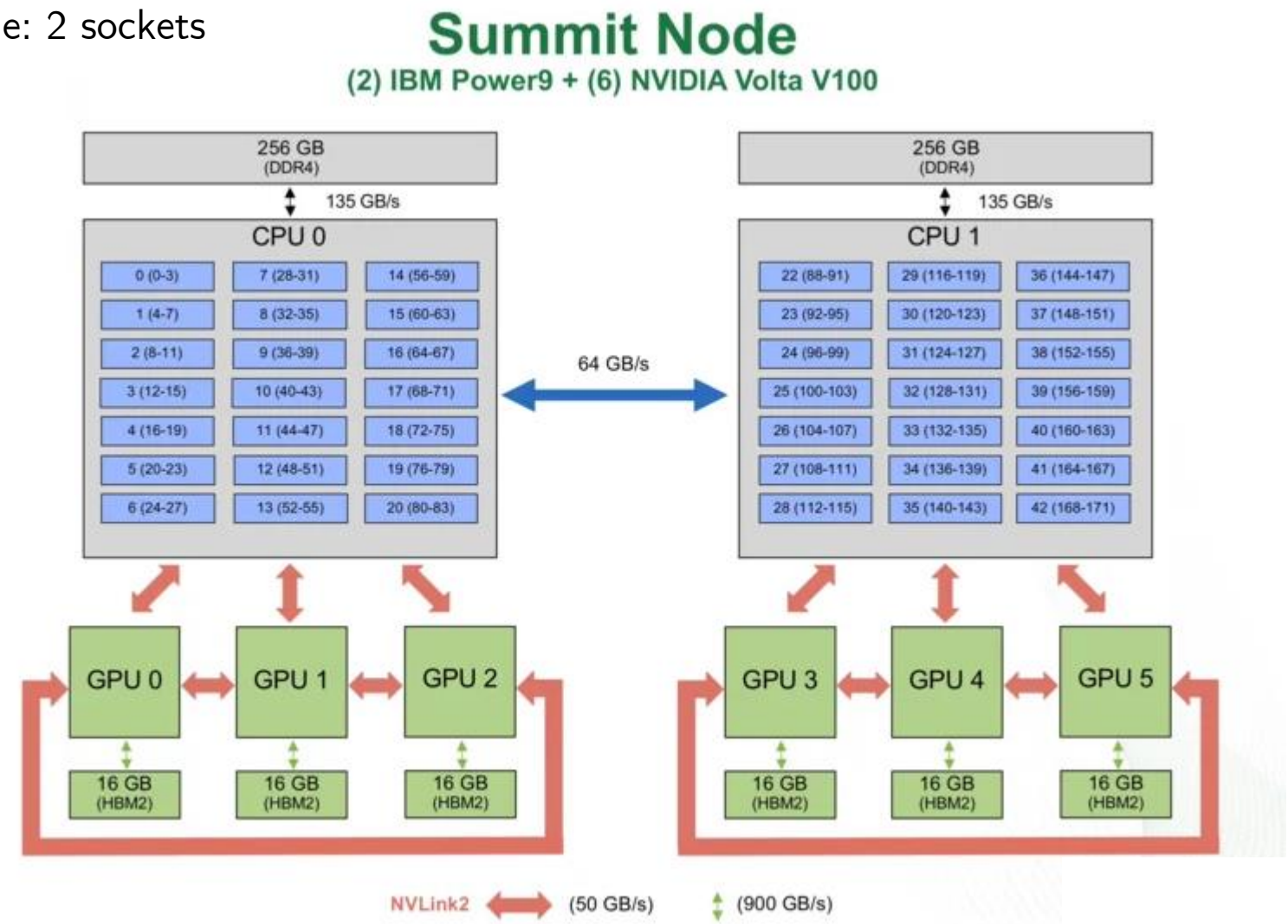
One Socket: 1 CPU, 3 GPUs



https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide

HPC Architectures Today

One Node: 2 sockets



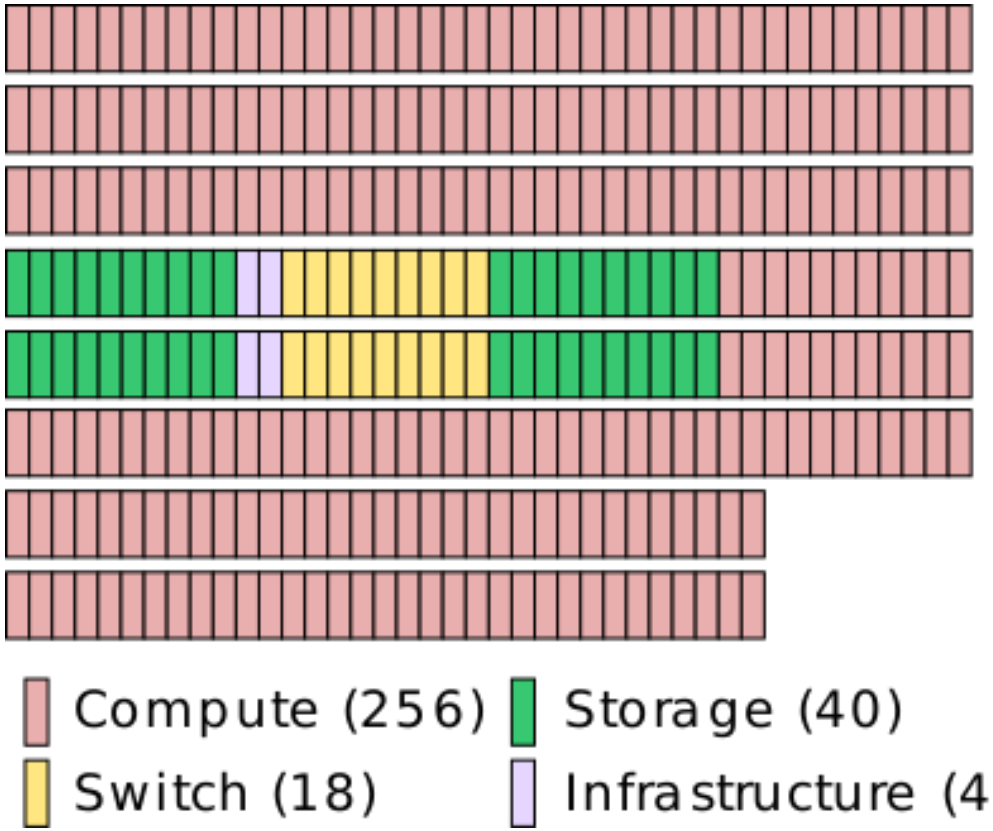
https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide

HPC Architectures Today

One Rack: 18 nodes

- Dual-rail EDR InfiniBand network with non-blocking fat-tree topology
- Node bandwidth of 23 GB/s

HPC Architectures Today



https://en.wikichip.org/wiki/supercomputers/summit

Designing High-Performance Parallel Algorithms

- To design an efficient parallel algorithm, must first model physical costs --- runtime or energy consumption --- of executing a program on a machine

- Tradeoff:
  - More detailed model: more accurate results for a particular machine, but results may not apply to other machines
  - Less detailed model: results applicable to a variety of machines, but may not be accurate for any
    - but abstracting machine details can still give us a general sense of an efficient implementation

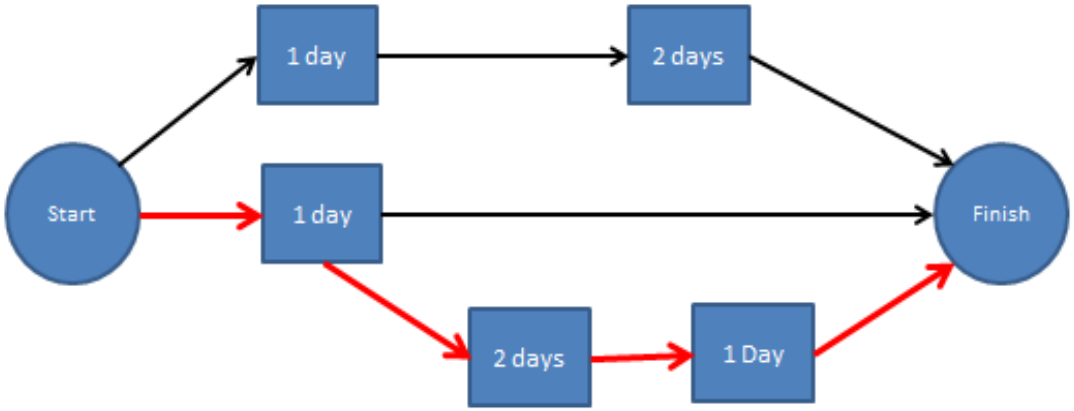
Performance Modeling: Latency-Bandwidth Model

A simplified runtime model:

- Time to perform a floating point operation: γ
- Time to move a message of n words: $\alpha + \beta n$
  - α = latency (seconds), β = 1/bandwidth (seconds/word)

$$\text{Runtime} = \gamma\ (\#\ \text{flops}) + \beta\ (\#\ \text{words}) + \alpha\ (\#\ \text{msgs})$$

#flops,words,msgs are counted along a critical path in the schedule:



Critical Path = 4 Days

Performance Modeling: Latency-Bandwidth Model

γ is per-flop:

- To improve: more parallelism (no longer increase clock frequency)

Performance Modeling: Latency-Bandwidth Model

γ is per-flop:

- To improve: more parallelism (no longer increase clock frequency)

β is per-word:

- Models bandwidth: maximum amount of data that can be in-flight simultaneously
- To improve: add more ports/wires/etc.

Performance Modeling: Latency-Bandwidth Model

γ is per-flop:

- To improve: more parallelism (no longer increase clock frequency)

β is per-word:

- Models bandwidth: maximum amount of data that can be in-flight simultaneously
- To improve: add more ports/wires/etc.

α is per-message and independent of message size

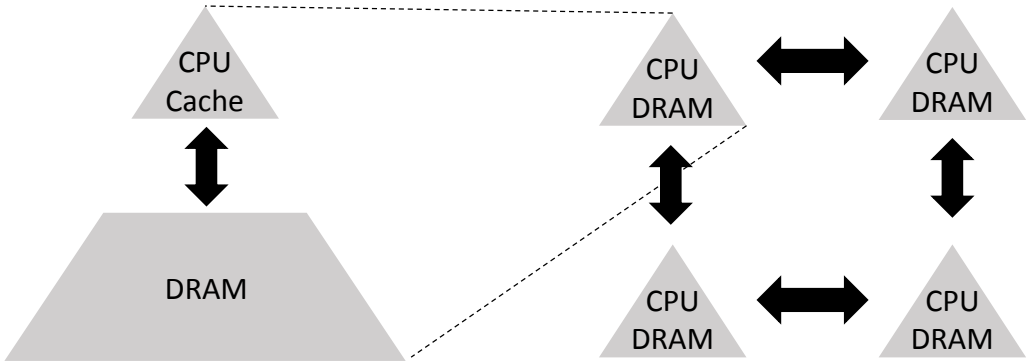
- Models latency: time for data to travel across machine
- Difficult to improve, due to fundamental limits (speed of light, atomic radius,...)

"Bandwidth is money, but latency is physics"

Exascale System Projections

	Today's Systems	Predicted Exascale Systems*
System Peak	10^{16} flops/s	10^{18} flops/s
Node Memory Bandwidth	10^{2} GB/s	10^{3} GB/s
Interconnect Bandwidth	10^{1} GB/s	10^{2} GB/s
Memory Latency	10^{-7} s	$5 \cdot 10^{-8}$ s
Interconnect Latency	10^{-6} s	$5 \cdot 10^{-7}$ s

*Sources: from P. Beckman (ANL), J. Shalf (LBL), and D. Unat (LBL)



Exascale System Projections

	Today's Systems	Predicted Exascale Systems*	Factor Improvement
System Peak	10^{16} flops/s	10^{18} flops/s	100
Node Memory Bandwidth	10^{2} GB/s	10^{3} GB/s	10
Interconnect Bandwidth	10^{1} GB/s	10^{2} GB/s	10
Memory Latency	10^{-7} s	$5 \cdot 10^{-8}$ s	2
Interconnect Latency	10^{-6} s	$5 \cdot 10^{-7}$ s	2

*Sources: from P. Beckman (ANL), J. Shalf (LBL), and D. Unat (LBL)

Exascale System Projections

	Today's Systems	Predicted Exascale Systems*	Factor Improvement
System Peak	10^{16} flops/s	10^{18} flops/s	100
Node Memory Bandwidth	10^{2} GB/s	10^{3} GB/s	10
Interconnect Bandwidth	10^{1} GB/s	10^{2} GB/s	10
Memory Latency	10^{-7} s	$5 \cdot 10^{-8}$ s	2
Interconnect Latency	10^{-6} s	$5 \cdot 10^{-7}$ s	2

*Sources: from P. Beckman (ANL), J. Shalf (LBL), and D. Unat (LBL)

- Movement of data (communication) is much more expensive than floating point operations (computation), in terms of both **time** and **energy**
- Gaps will only grow larger
- Reducing time spent moving data/waiting for data will be essential for applications at exascale!

Exascale Computing: The Modern Space Race

- "Exascale": 10^{18} floating point operations per second
  - with maximum energy consumption around 20-40 MWatts
- Advancing knowledge, addressing social challenges, improving quality of life, influencing policy, economic competitiveness

Nothing tends so much to the advancement of knowledge as the application of a new instrument.
- Sir Humphry Davy

- Large investment in HPC worldwide

Exascale Computing: The Modern Space Race

- "Exascale": 10^{18} floating point operations per second
  - with maximum energy consumption around 20-40 MWatts
- Advancing knowledge, addressing social challenges, improving quality of life, influencing policy, economic competitiveness

Nothing tends so much to the advancement of knowledge as the application of a new instrument.
- Sir Humphry Davy

- Large investment in HPC worldwide

- Technical challenges at all levels

hardware to algorithms to applications

Exascale Computing: The Modern Space Race

- "Exascale": 10^{18} floating point operations per second
  - with maximum energy consumption around 20-40 MWatts
- Advancing knowledge, addressing social challenges, improving quality of life, influencing policy, economic competitiveness

Nothing tends so much to the advancement of knowledge as the application of a new instrument.
- Sir Humphry Davy

- Large investment in HPC worldwide

- Technical challenges at all levels

hardware to algorithms to applications

An Exaflop of what?

- When will victory be declared?
  - When a supercomputer reaches exaflop performance on the LINPACK benchmark (TOP500)
    - Solving dense $Ax = b$ using Gaussian elimination with partial pivoting

An Exaflop of what?

- When will victory be declared?
  - When a supercomputer reaches exaflop performance on the LINPACK benchmark (TOP500)
    - Solving dense $Ax = b$ using Gaussian elimination with partial pivoting
  - Summit supercomputer has already exceeded exaflop performance for a certain genomics code (https://www.olcf.ornl.gov/2018/06/08/genomics-code-exceeds-exaops-on-summit-supercomputer/)

An Exaflop of what?

- When will victory be declared?
  - When a supercomputer reaches exaflop performance on the LINPACK benchmark (TOP500)
    - Solving dense $Ax = b$ using Gaussian elimination with partial pivoting
  - Summit supercomputer has already exceeded exaflop performance for a certain genomics code (https://www.olcf.ornl.gov/2018/06/08/genomics-code-exceeds-exaops-on-summit-supercomputer/)

- Does that mean we are done?

An Exaflop of what?

- When will victory be declared?
  - When a supercomputer reaches exaflop performance on the LINPACK benchmark (TOP500)
    - Solving dense $Ax = b$ using Gaussian elimination with partial pivoting
  - Summit supercomputer has already exceeded exaflop performance for a certain genomics code (https://www.olcf.ornl.gov/2018/06/08/genomics-code-exceeds-exaops-on-summit-supercomputer/)

- Does that mean we are done?
- LINPACK benchmark is typically a compute-bound problem ("BLAS-3")
- Not a good indication of performance for a large number of scientific applications!
  - Lots of remaining work even after exascale performance is achieved
  - Has led to incorporation of other benchmarks into the TOP500 ranking
    - e.g., HPCG: Solving sparse $Ax = b$ iteratively using the conjugate gradient method

Krylov subspace methods

- Linear systems $Ax = b$, eigenvalue problems, singular value problems, least squares, etc.
- Best for: A large & very sparse, stored implicitly, or only approximation needed

- Krylov Subspace Method is a projection process onto the Krylov subspace

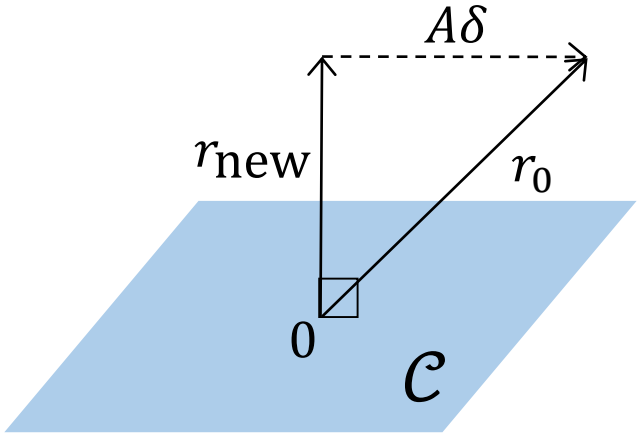
$$\mathcal{K}_i(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \dots, A^{i-1} r_0\}$$

where A is an $N \times N$ matrix and $r_0 = b - Ax_0$ is a length-N vector

- In each iteration,
  - Add a dimension to the Krylov subspace
    - Forms nested sequence of Krylov subspaces

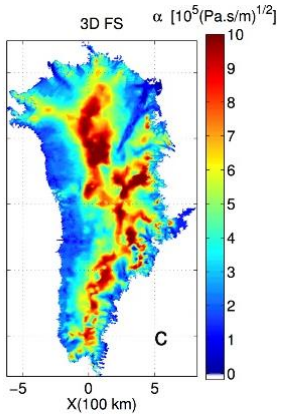
$$\mathcal{K}_1(A, r_0) \subset \mathcal{K}_2(A, r_0) \subset \cdots \subset \mathcal{K}_i(A, r_0)$$

  - Orthogonalize (with respect to some $\mathcal{C}_i$)
  - Select approximate solution $x_i \in x_0 + \mathcal{K}_i(A, r_0)$ using $r_i = b - Ax_i \perp \mathcal{C}_i$



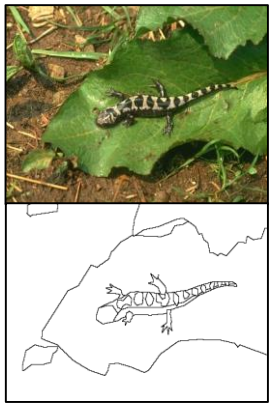
- Ex: Lanczos/Conjugate Gradient (CG), Arnoldi/Generalized Minimum Residual (GMRES), Biconjugate Gradient (BICG), BICGSTAB, GKL, LSQR, etc.

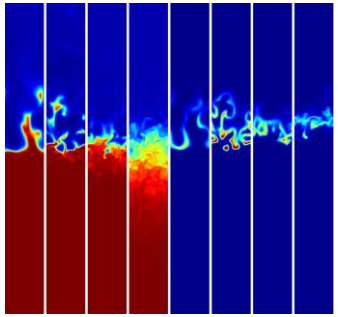
Krylov Subspace Methods in the Wild



Climate Modeling

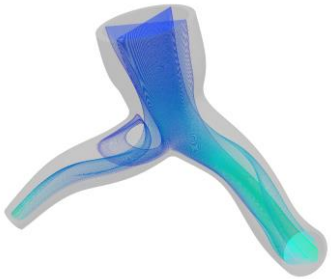
Computer Vision

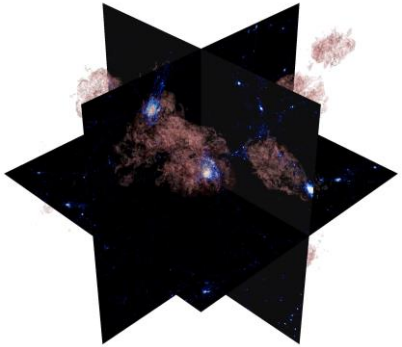




Chemical Engineering

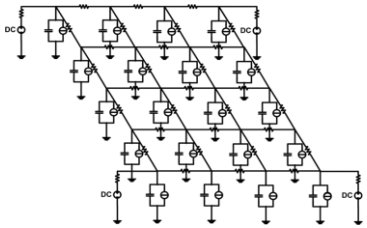
Medical Treatment

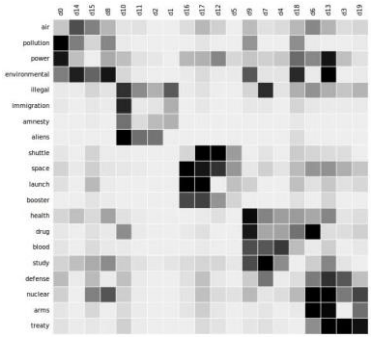




Computational Cosmology

Power Grid Modeling





Latent Semantic Analysis

Financial Portfolio Optimization

The conjugate gradient method

A is symmetric positive definite, $\mathcal{C}_i = \mathcal{K}_i(A, r_0)$

The conjugate gradient method

A is symmetric positive definite, $\mathcal{C}_i = \mathcal{K}_i(A, r_0)$

$$r_i \perp \mathcal{K}_i(A, r_0) \quad \Leftrightarrow \quad \|x - x_i\|_A = \min_{z \in x_0 + \mathcal{K}_i(A, r_0)} \|x - z\|_A$$

The conjugate gradient method

A is symmetric positive definite, $\mathcal{C}_i = \mathcal{K}_i(A, r_0)$

$$r_i \perp \mathcal{K}_i(A, r_0) \quad \Leftrightarrow \quad \|x - x_i\|_A = \min_{z \in x_0 + \mathcal{K}_i(A, r_0)} \|x - z\|_A$$

$$\Longrightarrow \quad r_{N+1} = 0$$

The conjugate gradient method

A is symmetric positive definite, $\mathcal{C}_i = \mathcal{K}_i(A, r_0)$

$$r_i \perp \mathcal{K}_i(A, r_0) \quad \Leftrightarrow \quad \|x - x_i\|_A = \min_{z \in x_0 + \mathcal{K}_i(A, r_0)} \|x - z\|_A$$

$$\Longrightarrow \quad r_{N+1} = 0$$

Connection with Lanczos

- With $v_1 = r_0/\|r_0\|$, i iterations of Lanczos produces $N \times i$ matrix $V_i = [v_1, \dots, v_i]$, and $i \times i$ tridiagonal matrix T_i such that
$$AV_i = V_i T_i + \delta_{i+1} v_{i+1} e_i^T, \qquad T_i = V_i^* A V_i$$
- CG approximation x_i is obtained by solving the reduced model
$$T_i y_i = \|r_0\| e_1, \qquad x_i = x_0 + V_i y_i$$

The conjugate gradient method

A is symmetric positive definite, $\mathcal{C}_i = \mathcal{K}_i(A, r_0)$

$$r_i \perp \mathcal{K}_i(A, r_0) \quad \Leftrightarrow \quad \|x - x_i\|_A = \min_{z \in x_0 + \mathcal{K}_i(A, r_0)} \|x - z\|_A$$

$$\Longrightarrow \quad r_{N+1} = 0$$

Connection with Lanczos

- With $v_1 = r_0/\|r_0\|$, i iterations of Lanczos produces $N \times i$ matrix $V_i = [v_1, \dots, v_i]$, and $i \times i$ tridiagonal matrix T_i such that
$$AV_i = V_i T_i + \delta_{i+1} v_{i+1} e_i^T, \qquad T_i = V_i^* A V_i$$
- CG approximation x_i is obtained by solving the reduced model
$$T_i y_i = \|r_0\| e_1, \qquad x_i = x_0 + V_i y_i$$
- Connections with orthogonal polynomials, Stieltjes problem of moments, Gauss-Cristoffel quadrature, others (see 2013 book of Liesen and Strakoš)

The conjugate gradient method

A is symmetric positive definite, $\mathcal{C}_i = \mathcal{K}_i(A, r_0)$

$$r_i \perp \mathcal{K}_i(A, r_0) \quad \Leftrightarrow \quad \|x - x_i\|_A = \min_{z \in x_0 + \mathcal{K}_i(A, r_0)} \|x - z\|_A$$

$$\Longrightarrow \quad r_{N+1} = 0$$

Connection with Lanczos

- With $v_1 = r_0/\|r_0\|$, i iterations of Lanczos produces $N \times i$ matrix $V_i = [v_1, \dots, v_i]$, and $i \times i$ tridiagonal matrix T_i such that
$$AV_i = V_i T_i + \delta_{i+1} v_{i+1} e_i^T, \qquad T_i = V_i^* A V_i$$
- CG approximation x_i is obtained by solving the reduced model
$$T_i y_i = \|r_0\| e_1, \qquad x_i = x_0 + V_i y_i$$
- Connections with orthogonal polynomials, Stieltjes problem of moments, Gauss-Cristoffel quadrature, others (see 2013 book of Liesen and Strakoš)

⇒ CG (and other Krylov subspace methods) are highly nonlinear

- Good for convergence, bad for ease of finite precision analysis

Implementation of CG

- Standard implementation due to Hestenes and Stiefel (1952) (HSCG)
- Uses three 2-term recurrences for updating x_i, r_i, p_i

$r_0 = b - Ax_0, \;\; p_0 = r_0$

for i = 1:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end

Implementation of CG

- Standard implementation due to Hestenes and Stiefel (1952) (HSCG)
- Uses three 2-term recurrences for updating x_i, r_i, p_i

$r_0 = b - Ax_0, \quad p_0 = r_0$

for $i = 1$:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end

minimizes $\|x - x_i\|_A$ along line
$z(\alpha) = x_{i-1} + \alpha p_{i-1}$

Implementation of CG

- Standard implementation due to Hestenes and Stiefel (1952) (HSCG)
- Uses three 2-term recurrences for updating x_i, r_i, p_i

$r_0 = b - Ax_0, \quad p_0 = r_0$

for $i = 1$:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end

minimizes $\|x - x_i\|_A$ along line

$$z(\alpha) = x_{i-1} + \alpha p_{i-1}$$

If

$$p_i \perp_A p_j \text{ for } i \neq j,$$

1-dimensional minimizations in each iteration give i-dimensional minimization over the whole subspace

$$x_0 + \mathcal{K}_i(A, r_0) = x_0 + \text{span}\{p_0, \dots p_{i-1}\}$$

Conjugate Gradient on the World's Fastest Computer

Summit - IBM Power System AC922

Site:	Oak Ridge National Laboratory
Manufacturer:	IBM
Cores:	2,282,544
Memory:	2,801,664 GB
Processor:	IBM POWER9 22C 3.07GHz
Interconnect:	Dual-rail Mellanox EDR Infiniband
Performance	
Theoretical peak:	187,659 TFlops/s
LINPACK benchmark:	122,300 Tflops/s
HPCG benchmark:	2,926 Tflops/s

Conjugate Gradient on the World's Fastest Computer

Summit - IBM Power System AC922

current #1 on top500

Site:	Oak Ridge National Laboratory
Manufacturer:	IBM
Cores:	2,282,544
Memory:	2,801,664 GB
Processor:	IBM POWER9 22C 3.07GHz
Interconnect:	Dual-rail Mellanox EDR Infiniband
Performance	
Theoretical peak:	187,659 TFlops/s
LINPACK benchmark:	122,300 Tflops/s
HPCG benchmark:	2,926 Tflops/s

Conjugate Gradient on the World's Fastest Computer

Summit - IBM Power System AC922

current #1 on top500

Site:	Oak Ridge National Laboratory
Manufacturer:	IBM
Cores:	2,282,544
Memory:	2,801,664 GB
Processor:	IBM POWER9 22C 3.07GHz
Interconnect:	Dual-rail Mellanox EDR Infiniband
Performance	
Theoretical peak:	187,659 TFlops/s
LINPACK benchmark:	122,300 Tflops/s
HPCG benchmark:	2,926 Tflops/s

LINPACK benchmark
(dense $Ax = b$, direct)
65% efficiency

Conjugate Gradient on the World's Fastest Computer

Summit - IBM Power System AC922

current #1 on top500

Site:	Oak Ridge National Laboratory
Manufacturer:	IBM
Cores:	2,282,544
Memory:	2,801,664 GB
Processor:	IBM POWER9 22C 3.07GHz
Interconnect:	Dual-rail Mellanox EDR Infiniband
Performance	
Theoretical peak:	187,659 TFlops/s
LINPACK benchmark:	122,300 Tflops/s
HPCG benchmark:	2,926 Tflops/s

LINPACK benchmark
(dense $Ax = b$, direct)
65% efficiency

HPCG benchmark
(sparse $Ax = b$, iterative)
1.5% efficiency

The Conjugate Gradient (CG) Method

$r_0 = b - Ax_0, \quad p_0 = r_0$

for i = 1:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

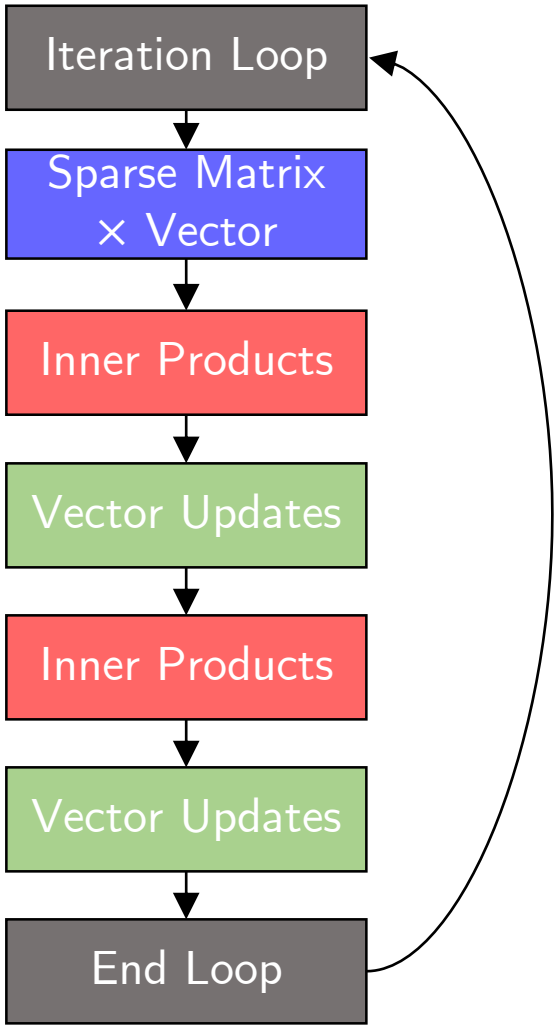
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end



The Conjugate Gradient (CG) Method

$r_0 = b - Ax_0, \quad p_0 = r_0$

for $i = 1$:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T Ap_{i-1}}$$

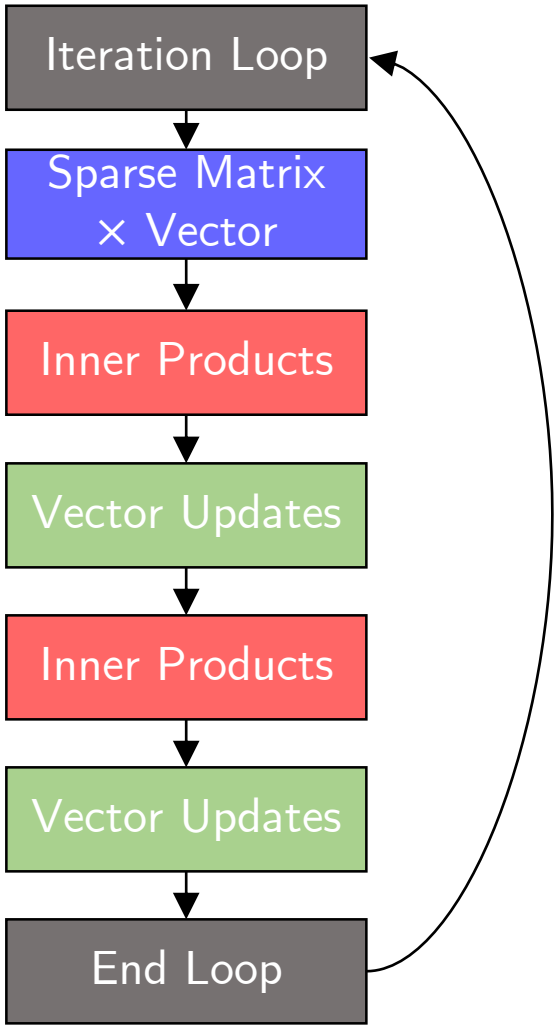
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} Ap_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end



The Conjugate Gradient (CG) Method

$r_0 = b - Ax_0, \quad p_0 = r_0$

for $i = 1$:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

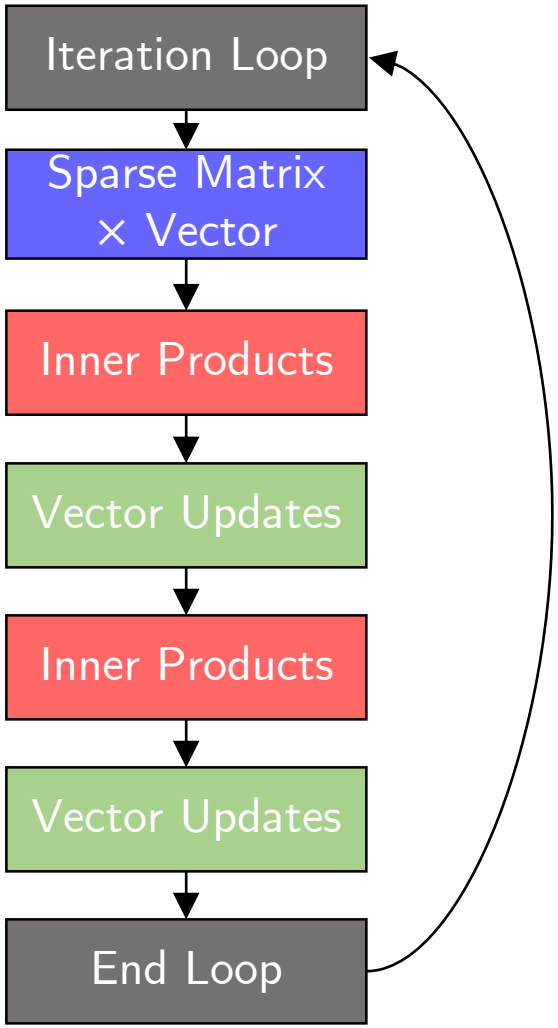
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end



The Conjugate Gradient (CG) Method

$r_0 = b - Ax_0, \quad p_0 = r_0$

for i = 1:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

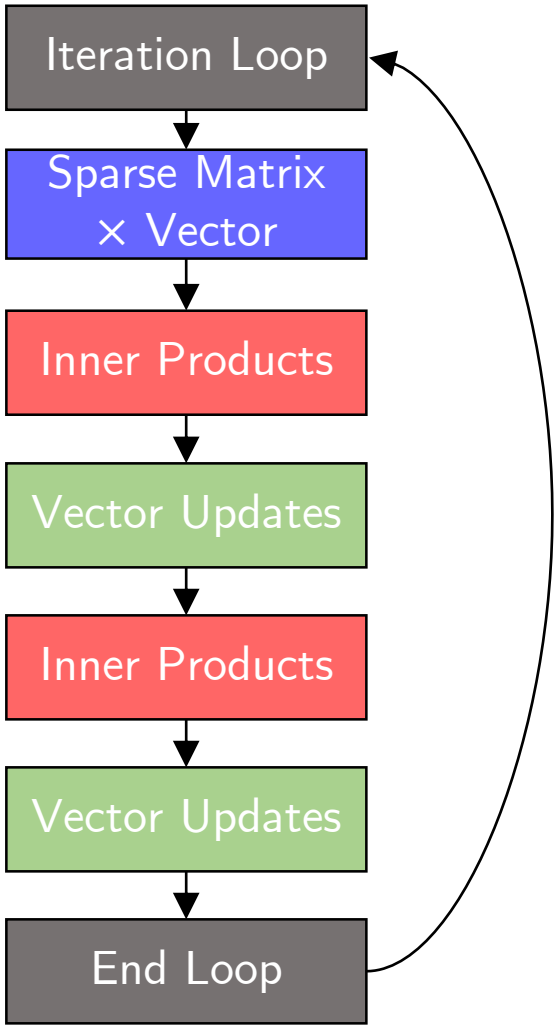
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end



The Conjugate Gradient (CG) Method

$r_0 = b - Ax_0, \quad p_0 = r_0$

for $i = 1$:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

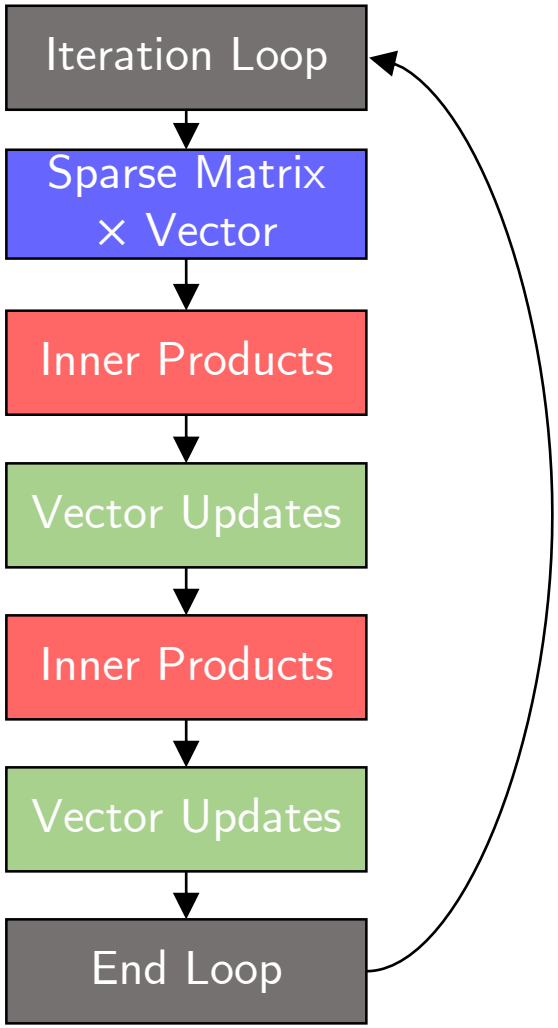
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

end



The Conjugate Gradient (CG) Method

$r_0 = b - Ax_0, \quad p_0 = r_0$

for $i = 1$:nmax

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}$$

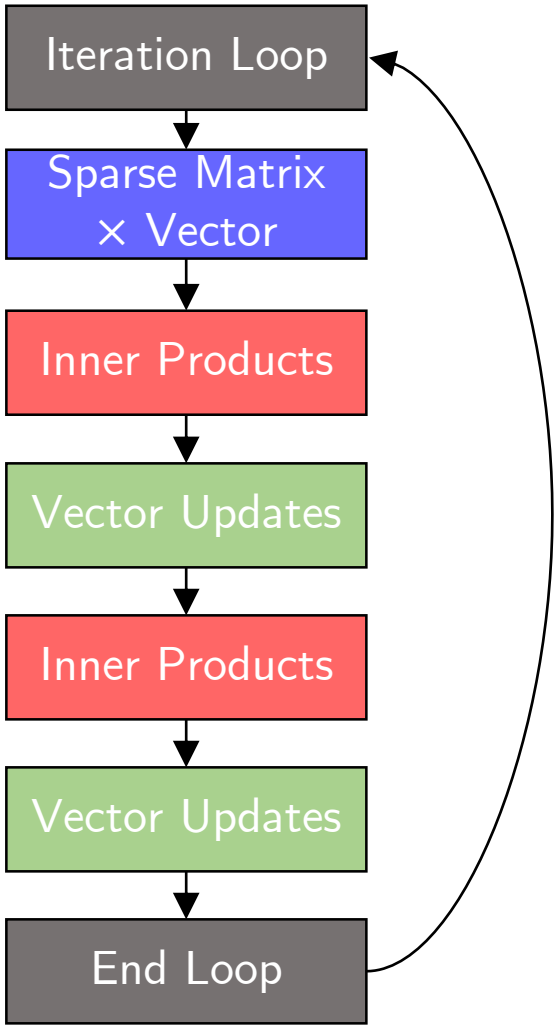
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$p_i = r_i + \beta_i p_{i-1}$$

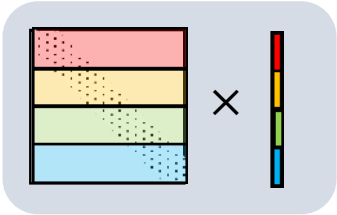
end



Cost Per Iteration

→ Sparse matrix-vector multiplication (SpMV)

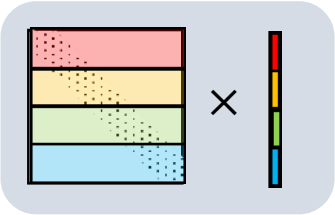
- $O(\text{nnz})$ flops
- Must communicate vector entries w/neighboring processors (nearest neighbor MPI collective)
- Must read A/vector from slow memory



Cost Per Iteration

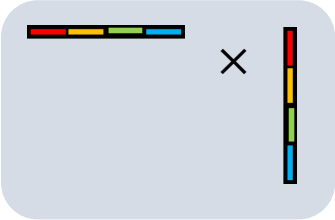
→ Sparse matrix-vector multiplication (SpMV)

- $O(\text{nnz})$ flops
- Must communicate vector entries w/neighboring processors (nearest neighbor MPI collective)
- Must read A/vector from slow memory



→ Inner products

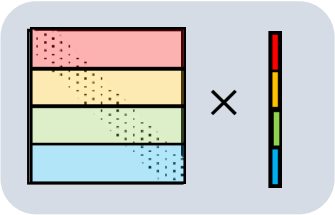
- $O(N)$ flops
- global synchronization (MPI_Allreduce)
  - all processors must exchange data and wait for *all* communication to finish before proceeding
- Multiple reads/writes to slow memory



Cost Per Iteration

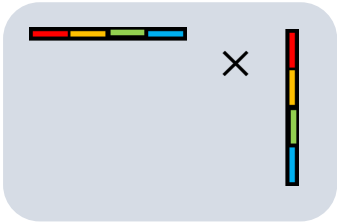
→ Sparse matrix-vector multiplication (SpMV)

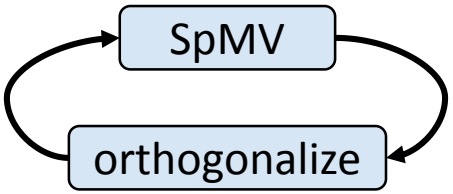
- $O(\text{nnz})$ flops
- Must communicate vector entries w/neighboring processors (nearest neighbor MPI collective)
- Must read A/vector from slow memory



→ Inner products

- $O(N)$ flops
- global synchronization (MPI_Allreduce)
  - all processors must exchange data and wait for *all* communication to finish before proceeding
- Multiple reads/writes to slow memory





Low computation/communication ratio

⇒ **Performance is communication-bound**

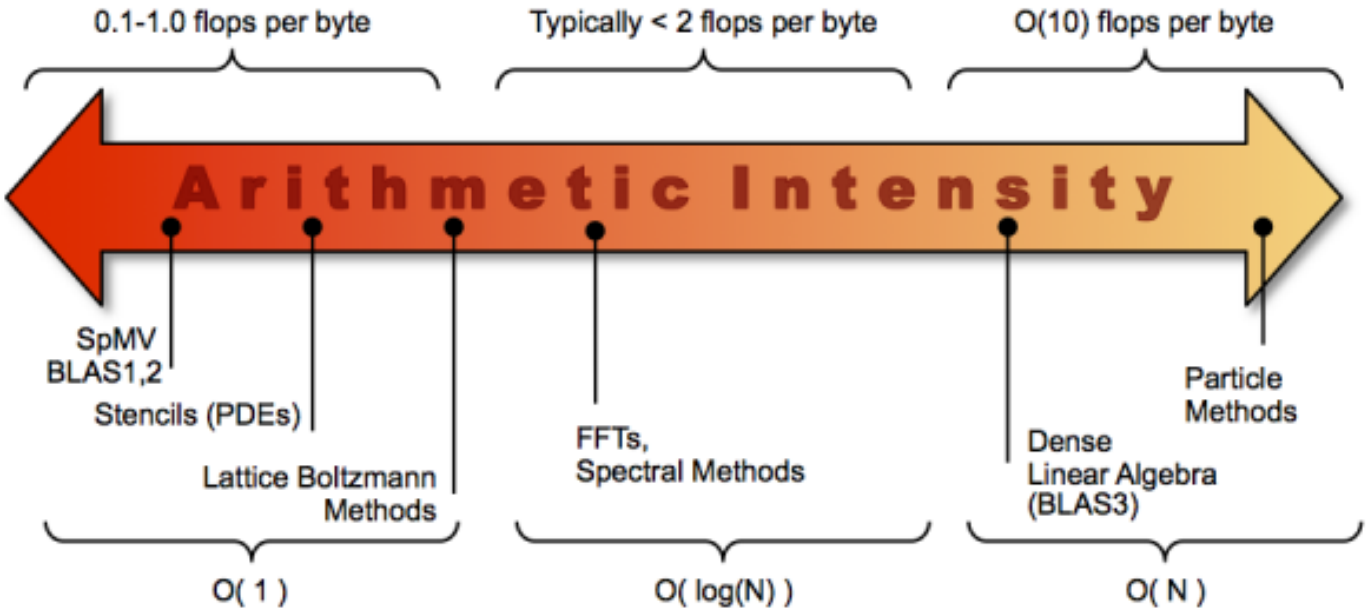


Image source: Sam Williams

Roofline Model Example

Roofline Model (Williams, Waterman, Patterson, 2009)

- Provides estimates of performance for various applications (based on arithmetic intensity) for given machine
- attainable flop/s = min(peak flop/s, peak bandwidth × arithmetic intensity)
- "ceilings" give peak bandwidth or peak flops in absence of possible optimizations

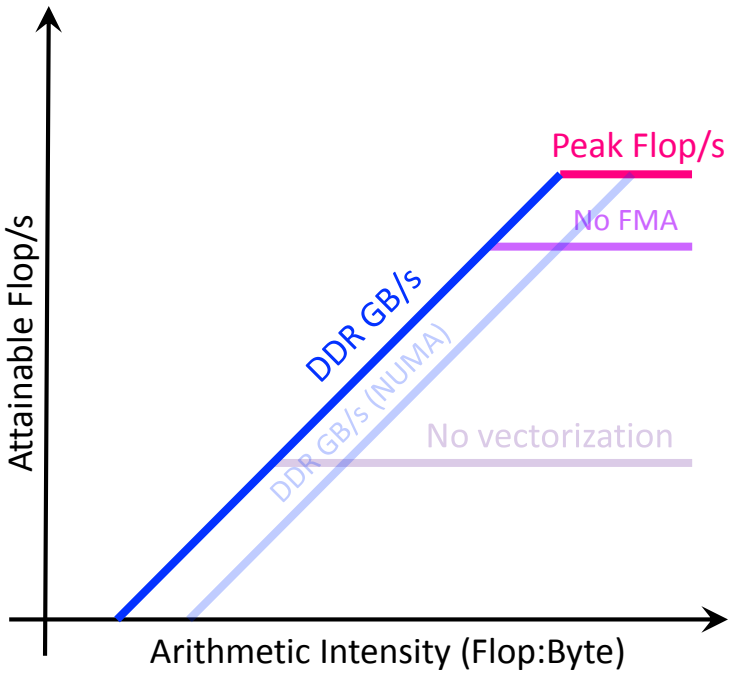


Image source: Sam Williams

Roofline Model Example

Roofline Model (Williams, Waterman, Patterson, 2009)

- Provides estimates of performance for various applications (based on arithmetic intensity) for given machine
- attainable flop/s = min(peak flop/s, peak bandwidth × arithmetic intensity)
- "ceilings" give peak bandwidth or peak flops in absence of possible optimizations

Generally three approaches to improving performance:

- Maximize in-core performance (e.g. get compiler to vectorize)

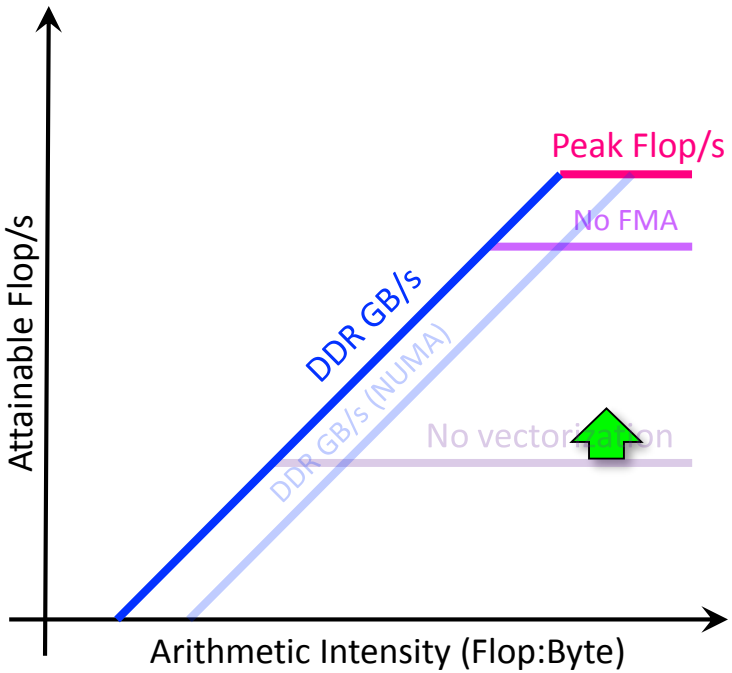
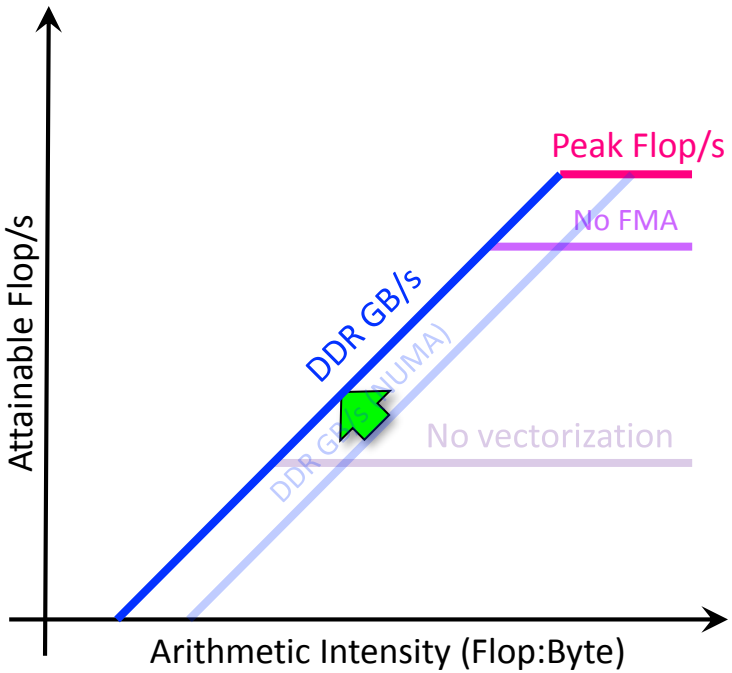


Image source: Sam Williams

Roofline Model Example

Roofline Model (Williams, Waterman, Patterson, 2009)

- Provides estimates of performance for various applications (based on arithmetic intensity) for given machine
- attainable flop/s = min(peak flop/s, peak bandwidth × arithmetic intensity)
- "ceilings" give peak bandwidth or peak flops in absence of possible optimizations



Generally three approaches to improving performance:

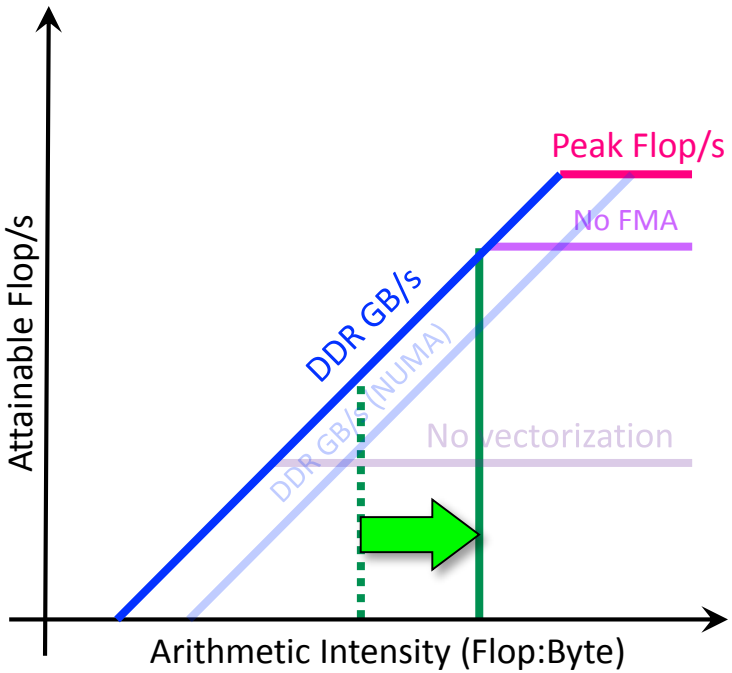
- Maximize in-core performance (e.g. get compiler to vectorize)
- Maximize memory bandwidth (e.g. NUMA-aware allocation)

Image source: Sam Williams

Roofline Model Example

Roofline Model (Williams, Waterman, Patterson, 2009)

- Provides estimates of performance for various applications (based on arithmetic intensity) for given machine
- attainable flop/s = min(peak flop/s, peak bandwidth × arithmetic intensity)
- "ceilings" give peak bandwidth or peak flops in absence of possible optimizations



Generally three approaches to improving performance:

- Maximize in-core performance (e.g. get compiler to vectorize)
- Maximize memory bandwidth (e.g. NUMA-aware allocation)
- Minimize data movement (increase AI)

Image source: Sam Williams

Synchronization-reducing variants

Motivated many approaches to reducing synchronization (increasing ratio of computation to communication) in CG:

Synchronization-reducing variants

Motivated many approaches to reducing synchronization (increasing ratio of computation to communication) in CG:

- Early work: CG with a single synchronization point per iteration
  - 3-term recurrence CG
  - Using modified computation of recurrence coefficients
  - Using auxiliary vectors

Synchronization-reducing variants

Motivated many approaches to reducing synchronization (increasing ratio of computation to communication) in CG:

- Early work: CG with a single synchronization point per iteration
  - 3-term recurrence CG
  - Using modified computation of recurrence coefficients
  - Using auxiliary vectors
- Pipelined Krylov subspace methods
  - Uses modified coefficients and auxiliary vectors to reduce synchronization points to 1 per iteration
  - Modifications also allow decoupling of SpMV and inner products - enables overlapping (MPI non-blocking collectives)

Synchronization-reducing variants

Motivated many approaches to reducing synchronization (increasing ratio of computation to communication) in CG:

- Early work: CG with a single synchronization point per iteration
  - 3-term recurrence CG
  - Using modified computation of recurrence coefficients
  - Using auxiliary vectors
- Pipelined Krylov subspace methods
  - Uses modified coefficients and auxiliary vectors to reduce synchronization points to 1 per iteration
  - Modifications also allow decoupling of SpMV and inner products - enables overlapping (MPI non-blocking collectives)
- s-step Krylov subspace methods
  - Compute iterations in blocks of s using a different Krylov subspace basis
  - Enables one synchronization per s iterations

High Performance Krylov Subspace Methods

- To improve performance of Krylov subspace methods, we must reduce the cost of data movement
- Communication "hiding" approaches
  - Use non-blocking MPI communication
  - Do useful computation while waiting for communication (overlapping)
  - "Pipelined" Krylov subspace methods
    - Historical background, derivation
    - Performance results
    - Recent work on "deep pipelined" methods
- Communication "avoiding" approaches
  - Mathematically unroll iteration loop, allows all communication for multiple iterations to be done in one step
  - "s-step" Krylov subspace methods
    - Historical background, derivation
    - Implementation details (matrix powers kernel, TSQR)
    - Performance results
- Other approaches: enlarged KSMs, combination of pipelined and s-step approaches

Early approaches to reducing synchronization

- Goal: Reduce the 2 synchronization points per iteration in (HS)CG to 1 synchronization point per iteration

Early approaches to reducing synchronization

- Goal: Reduce the 2 synchronization points per iteration in (HS)CG to 1 synchronization point per iteration
- Compute β_i from α_{i-1} and Ap_{i-1} using relation

$$\|r_i\|^2 = \alpha_{i-1}^2 \|Ap_{i-1}\|^2 - \|r_{i-1}\|^2$$

  - Can then also merge the updates of x_i, r_i, and p_i
  - Developed independently by Johnson (1983, 1984), van Rosendale (1983, 1984), Saad (1985)
  - Many other similar approaches

Early approaches to reducing synchronization

- Goal: Reduce the 2 synchronization points per iteration in (HS)CG to 1 synchronization point per iteration
- Compute β_i from α_{i-1} and Ap_{i-1} using relation

$$\|r_i\|^2 = \alpha_{i-1}^2 \|Ap_{i-1}\|^2 - \|r_{i-1}\|^2$$

  - Can then also merge the updates of x_i, r_i, and p_i
  - Developed independently by Johnson (1983, 1984), van Rosendale (1983, 1984), Saad (1985)
  - Many other similar approaches

- Could also compute α_{i-1} from β_{i-1}:

$$\alpha_{i-1} = \left(\frac{r_{i-1}^T A r_{i-1}}{r_{i-1}^T r_{i-1}} - \frac{\beta_{i-1}}{\alpha_{i-2}} \right)^{-1}$$

CG with two three-term recurrences (STCG)

- HSCG recurrences can be written as

$$AP_i = R_{i+1}\underline{L}_i, \qquad R_i = P_i U_i$$

we can combine these to obtain a 3-term recurrence for the residuals (STCG):

$$AR_i = R_{i+1}\underline{T}_i, \qquad \underline{T}_i = \underline{L}_i U_i$$

CG with two three-term recurrences (STCG)

- HSCG recurrences can be written as

$$AP_i = R_{i+1}\underline{L}_i, \qquad R_i = P_i U_i$$

we can combine these to obtain a 3-term recurrence for the residuals (STCG):

$$AR_i = R_{i+1}\underline{T}_i, \qquad \underline{T}_i = \underline{L}_i U_i$$

- First developed by Stiefel (1952/53), also Rutishauser (1959) and Hageman and Young (1981)
- Motivated by relation to three-term recurrences for orthogonal polynomials

$r_0 = b - Ax_0, \quad p_0 = r_0, \quad x_{-1} = x_0, \quad r_{-1} = r_0, \quad e_{-1} = 0$

for $i = 1$:nmax

$$q_{i-1} = \frac{(r_{i-1}, Ar_{i-1})}{(r_{i-1}, r_{i-1})} - e_{i-2}$$

$$x_i = x_{i-1} + \frac{1}{q_{i-1}}\left(r_{i-1} + e_{i-2}(x_{i-1} - x_{i-2})\right)$$

$$r_i = r_{i-1} + \frac{1}{q_{i-1}}\left(-Ar_{i-1} + e_{i-2}(r_{i-1} - r_{i-2})\right)$$

$$e_{i-1} = q_{i-1}\frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$$

end

CG with two three-term recurrences (STCG)

- HSCG recurrences can be written as

$$AP_i = R_{i+1}\underline{L}_i, \qquad R_i = P_i U_i$$

we can combine these to obtain a 3-term recurrence for the residuals (STCG):

$$AR_i = R_{i+1}\underline{T}_i, \qquad \underline{T}_i = \underline{L}_i U_i$$

- First developed by Stiefel (1952/53), also Rutishauser (1959) and Hageman and Young (1981)
- Motivated by relation to three-term recurrences for orthogonal polynomials

$r_0 = b - Ax_0, \quad p_0 = r_0, \quad x_{-1} = x_0, \quad r_{-1} = r_0, \quad e_{-1} = 0$

for $i = 1$:nmax

$$q_{i-1} = \frac{(r_{i-1}, Ar_{i-1})}{(r_{i-1}, r_{i-1})} - e_{i-2}$$

$$x_i = x_{i-1} + \frac{1}{q_{i-1}}\left(r_{i-1} + e_{i-2}(x_{i-1} - x_{i-2})\right)$$

$$r_i = r_{i-1} + \frac{1}{q_{i-1}}\left(-Ar_{i-1} + e_{i-2}(r_{i-1} - r_{i-2})\right)$$

$$e_{i-1} = q_{i-1}\frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$$

end

Can be accomplished with a single synchronization point on parallel computers (Strakoš 1985, 1987)

CG with two three-term recurrences (STCG)

- HSCG recurrences can be written as

$$AP_i = R_{i+1}\underline{L}_i, \qquad R_i = P_i U_i$$

we can combine these to obtain a 3-term recurrence for the residuals (STCG):

$$AR_i = R_{i+1}\underline{T}_i, \qquad \underline{T}_i = \underline{L}_i U_i$$

- First developed by Stiefel (1952/53), also Rutishauser (1959) and Hageman and Young (1981)
- Motivated by relation to three-term recurrences for orthogonal polynomials

$r_0 = b - Ax_0, \quad p_0 = r_0, \quad x_{-1} = x_0, \quad r_{-1} = r_0, \quad e_{-1} = 0$

for $i = 1$:nmax

$$q_{i-1} = \frac{(r_{i-1}, Ar_{i-1})}{(r_{i-1}, r_{i-1})} - e_{i-2}$$

$$x_i = x_{i-1} + \frac{1}{q_{i-1}}\left(r_{i-1} + e_{i-2}(x_{i-1} - x_{i-2})\right)$$

$$r_i = r_{i-1} + \frac{1}{q_{i-1}}\left(-Ar_{i-1} + e_{i-2}(r_{i-1} - r_{i-2})\right)$$

$$e_{i-1} = q_{i-1}\frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$$

end

Can be accomplished with a single synchronization point on parallel computers (Strakoš 1985, 1987)

- Similar approach (computing α_i using β_{i-1}) used by D'Azevedo, Eijkhout, Romaine (1992, 1993)

Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = Ar_i$$

$$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$$

$$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

end

Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = Ar_i$$

$$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$$

$$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

end

Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = Ar_i$

$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$

$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

end

Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = Ar_i$$

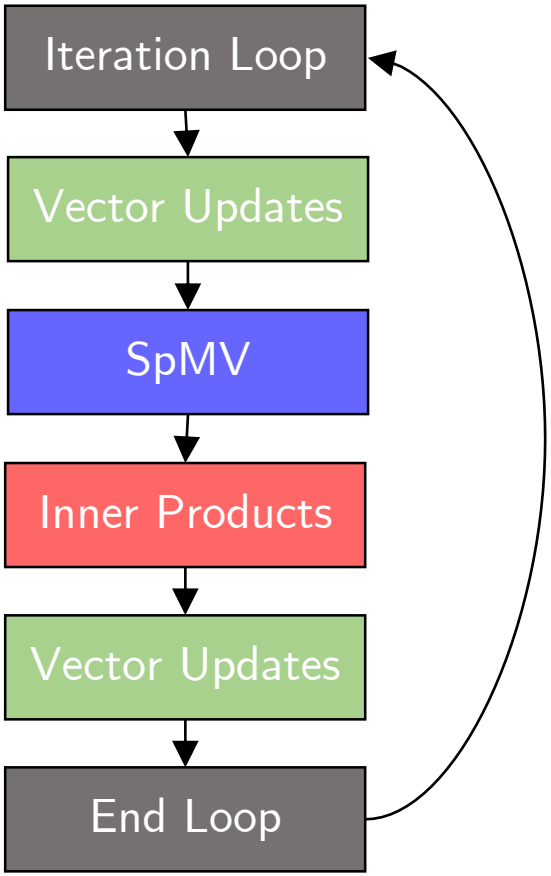
$$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$$

$$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

end



Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = Ar_i$

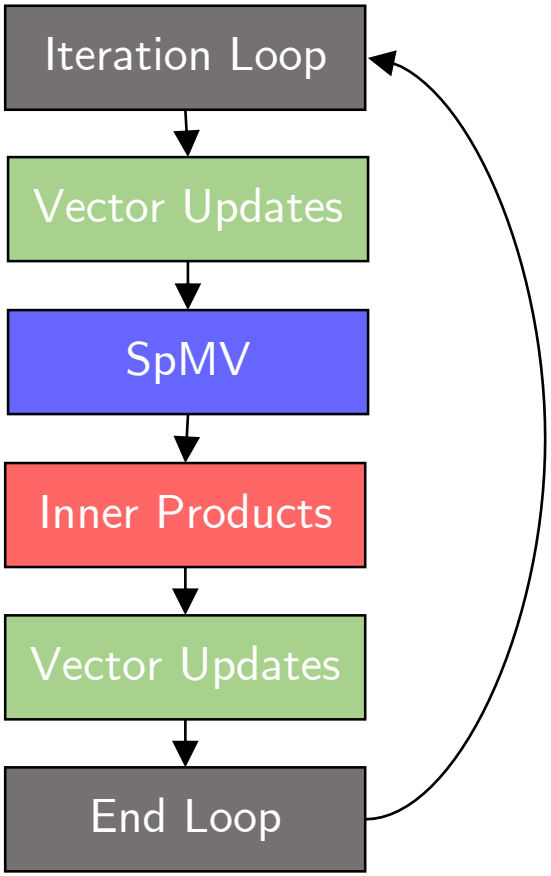
$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$

$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

end



Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = Ar_i$

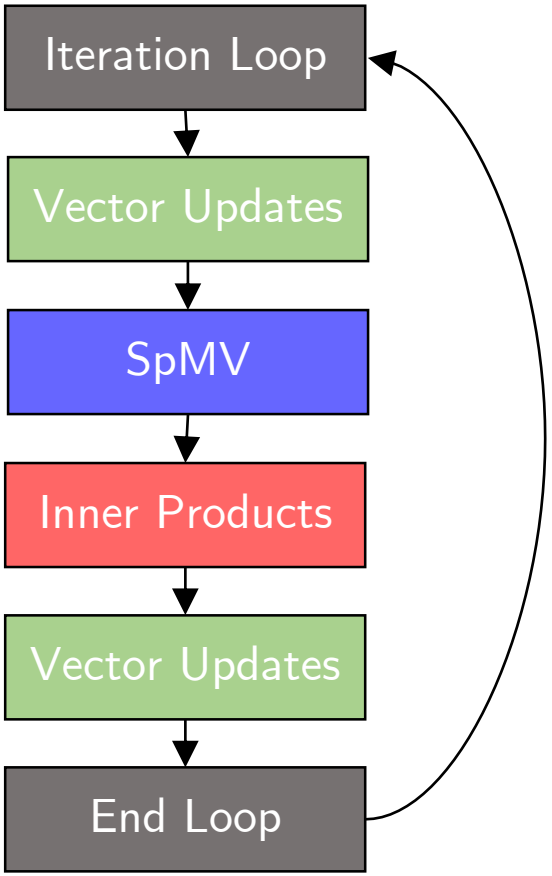
$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$

$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

end



Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = Ar_i$

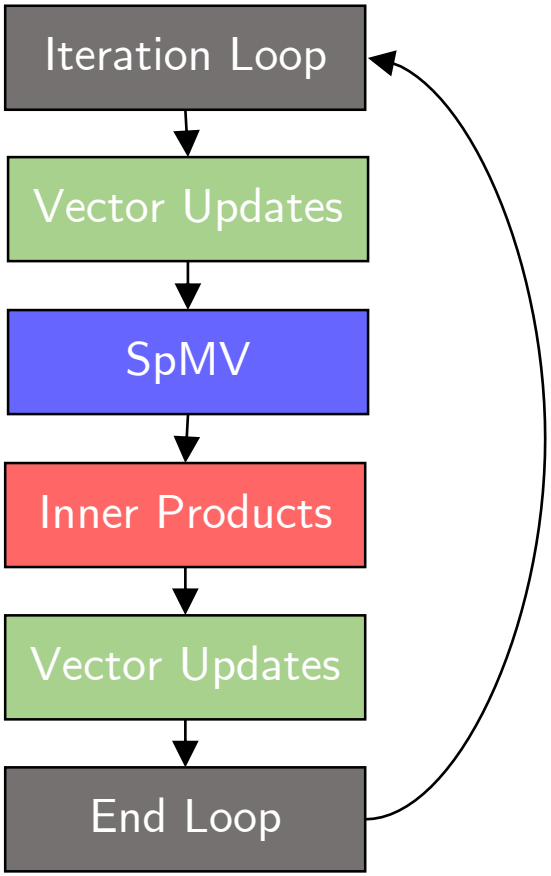
$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$

$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

end



Chronopoulos and Gear's CG (ChG CG)

- Chronopoulos and Gear (1989)
- Looks like HSCG, but very similar to 3-term recurrence CG (STCG)
- Reduces synchronizations/iteration to 1 by changing computation of α_i and using an auxiliary recurrence for Ap_i

$r_0 = b - Ax_0,\ \ p_0 = r_0,$

$s_0 = Ap_0,\ \ \alpha_0 = (r_0, r_0)/(p_0, s_0)$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = Ar_i$

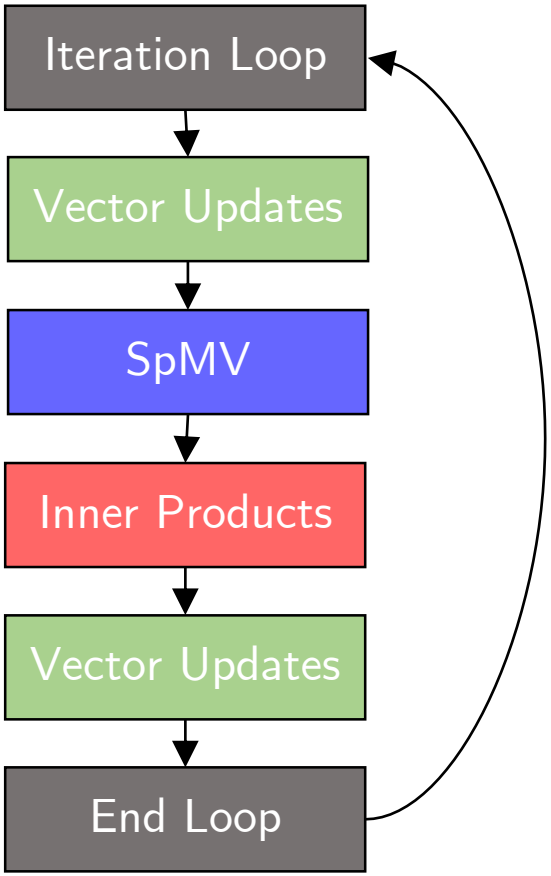
$\beta_i = \frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$

$\alpha_i = \frac{(r_i, r_i)}{(w_i, r_i) - (\beta_i/\alpha_{i-1})(r_i, r_i)}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

end



Pipelined CG (GVCG)

- Pipelined CG of Ghysels and Vanroose (2014)

- Similar to Chronopoulos and Gear approach
  - Uses auxiliary vector $s_i \equiv Ap_i$ and same formula for α_i

Pipelined CG (GVCG)

- Pipelined CG of Ghysels and Vanroose (2014)

- Similar to Chronopoulos and Gear approach
  - Uses auxiliary vector $s_i \equiv Ap_i$ and same formula for α_i

- Also uses auxiliary vectors for Ar_i and $A^2 r_i$ to remove sequential dependency between SpMV and inner products

Pipelined CG (GVCG)

- Pipelined CG of Ghysels and Vanroose (2014)

- Similar to Chronopoulos and Gear approach
  - Uses auxiliary vector $s_i \equiv Ap_i$ and same formula for α_i

- Also uses auxiliary vectors for Ar_i and $A^2 r_i$ to remove sequential dependency between SpMV and inner products

  - Allows the use of nonblocking (asynchronous) MPI communication to *overlap* SpMV and inner products

  - Hides the latency of global communications

GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$
$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$$

$$q_i = Aw_i$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i / \alpha_{i-1}) r_i^T r_i}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

$$z_i = q_i + \beta_i z_{i-1}$$

end

GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$

$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$$

$$q_i = Aw_i$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

$$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i / \alpha_{i-1}) r_i^T r_i}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

$$z_i = q_i + \beta_i z_{i-1}$$

end

GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$
$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$$

$$q_i = Aw_i$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

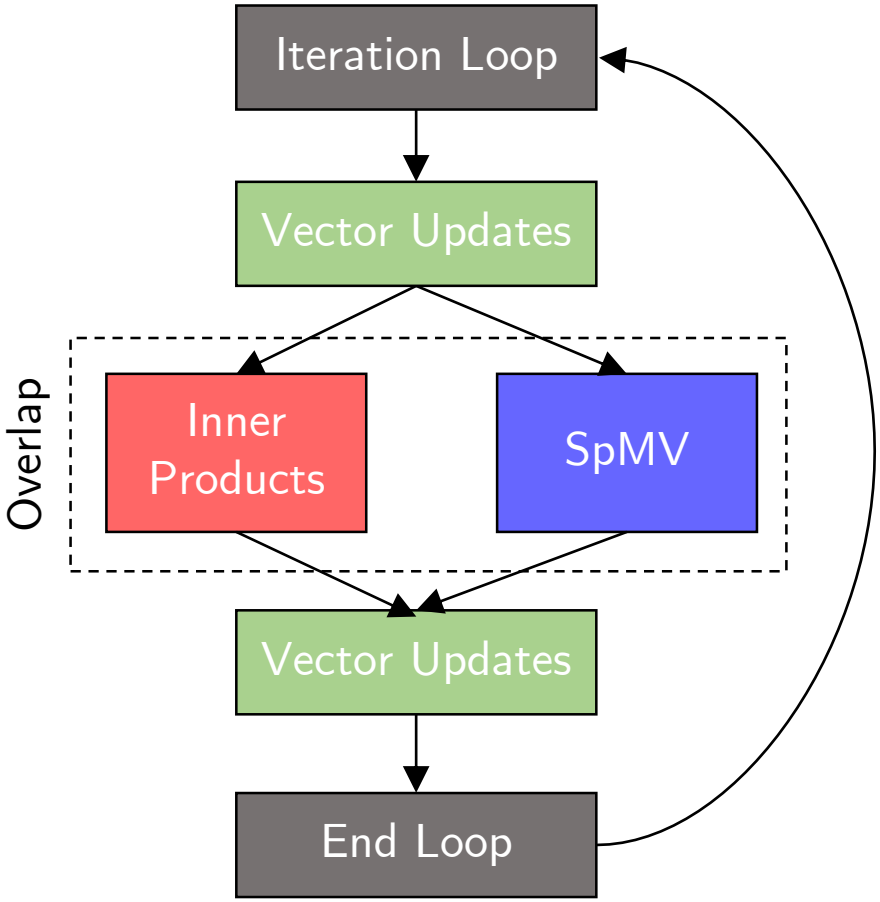
$$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i / \alpha_{i-1}) r_i^T r_i}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

$$z_i = q_i + \beta_i z_{i-1}$$

end



GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$
$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$

$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$

$$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$$

$$q_i = Aw_i$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

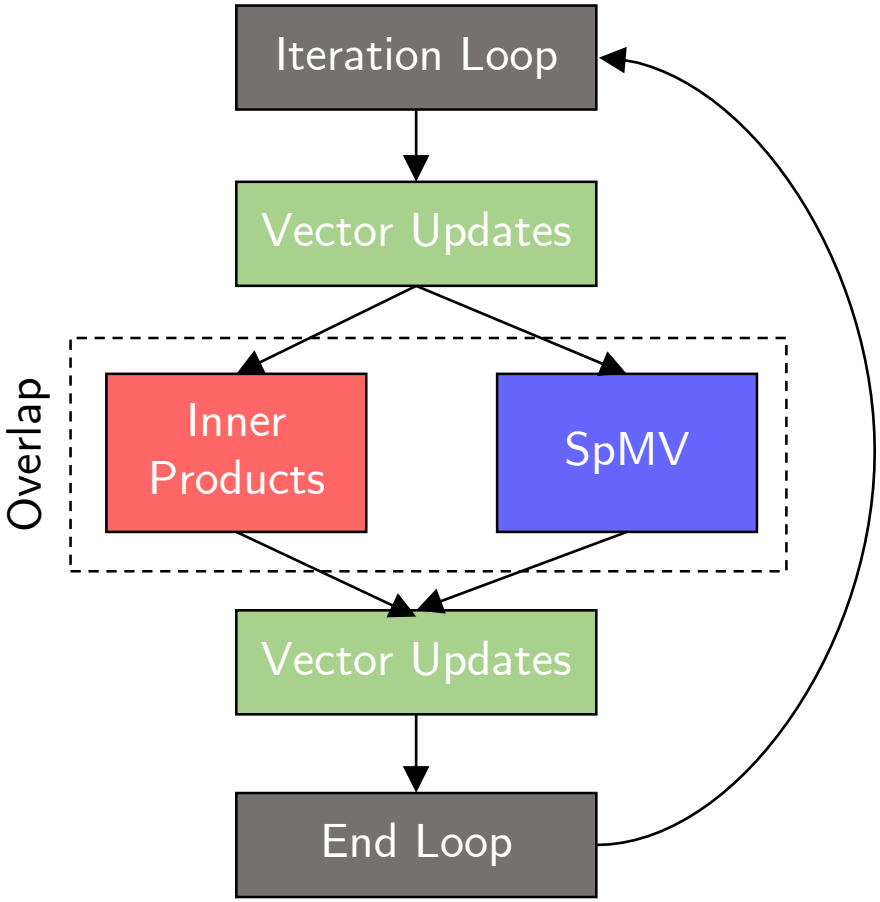
$$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i / \alpha_{i-1}) r_i^T r_i}$$

$$p_i = r_i + \beta_i p_{i-1}$$

$$s_i = w_i + \beta_i s_{i-1}$$

$$z_i = q_i + \beta_i z_{i-1}$$

end



GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$
$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$

$q_i = Aw_i$

$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$

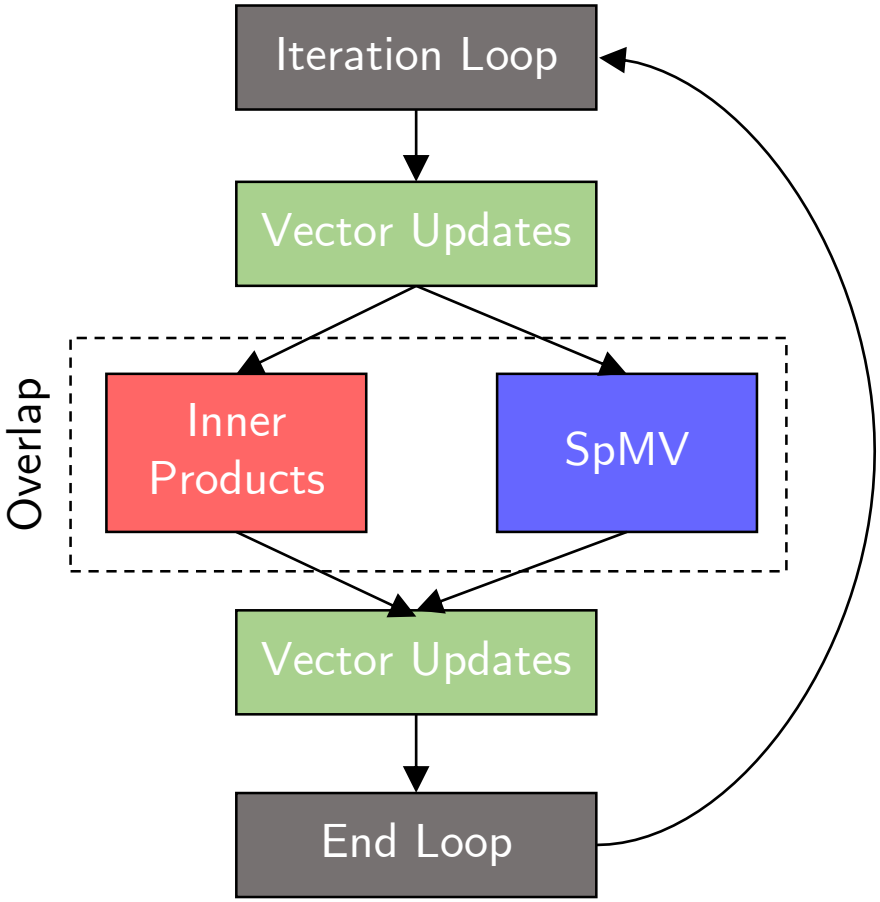
$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i / \alpha_{i-1}) r_i^T r_i}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

$z_i = q_i + \beta_i z_{i-1}$

end



GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$
$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$

$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$

$q_i = Aw_i$

$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$

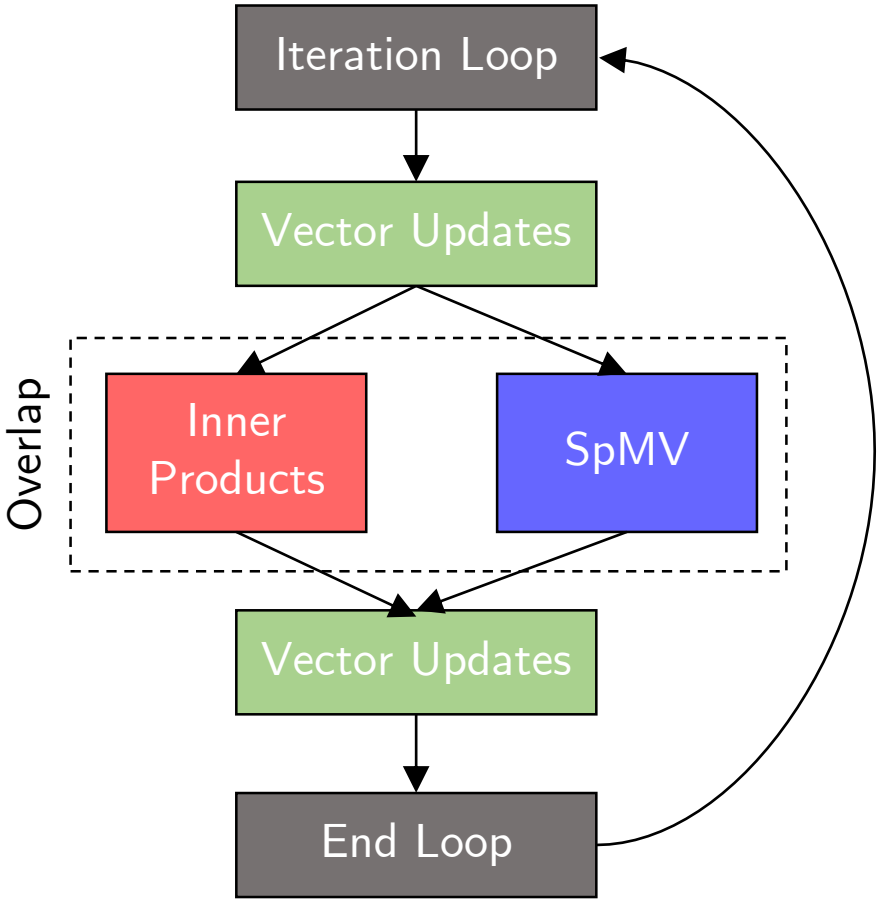
$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i / \alpha_{i-1}) r_i^T r_i}$

$p_i = r_i + \beta_i p_{i-1}$

$s_i = w_i + \beta_i s_{i-1}$

$z_i = q_i + \beta_i z_{i-1}$

end



GVCG (Ghysels and Vanroose 2014)

$r_0 = b - Ax_0,\ p_0 = r_0$

$s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0,$
$\alpha_0 = r_0^T r_0 / p_0^T s_0$

for i = 1:nmax

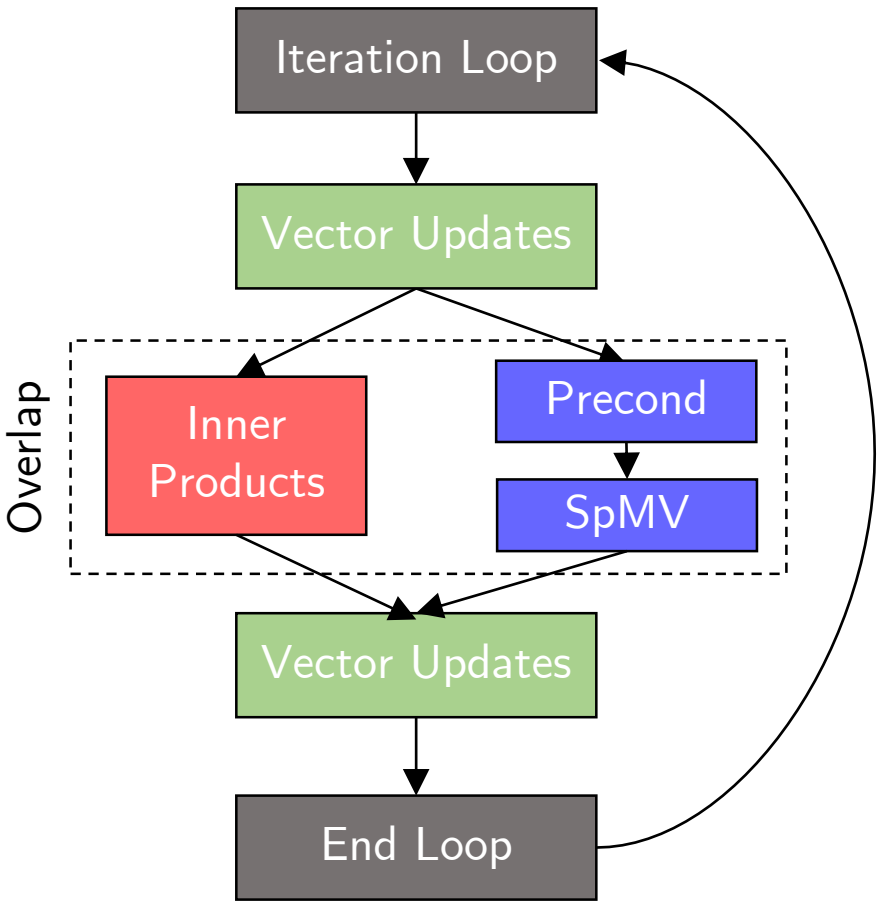
$$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$$
$$r_i = r_{i-1} - \alpha_{i-1} s_{i-1}$$
$$w_i = w_{i-1} - \alpha_{i-1} z_{i-1}$$

$$q_i = Aw_i$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$
$$\alpha_i = \frac{r_i^T r_i}{w_i^T r_i - (\beta_i/\alpha_{i-1}) r_i^T r_i}$$

$$p_i = r_i + \beta_i p_{i-1}$$
$$s_i = w_i + \beta_i s_{i-1}$$
$$z_i = q_i + \beta_i z_{i-1}$$

end

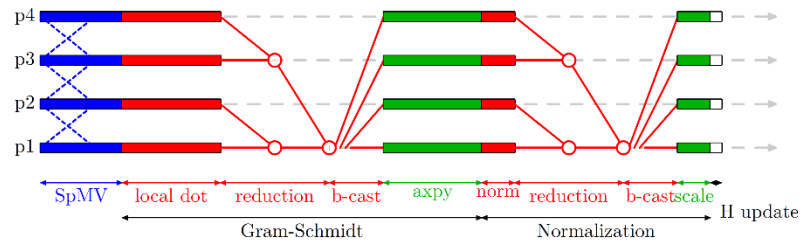


MPI Non-Blocking Communication

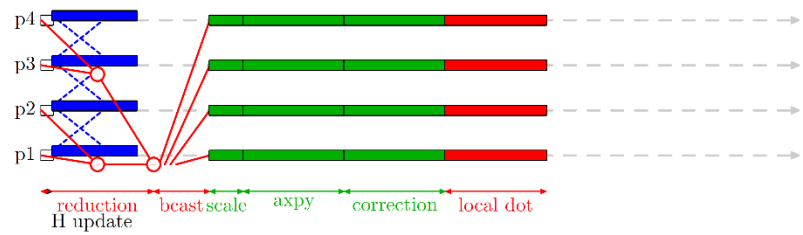
- "Non-blocking" or "asynchronous" collectives available since MPI 3

```
MPI_Iallreduce(...,MPI_Request,...)
// ...other work (SpMV, precond., etc)
MPI_Wait(...,MPI_Request)
```

Classical GMRES



Pipelined GMRES



P. Ghysels, et al. SIAM J. Scientific Computing, 35(1):C48C71, (2013).

PETSc provides a construct for asynchronous dot-products:

```
VecDotBegin (...,&dot);
PetscCommSplitReductionBegin (comm);
// ...other work
VecDotEnd (...,&dot);
```

call to MPI_Wait

call to MPI_Iallreduce

Deep Pipelining

- Motivation: want to have perfect overlap of computation of inner products and SpMVs/preconditioner application
- But this depends on the machine, matrix, etc.
- If inner products take much longer than 1 SpMV, do ℓ SpMVs instead
  - $\Rightarrow$ "deep" pipelined method with pipeline length ℓ
  - ℓ should be chosen to be the number of SpMV/precond. operations that can be done in the time it takes for one Allreduce

- Deep pipelined GMRES variant [Ghysels, Ashby, Meerbergen, Vanroose, SIAM J. Sci. Comput, 35(1), 2013]
- Deep pipelined CG variant [Cornelis, Cools, Vanroose, arXiv:1801.04728, 2018]

Available Software

- Implementations in PETSc:
  - KSPPGMRES: pipelined GMRES
  - KSPPIPECG: pipelined CG
  - KSPPIPECR: pipelined CR
  - KSPGROPPCG: Gropp asynchronous variant
  - KSPPIPEBCGS: pipelined BiCGSTAB
  - KSPPIPELCG: deep pipelined CG

Performance of (Deep) Pipelined CG

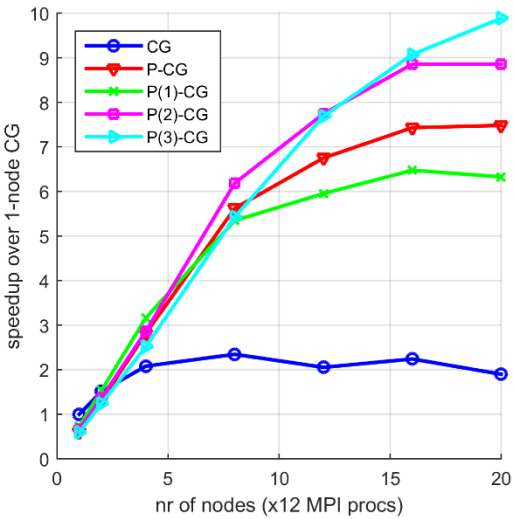


FIG. 5. *Strong scaling experiment on up to 20 nodes (240 processes) for a 5-point stencil 2D Poisson problem with 1.000.000 unknowns. Speedup over single-node classic CG for various pipeline lengths. All methods converged to* $\|r_i\|_2/\|b\|_2 = 1.0e\text{-}5$ *in 1342 iterations.*

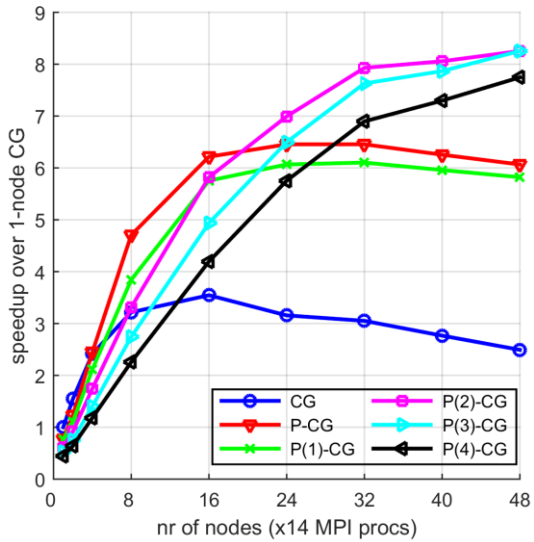


FIG. 6. *Strong scaling experiment on up to 48 nodes (672 processes) for a 5-point stencil 2D Poisson problem with 3.062.500 unknowns. Speedup over single-node classic CG for various pipeline lengths. All methods performed 1500 iterations with* $\|r_i\|_2/\|b\|_2 = 6.3e\text{-}4$.

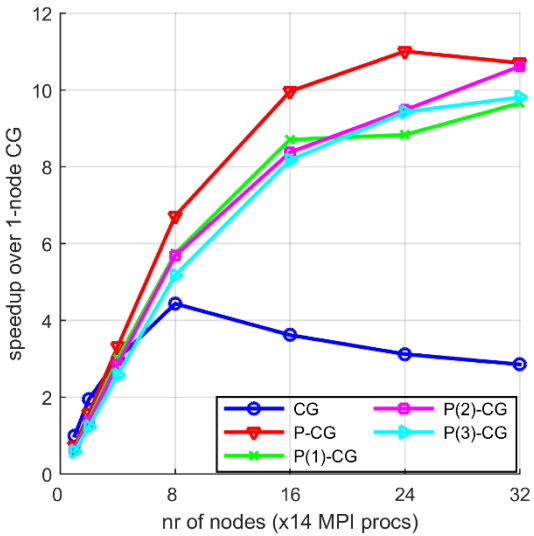


FIG. 7. *Strong scaling experiment on up to 32 nodes (448 processes) for a block Jacobi preconditioned 2D Poisson problem with 3.062.500 unknowns. All methods performed 600 iterations with* $\|r_i\|_2/\|b\|_2 = 1.8e\text{-}4$ *(on 1 node) and* $\|r_i\|_2/\|b\|_2 \leq 9.3e\text{-}4$ *(on 32 nodes).*

20 compute nodes, each with two 6-core Intel Xeon X5660 Nehalem 2:80 GHz processors each (12 cores per node); 4QDR InfiniBand

48 compute nodes, each with two 14-core Intel E5-2680v4, Broadwell generation CPUs; EDR InfiniBand

(Cornelis, Cools, Vanroose, arXiv: 1801.04728, 2018)

s-step Krylov subspace methods

- Idea: Compute blocks of s iterations at once
  - Compute updates in a different basis
  - Communicate every s iterations instead of every iteration
  - Reduces number of synchronizations per iteration by a factor of s

s-step Krylov subspace methods

- Idea: Compute blocks of s iterations at once
  - Compute updates in a different basis
  - Communicate every s iterations instead of every iteration
  - Reduces number of synchronizations per iteration by a factor of s

- An idea rediscovered many times...

s-step Krylov subspace methods

- Idea: Compute blocks of s iterations at once
  - Compute updates in a different basis
  - Communicate every s iterations instead of every iteration
  - Reduces number of synchronizations per iteration by a factor of s

- An idea rediscovered many times...
- First related work: s-dimensional steepest descent, least squares
  - Khabaza ('63), Forsythe ('68), Marchuk and Kuznecov ('68)

s-step Krylov subspace methods

- Idea: Compute blocks of s iterations at once
  - Compute updates in a different basis
  - Communicate every s iterations instead of every iteration
  - Reduces number of synchronizations per iteration by a factor of s

- An idea rediscovered many times...
- First related work: s-dimensional steepest descent, least squares
  - Khabaza ('63), Forsythe ('68), Marchuk and Kuznecov ('68)
- Flurry of work on s-step Krylov methods in '80s/early '90s: see, e.g., Van Rosendale (1983); Chronopoulos and Gear (1989)

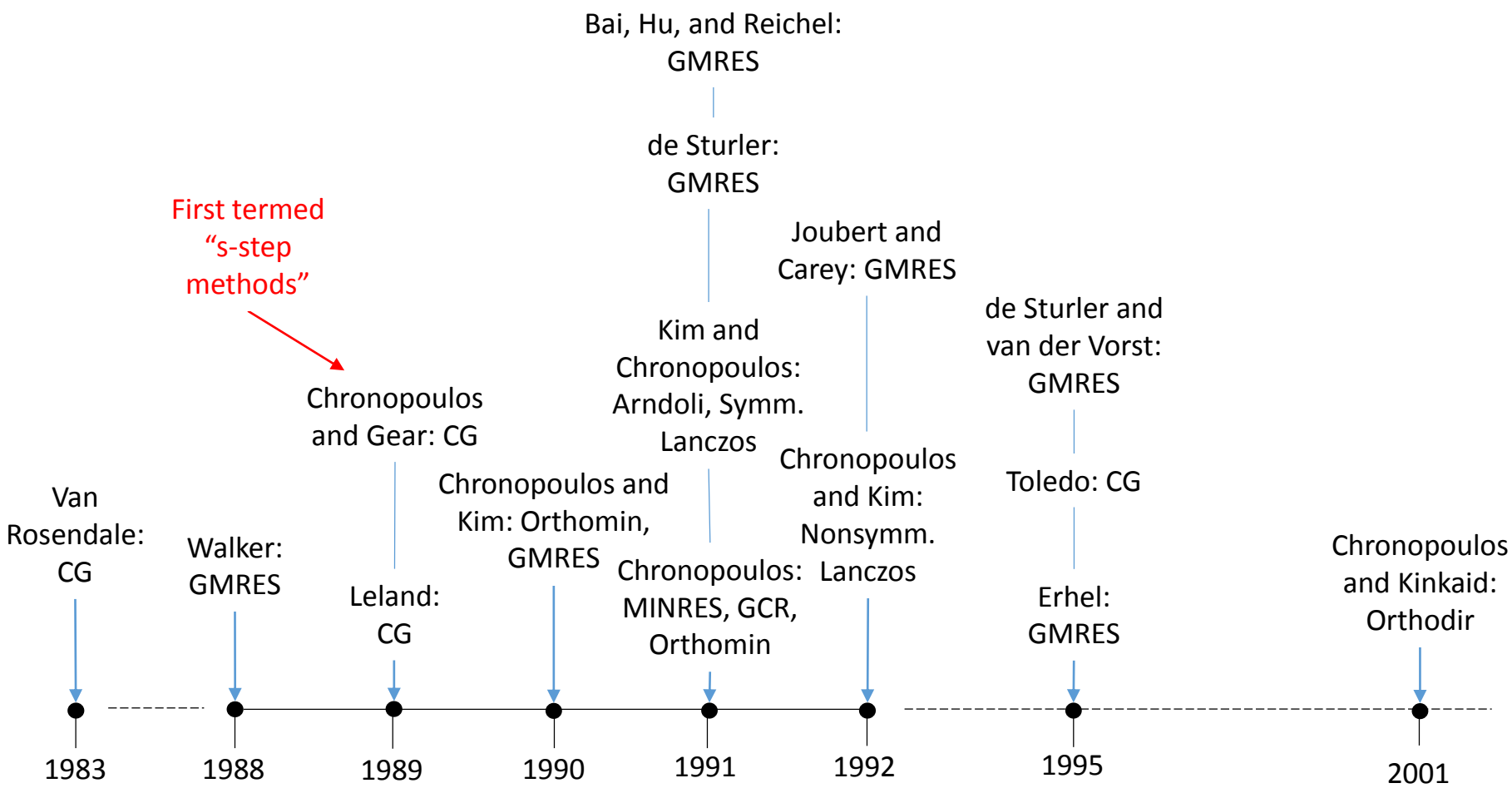
s-step Krylov subspace methods

- Idea: Compute blocks of s iterations at once
  - Compute updates in a different basis
  - Communicate every s iterations instead of every iteration
  - Reduces number of synchronizations per iteration by a factor of s

- An idea rediscovered many times...
- First related work: s-dimensional steepest descent, least squares
  - Khabaza ('63), Forsythe ('68), Marchuk and Kuznecov ('68)
- Flurry of work on s-step Krylov methods in '80s/early '90s: see, e.g., Van Rosendale (1983); Chronopoulos and Gear (1989)

- Resurgence of interest in recent years due to growing problem sizes; growing relative cost of communication

History of s-step Krylov Subspace Methods



s-step CG

Key observation: After iteration i, for $j \in \{0, .., s\}$,

$$x_{i+j} - x_i, \quad r_{i+j}, \quad p_{i+j} \quad \in \quad \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$$

s-step CG

Key observation: After iteration i, for $j \in \{0, .., s\}$,

$$x_{i+j} - x_i, \quad r_{i+j}, \quad p_{i+j} \quad \in \quad \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$$

s steps of s-step CG:

s-step CG

Key observation: After iteration i, for $j \in \{0, .., s\}$,

$$x_{i+j} - x_i, \quad r_{i+j}, \quad p_{i+j} \quad \in \quad \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$$

s steps of s-step CG:

Expand solution space s dimensions at once

Compute "basis" matrix $\mathcal{Y}$ such that $\operatorname{span}(\mathcal{Y}) = \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$ according to the recurrence $A\underline{\mathcal{Y}} = \mathcal{Y}\,\mathcal{B}$

s-step CG

Key observation: After iteration i, for $j \in \{0, .., s\}$,

$$x_{i+j} - x_i, \quad r_{i+j}, \quad p_{i+j} \quad \in \quad \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$$

s steps of s-step CG:

Expand solution space s dimensions at once

Compute "basis" matrix $\mathcal{Y}$ such that $\operatorname{span}(\mathcal{Y}) = \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$ according to the recurrence $A\underline{\mathcal{Y}} = \mathcal{Y}\,\mathcal{B}$

Compute inner products between basis vectors in one synchronization

$$\mathcal{G} = \mathcal{Y}^T \mathcal{Y}$$

s-step CG

Key observation: After iteration i, for $j \in \{0, .., s\}$,

$$x_{i+j} - x_i, \quad r_{i+j}, \quad p_{i+j} \quad \in \quad \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$$

s steps of s-step CG:

Expand solution space s dimensions at once

Compute "basis" matrix $\mathcal{Y}$ such that $\operatorname{span}(\mathcal{Y}) = \mathcal{K}_{s+1}(A, p_i) + \mathcal{K}_s(A, r_i)$ according to the recurrence $A\underline{\mathcal{Y}} = \mathcal{Y}\,\mathcal{B}$

Compute inner products between basis vectors in one synchronization

$$\mathcal{G} = \mathcal{Y}^T \mathcal{Y}$$

Compute s iterations of vector updates

Perform s iterations of vector updates by updating coordinates in basis $\mathcal{Y}$:

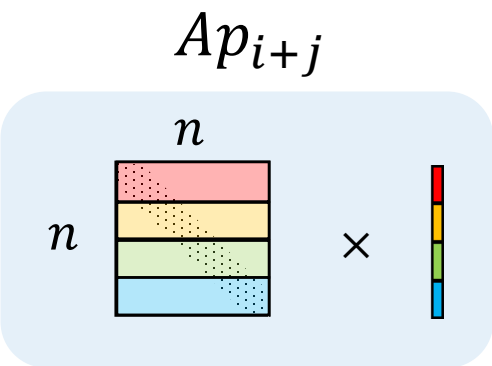
$$x_{i+j} - x_i = \mathcal{Y} x'_j, \qquad r_{i+j} = \mathcal{Y} r'_j, \qquad p_{i+j} = \mathcal{Y} p'_j$$

s-step CG

For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:

s-step CG

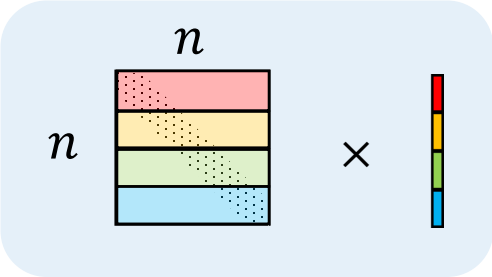
For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:



s-step CG

For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:

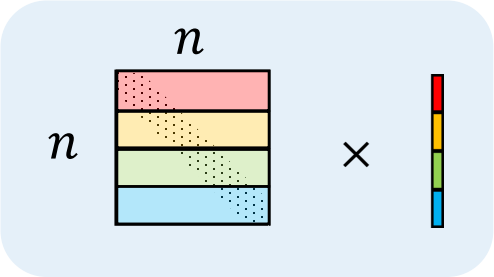
$$Ap_{i+j} \quad = \quad A\underline{\mathcal{Y}}p'_j$$



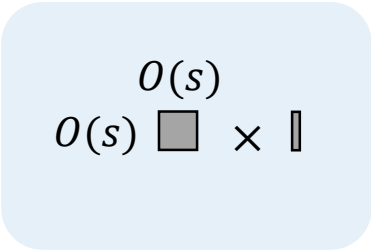
s-step CG

For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:

$$Ap_{i+j} \quad = \quad A\underline{\mathcal{Y}}p'_j \quad = \quad \mathcal{Y}(\mathcal{B}p'_j)$$



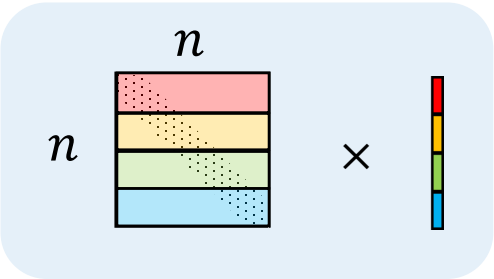
$\rightarrow$



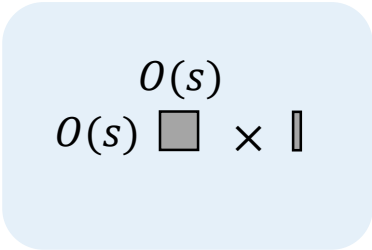
s-step CG

For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:

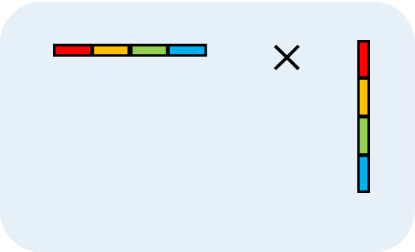
$$Ap_{i+j} \quad = \quad A\underline{\mathcal{Y}}p'_j \quad = \quad \mathcal{Y}(\mathcal{B}p'_j)$$



$\rightarrow$



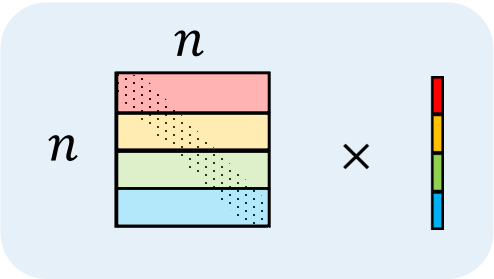
$$(r_{i+j}, r_{i+j})$$



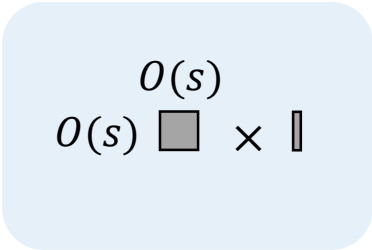
s-step CG

For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:

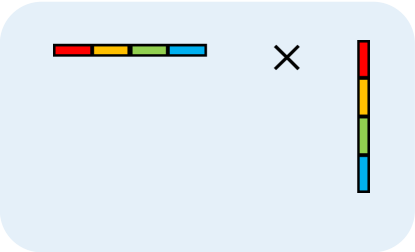
$$Ap_{i+j} = A\underline{\mathcal{Y}}p'_j = \mathcal{Y}(\mathcal{B}p'_j)$$



$\rightarrow$



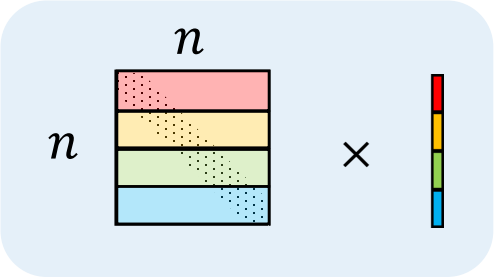
$$(r_{i+j}, r_{i+j}) = r_j'^T \mathcal{Y}^T \mathcal{Y} r'_j$$



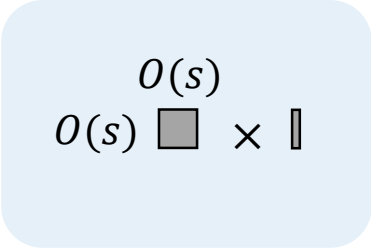
s-step CG

For s iterations of updates, inner products and SpMVs (in basis $\mathcal{Y}$) can be computed by independently by each processor without communication:

$$Ap_{i+j} = A\underline{\mathcal{Y}}p'_j = \mathcal{Y}(\mathcal{B}p'_j)$$

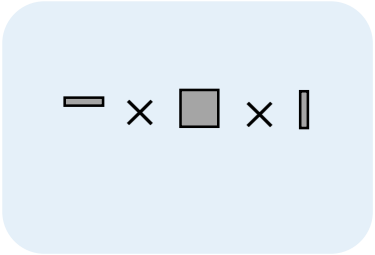


$\rightarrow$



$$(r_{i+j}, r_{i+j}) = r_j'^T \mathcal{Y}^T \mathcal{Y} r'_j = r_j'^T \mathcal{G} r'_j$$

$\rightarrow$



s-step CG

$r_0 = b - Ax_0, p_0 = r_0$

for $k = 0$:nmax$/s$

Compute $\mathcal{Y}_k$ and $\mathcal{B}_k$ such that $A\underline{\mathcal{Y}_k} = \mathcal{Y}_k\mathcal{B}_k$ and

$$\text{span}(\mathcal{Y}_k) = \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk})$$

$\mathcal{G}_k = \mathcal{Y}_k^T\mathcal{Y}_k$

$x_0' = 0, r_0' = e_{s+2}, p_0' = e_1$

for $j = 1{:}\,s$

$$\alpha_{sk+j-1} = \frac{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}{p_{j-1}'^T\mathcal{G}_k\mathcal{B}_k p_{j-1}'}$$

$$x_j' = x_{j-1}' + \alpha_{sk+j-1}p_{j-1}'$$

$$r_j' = r_{j-1}' - \alpha_{sk+j-1}\mathcal{B}_k p_{j-1}'$$

$$\beta_{sk+j} = \frac{r_j'^T\mathcal{G}_k r_j'}{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}$$

$$p_j' = r_j' + \beta_{sk+j}p_{j-1}'$$

end

$[x_{s(k+1)} - x_{sk}, r_{s(k+1)}, p_{s(k+1)}] = \mathcal{Y}_k[x_s', r_s', p_s']$

end

s-step CG

$r_0 = b - Ax_0, p_0 = r_0$

for $k = 0{:}\text{nmax}/s$

Compute $\mathcal{Y}_k$ and $\mathcal{B}_k$ such that $A\underline{\mathcal{Y}_k} = \mathcal{Y}_k\mathcal{B}_k$ and

$$\text{span}(\mathcal{Y}_k) = \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk})$$

$\mathcal{G}_k = \mathcal{Y}_k^T\mathcal{Y}_k$

$x_0' = 0, r_0' = e_{s+2}, p_0' = e_1$

for $j = 1{:}\,s$

$$\alpha_{sk+j-1} = \frac{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}{p_{j-1}'^T\mathcal{G}_k\mathcal{B}_k p_{j-1}'}$$

$$x_j' = x_{j-1}' + \alpha_{sk+j-1}p_{j-1}'$$

$$r_j' = r_{j-1}' - \alpha_{sk+j-1}\mathcal{B}_k p_{j-1}'$$

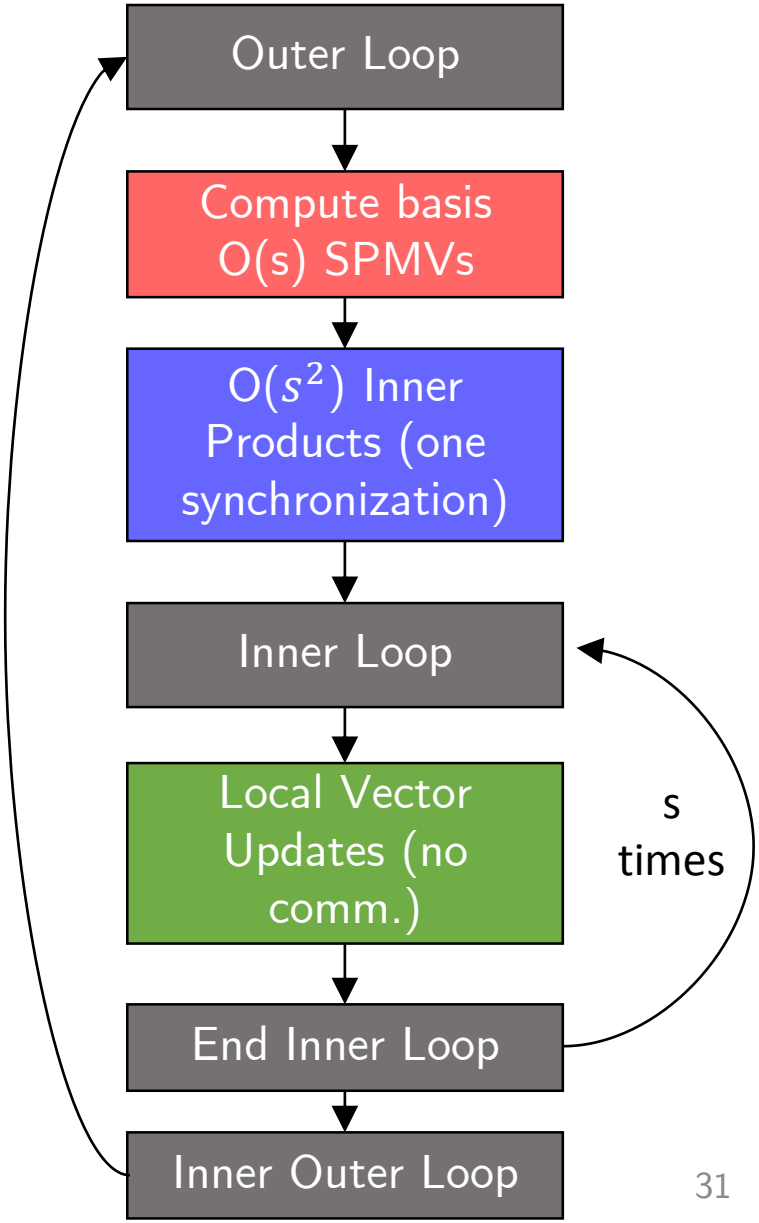
$$\beta_{sk+j} = \frac{r_j'^T\mathcal{G}_k r_j'}{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}$$

$$p_j' = r_j' + \beta_{sk+j}p_{j-1}'$$

end

$[x_{s(k+1)} - x_{sk}, r_{s(k+1)}, p_{s(k+1)}] = \mathcal{Y}_k[x_s', r_s', p_s']$

end



s-step CG

$r_0 = b - Ax_0, p_0 = r_0$

for $k = 0{:}\text{nmax}/s$

Compute $\mathcal{Y}_k$ and $\mathcal{B}_k$ such that $A\underline{\mathcal{Y}_k} = \mathcal{Y}_k\mathcal{B}_k$ and

$\text{span}(\mathcal{Y}_k) = \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk})$

$\mathcal{G}_k = \mathcal{Y}_k^T\mathcal{Y}_k$

$x_0' = 0, r_0' = e_{s+2}, p_0' = e_1$

for $j = 1{:}\,s$

$$\alpha_{sk+j-1} = \frac{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}{p_{j-1}'^T\mathcal{G}_k\mathcal{B}_k p_{j-1}'}$$

$$x_j' = x_{j-1}' + \alpha_{sk+j-1}p_{j-1}'$$

$$r_j' = r_{j-1}' - \alpha_{sk+j-1}\mathcal{B}_k p_{j-1}'$$

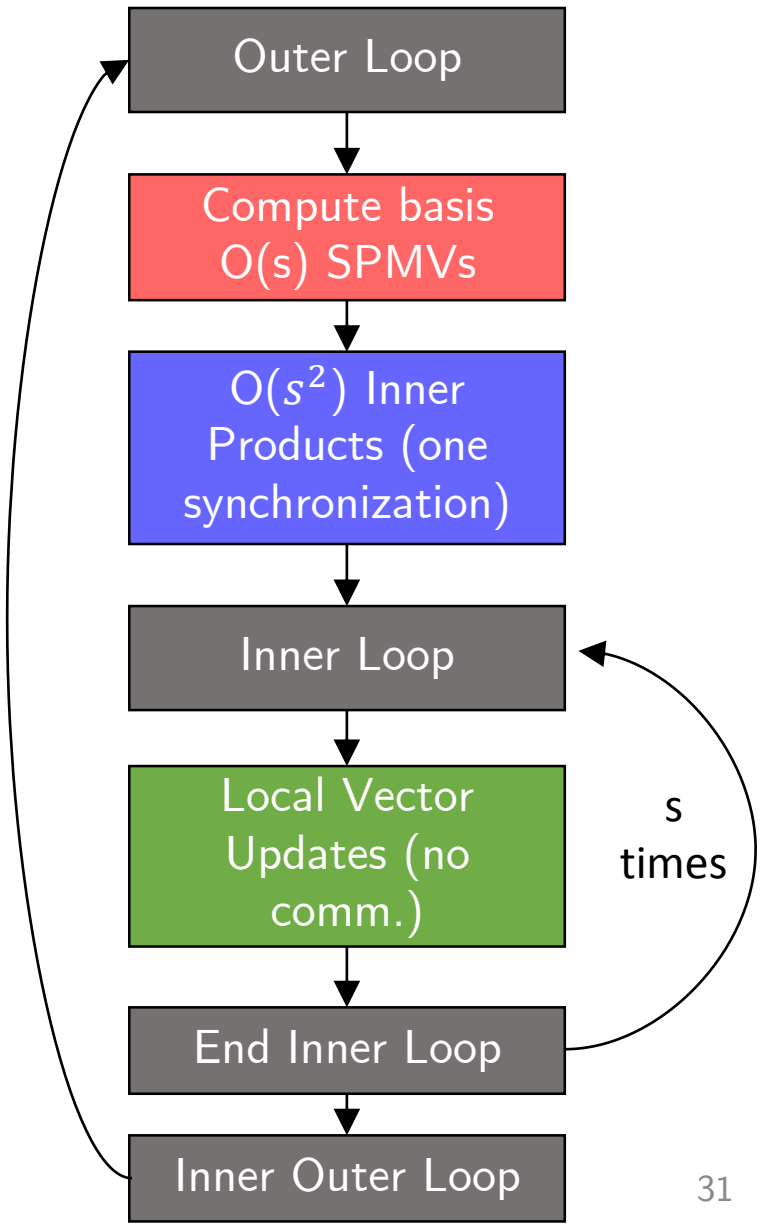
$$\beta_{sk+j} = \frac{r_j'^T\mathcal{G}_k r_j'}{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}$$

$$p_j' = r_j' + \beta_{sk+j}p_{j-1}'$$

end

$[x_{s(k+1)} - x_{sk}, r_{s(k+1)}, p_{s(k+1)}] = \mathcal{Y}_k[x_s', r_s', p_s']$

end



s-step CG

$r_0 = b - Ax_0, p_0 = r_0$

for $k = 0{:}\text{nmax}/s$

Compute $\mathcal{Y}_k$ and $\mathcal{B}_k$ such that $A\underline{\mathcal{Y}_k} = \mathcal{Y}_k\mathcal{B}_k$ and

$\text{span}(\mathcal{Y}_k) = \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk})$

$\mathcal{G}_k = \mathcal{Y}_k^T\mathcal{Y}_k$

$x_0' = 0, r_0' = e_{s+2}, p_0' = e_1$

for $j = 1{:}\,s$

$$\alpha_{sk+j-1} = \frac{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}{p_{j-1}'^T\mathcal{G}_k\mathcal{B}_k p_{j-1}'}$$

$$x_j' = x_{j-1}' + \alpha_{sk+j-1}p_{j-1}'$$

$$r_j' = r_{j-1}' - \alpha_{sk+j-1}\mathcal{B}_k p_{j-1}'$$

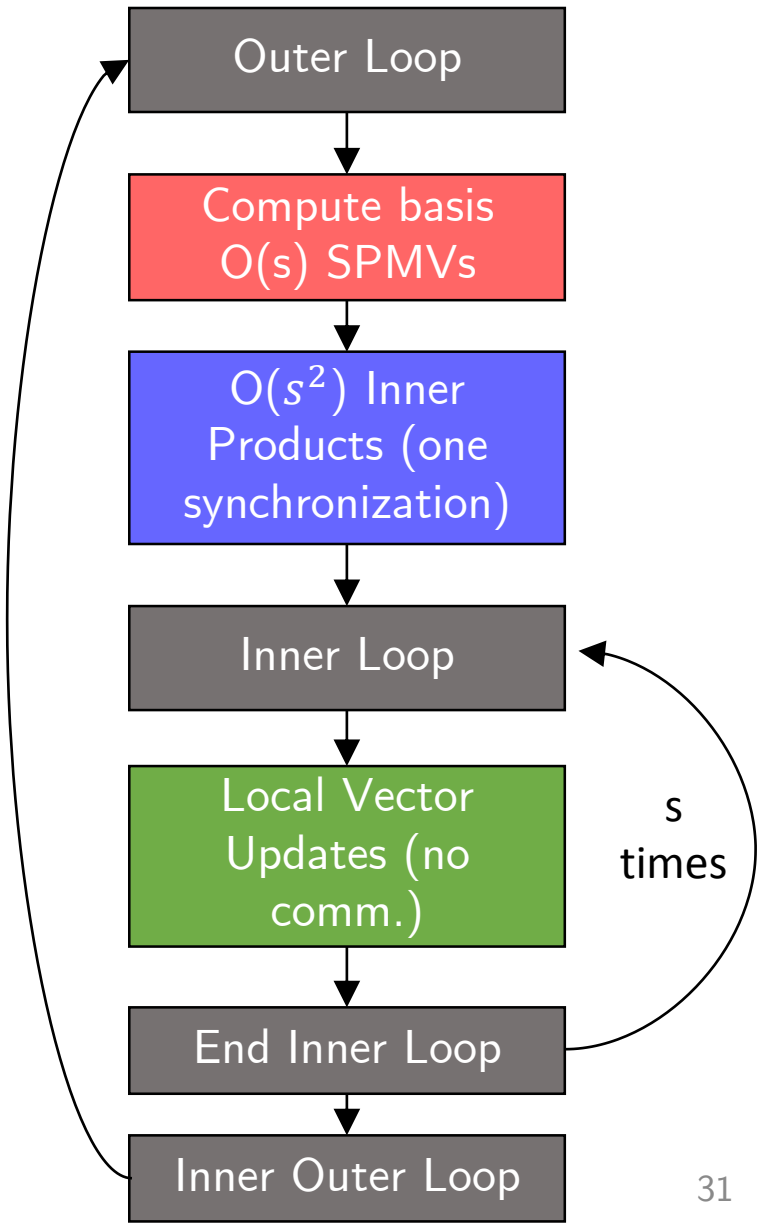
$$\beta_{sk+j} = \frac{r_j'^T\mathcal{G}_k r_j'}{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}$$

$$p_j' = r_j' + \beta_{sk+j}p_{j-1}'$$

end

$[x_{s(k+1)} - x_{sk}, r_{s(k+1)}, p_{s(k+1)}] = \mathcal{Y}_k[x_s', r_s', p_s']$

end



s-step CG

$r_0 = b - Ax_0, p_0 = r_0$

for $k = 0{:}\text{nmax}/s$

Compute $\mathcal{Y}_k$ and $\mathcal{B}_k$ such that $A\underline{\mathcal{Y}_k} = \mathcal{Y}_k\mathcal{B}_k$ and

$\text{span}(\mathcal{Y}_k) = \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk})$

$\mathcal{G}_k = \mathcal{Y}_k^T\mathcal{Y}_k$

$x_0' = 0, r_0' = e_{s+2}, p_0' = e_1$

for $j = 1{:}\,s$

$$\alpha_{sk+j-1} = \frac{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}{p_{j-1}'^T\mathcal{G}_k\mathcal{B}_k p_{j-1}'}$$

$$x_j' = x_{j-1}' + \alpha_{sk+j-1}p_{j-1}'$$

$$r_j' = r_{j-1}' - \alpha_{sk+j-1}\mathcal{B}_k p_{j-1}'$$

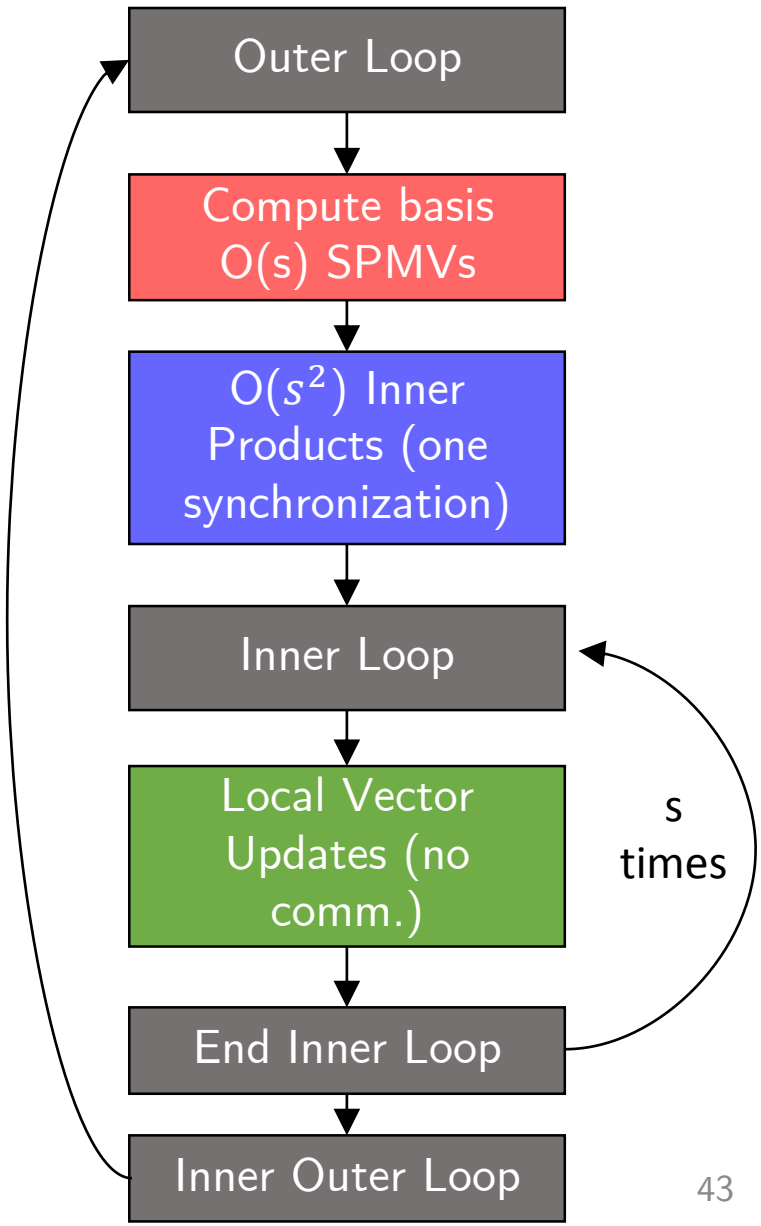
$$\beta_{sk+j} = \frac{r_j'^T\mathcal{G}_k r_j'}{r_{j-1}'^T\mathcal{G}_k r_{j-1}'}$$

$$p_j' = r_j' + \beta_{sk+j}p_{j-1}'$$

end

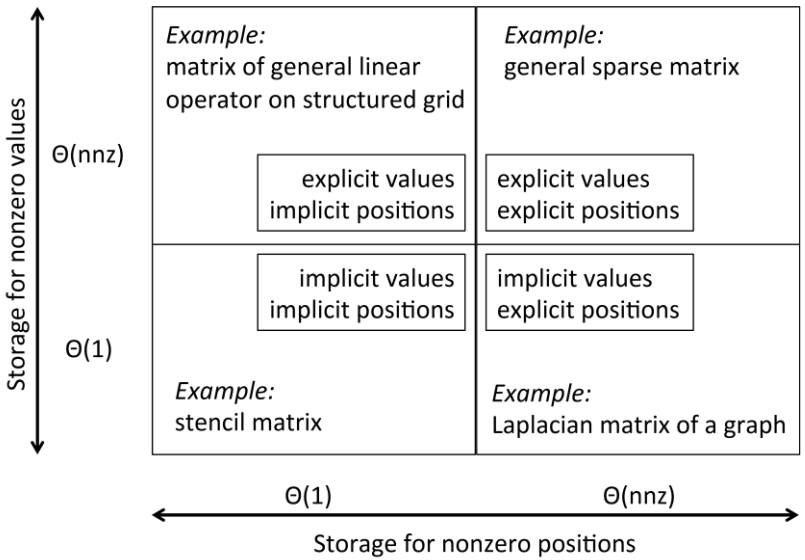
$[x_{s(k+1)} - x_{sk}, r_{s(k+1)}, p_{s(k+1)}] = \mathcal{Y}_k[x_s', r_s', p_s']$

end



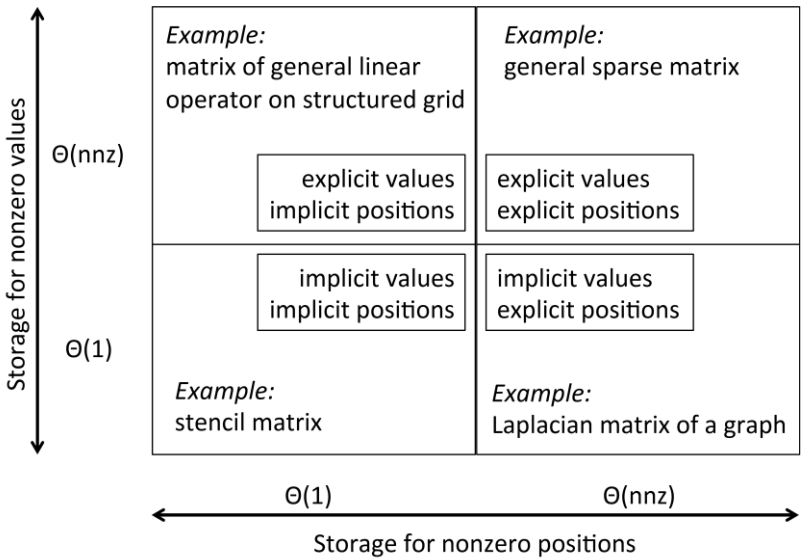
Sparse Matrix Computations

- Sparse Matrix x Vector (SpMV) ($y = Ax$)
  - Very communication-bound; no reuse
  - Lower bound depends on sparsity structure, algorithm used (1D rowwise/colwise, 2D, etc.)
  - Communication cost depends on partition
  - Hypergraph models capture communication dependencies (Catalyurek, Aykanat, 1999)
    - minimize hypergraph cut = minimize words moved



Sparse Matrix Computations

- Sparse Matrix x Vector (SpMV) ($y = Ax$)
  - Very communication-bound; no reuse
  - Lower bound depends on sparsity structure, algorithm used (1D rowwise/colwise, 2D, etc.)
  - Communication cost depends on partition
  - Hypergraph models capture communication dependencies (Catalyurek, Aykanat, 1999)
    - minimize hypergraph cut = minimize words moved



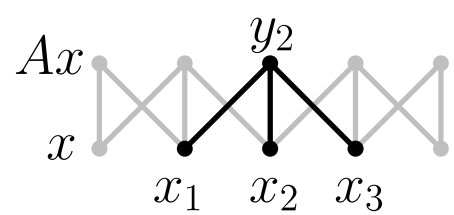
- Repeated SpMVs ($Y = [Ax, A^2x, \ldots, A^kx]$)
  - Naive approach: k repeated SpMVs
  - Communication-avoiding approach: "matrix powers kernel"
    - see, e.g., (Demmel, Hoemmen, Mohiyuddin, Yelick, 2008)

SpMV Dependency Graph

$G = (V, E)$ where $V = \{y_0, \dots, y_{n-1}\} \cup \{x_0, \dots, x_{n-1}\}$ and $(y_i, x_j) \in E$ if $A_{ij} \neq 0$

Example: Tridiagonal matrix

$$\underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} \Rightarrow$$

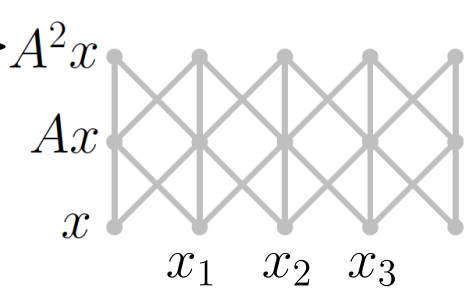


SpMV Dependency Graph

$G = (V, E)$ where $V = \{y_0, \dots, y_{n-1}\} \cup \{x_0, \dots, x_{n-1}\}$ and $(y_i, x_j) \in E$ if $A_{ij} \neq 0$

Example: Tridiagonal matrix

$$\underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} \Rightarrow$$

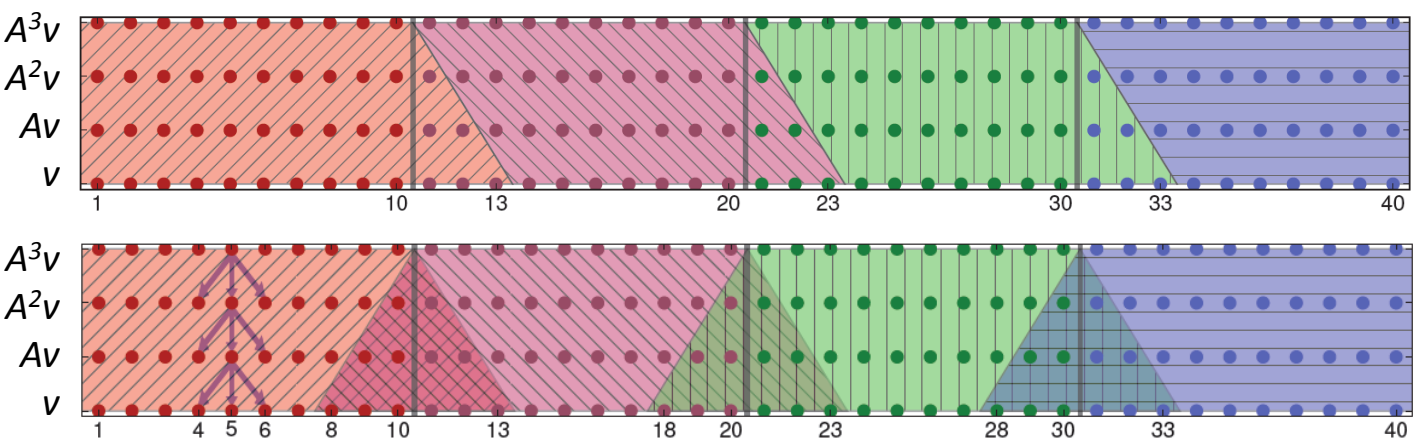


The Matrix Powers Kernel (Demmel et al., 2007)

Avoids communication:

- In serial, by exploiting temporal locality:
  - Reading A, reading vectors
- In parallel, by doing only 1 'expand' phase (instead of s).
- Requires sufficiently low 'surface-to-volume' ratio

Tridiagonal Example:



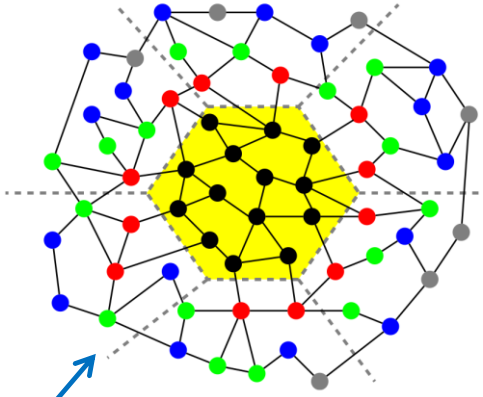
Sequential

Parallel

The Matrix Powers Kernel (Demmel et al., 2007)

Avoids communication:

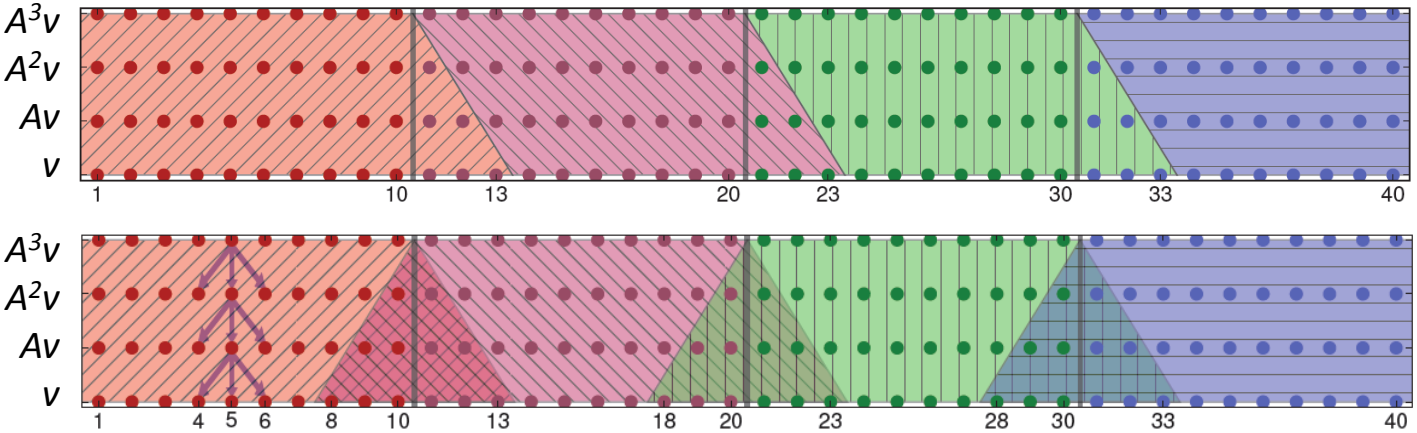
- In serial, by exploiting temporal locality:
  - Reading A, reading vectors
- In parallel, by doing only 1 'expand' phase (instead of s).
- Requires sufficiently low 'surface-to-volume' ratio



Also works for general graphs!

black = local elements
red = 1-level dependencies
green = 2-level dependencies
blue = 3-level dependencies

Tridiagonal Example:

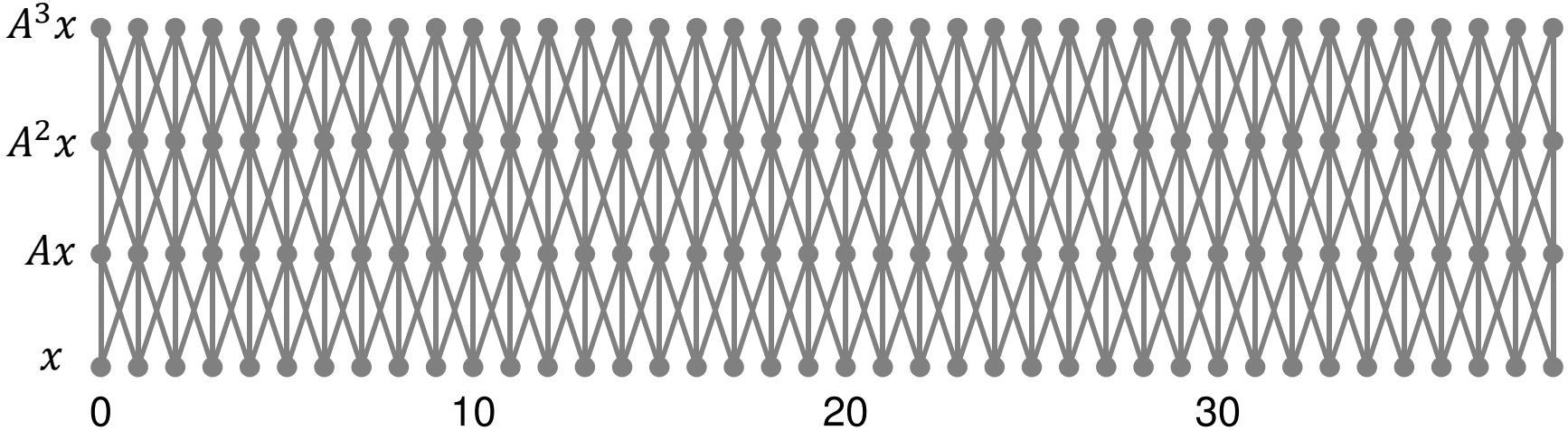


Sequential

Parallel

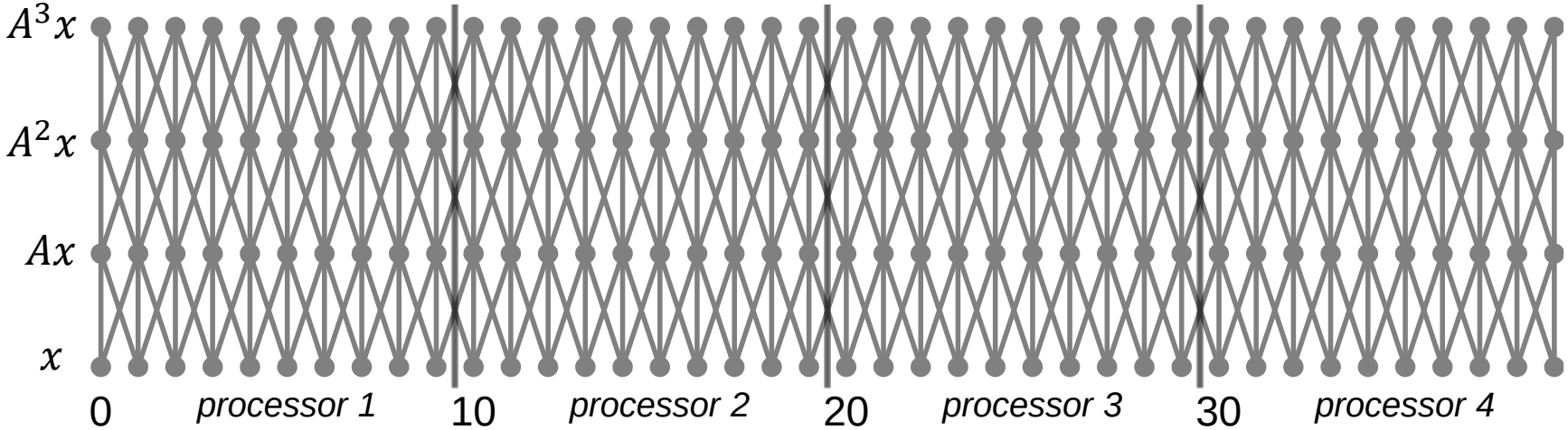
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



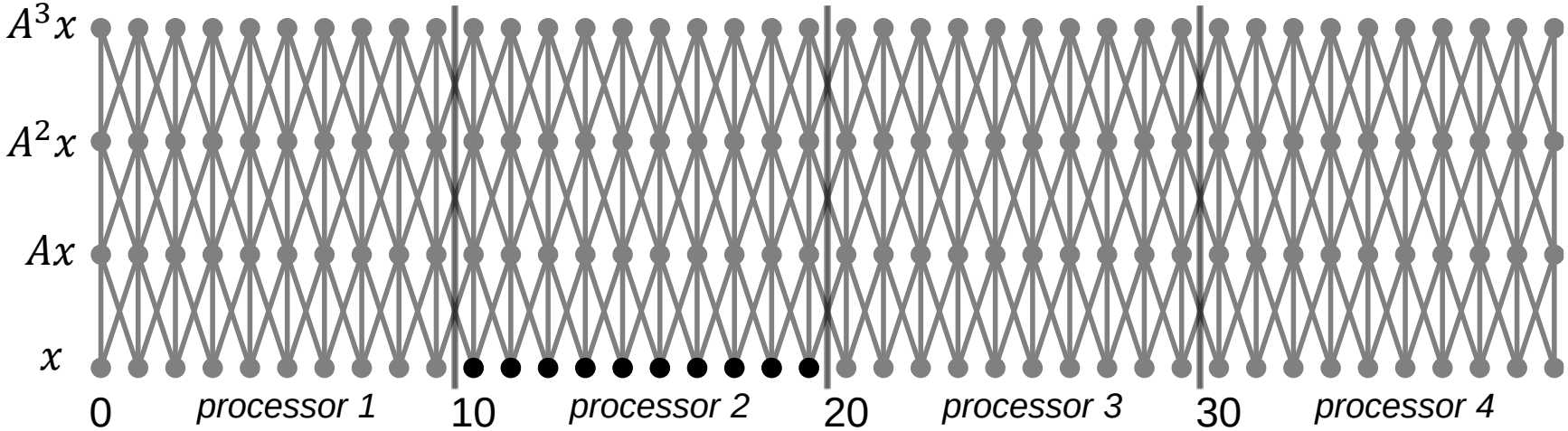
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



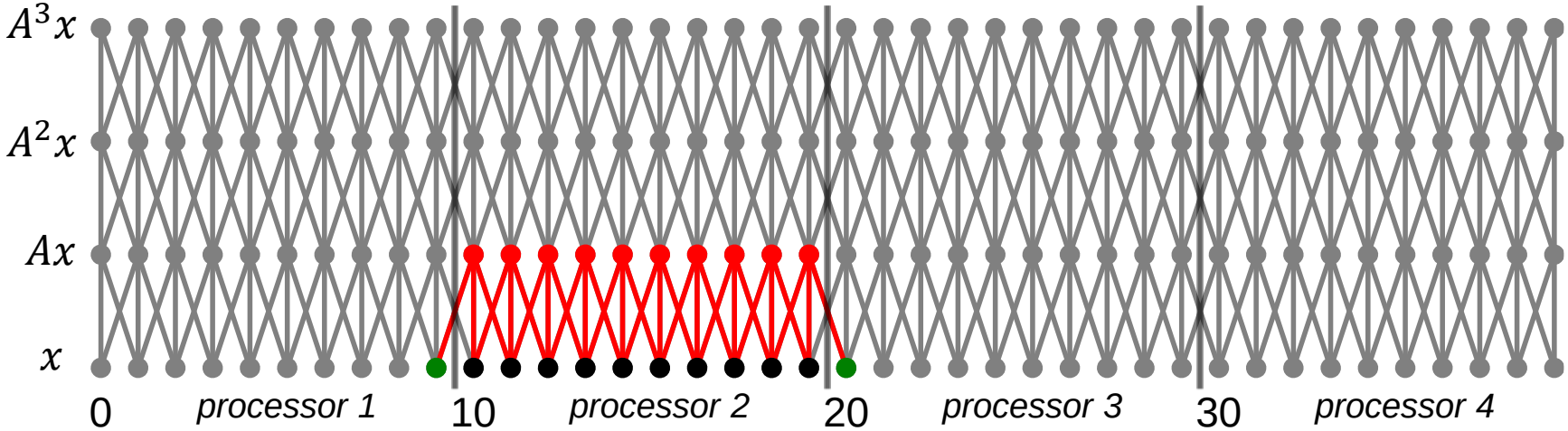
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



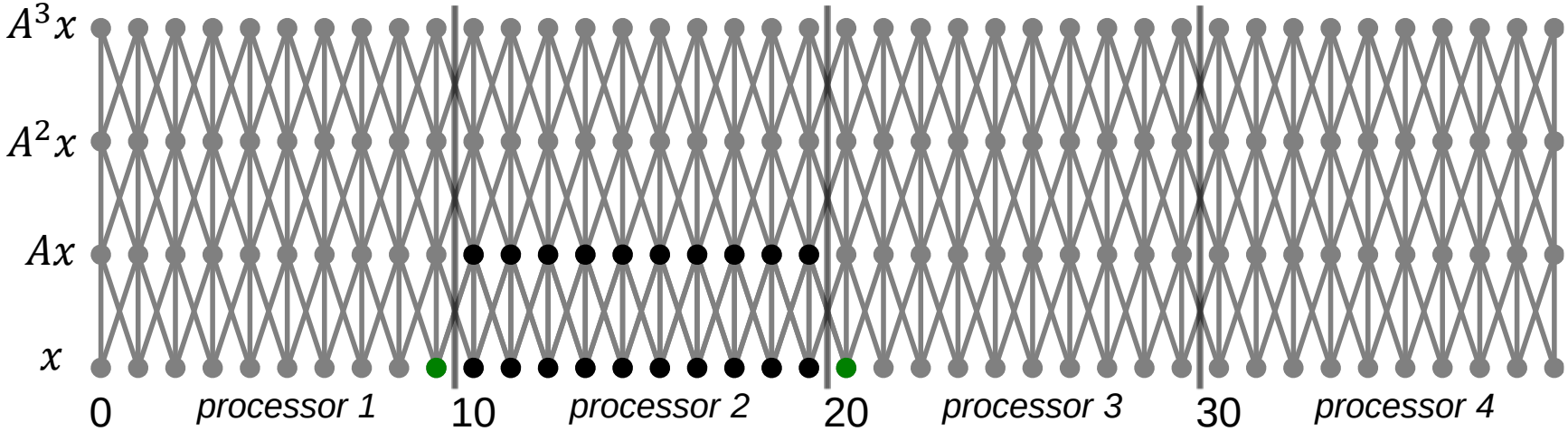
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



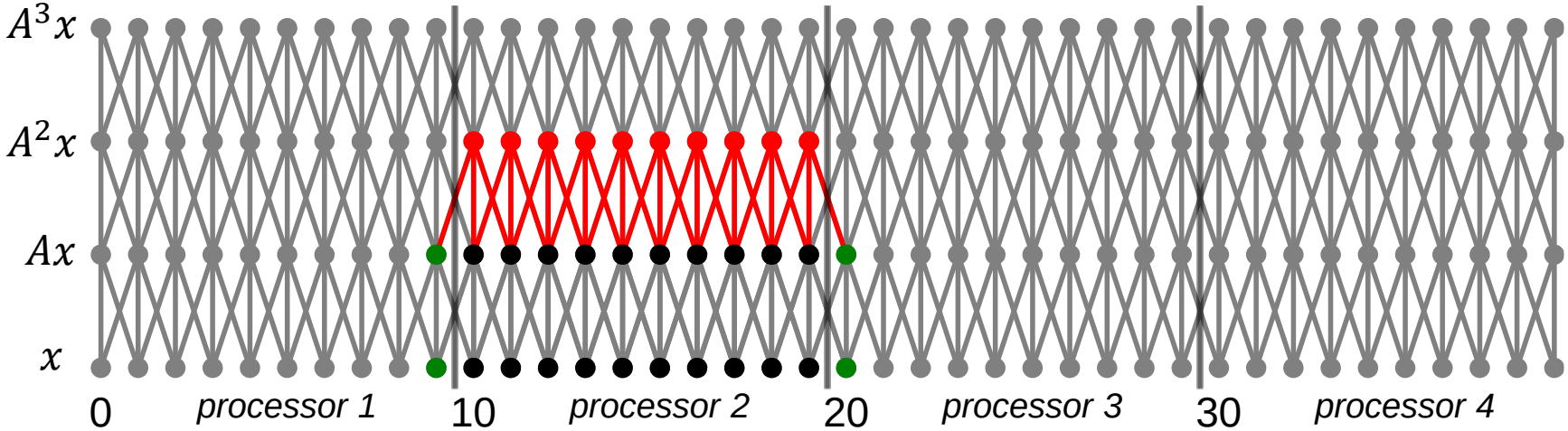
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



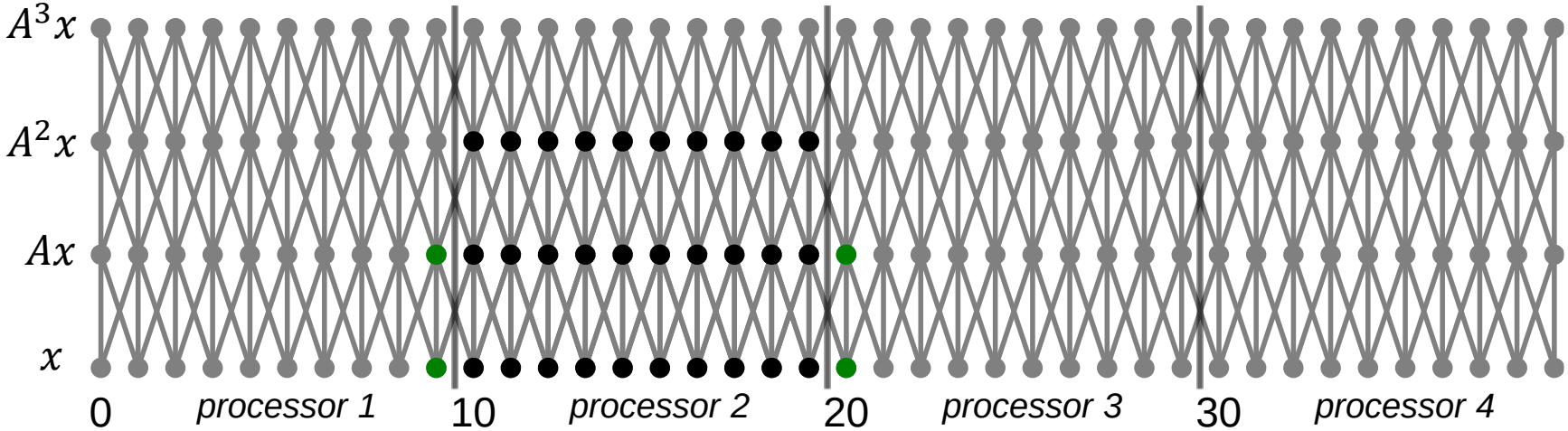
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



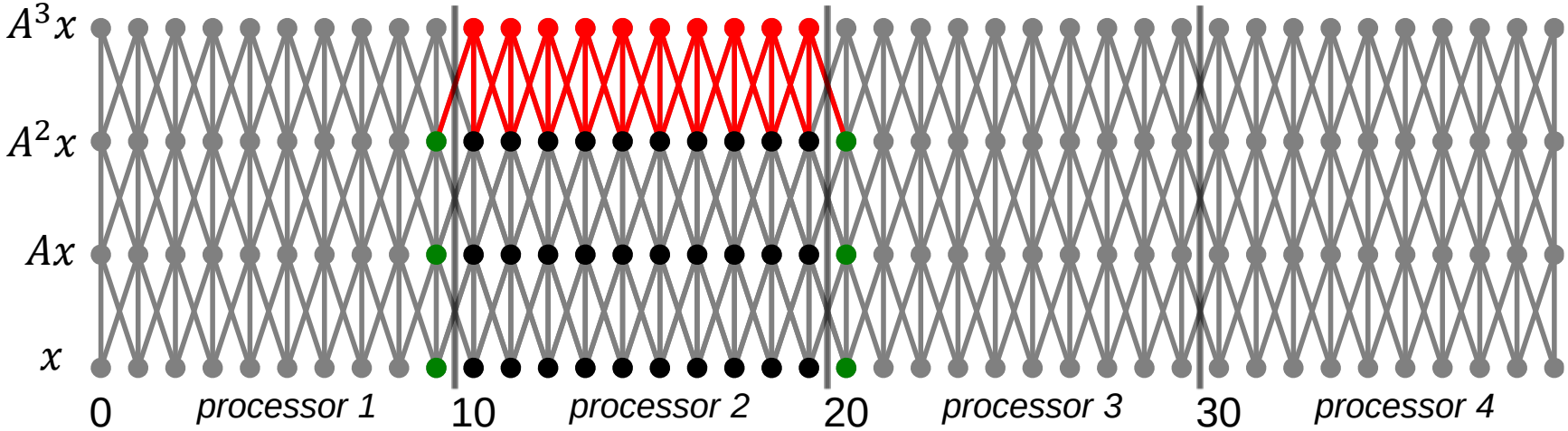
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



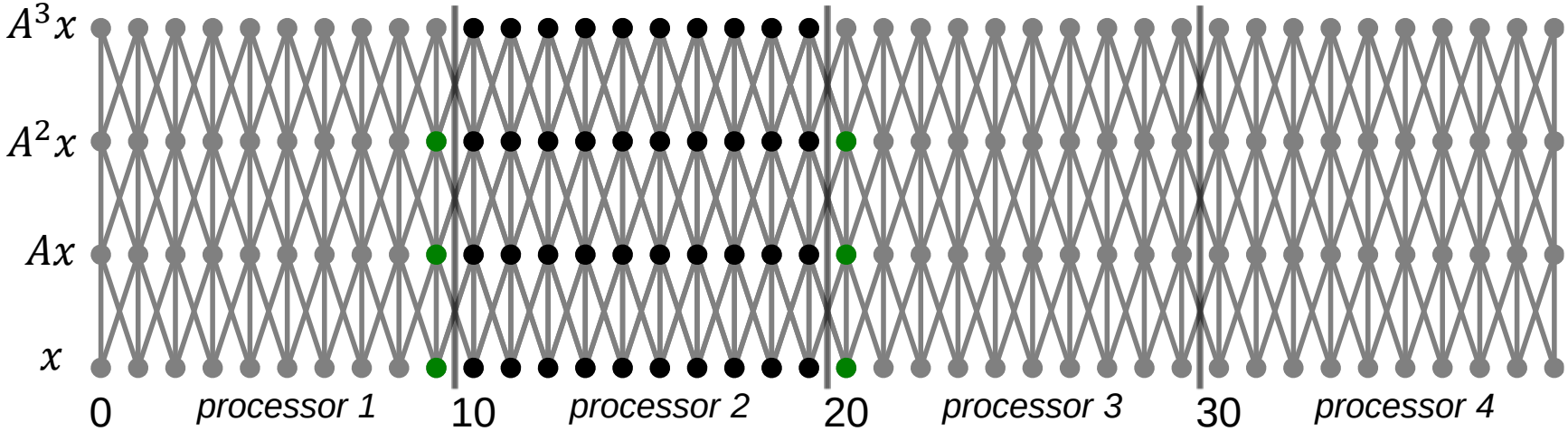
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



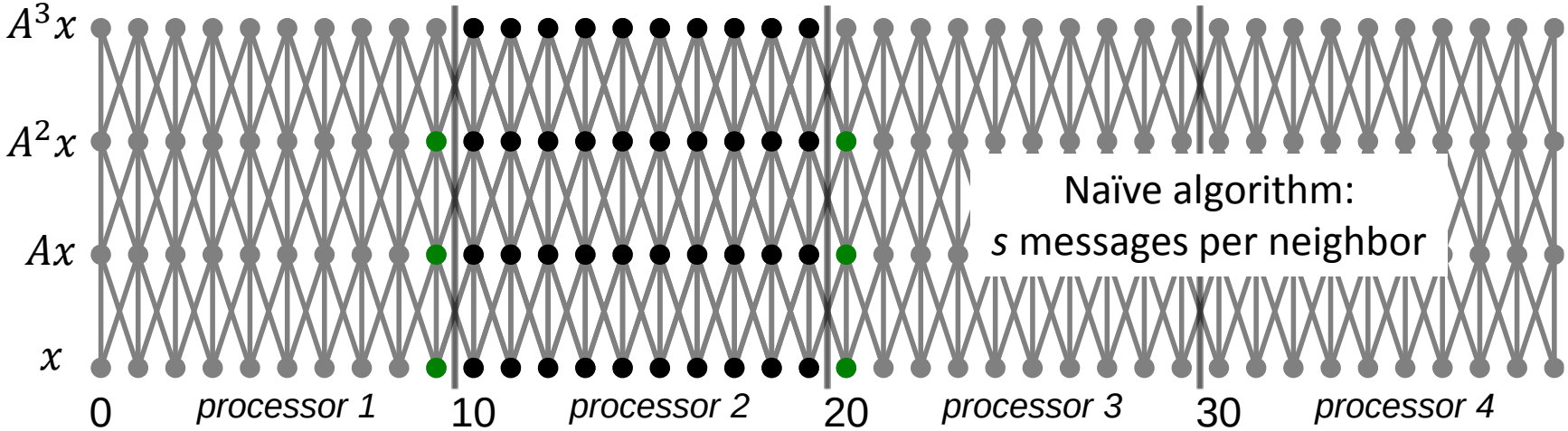
Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4



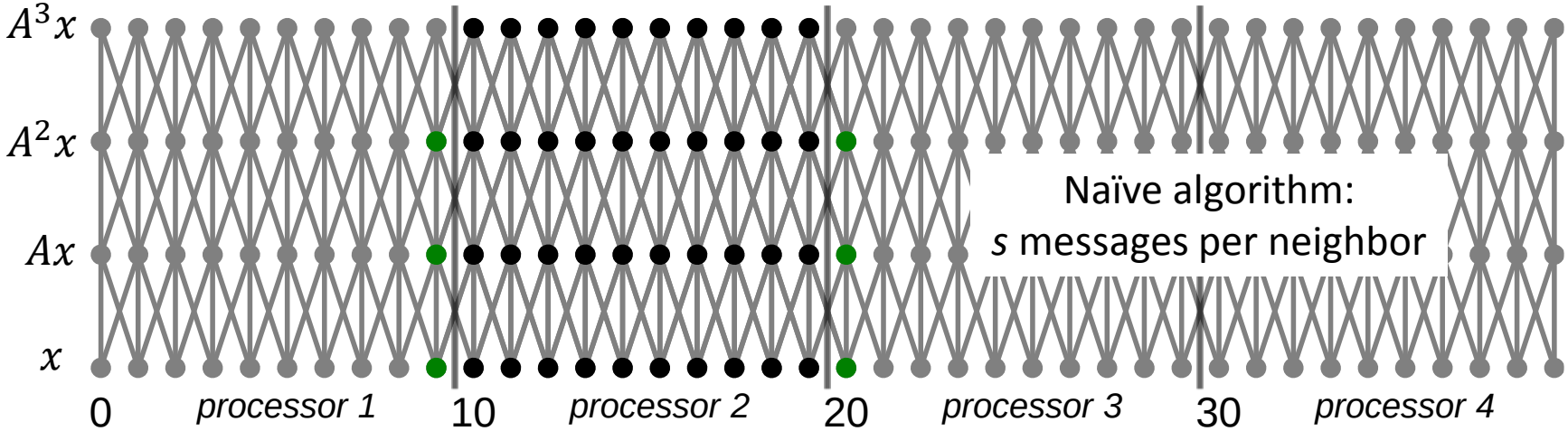
Parallel Matrix Powers Kernel

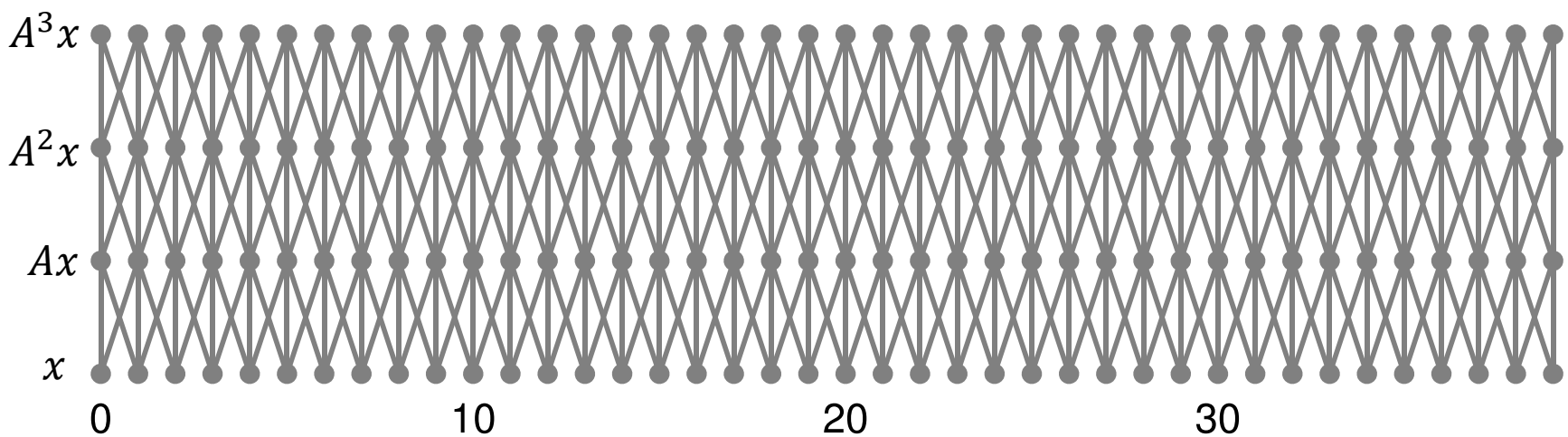
Example: tridiagonal matrix, s = 3, n = 40, p = 4



Parallel Matrix Powers Kernel

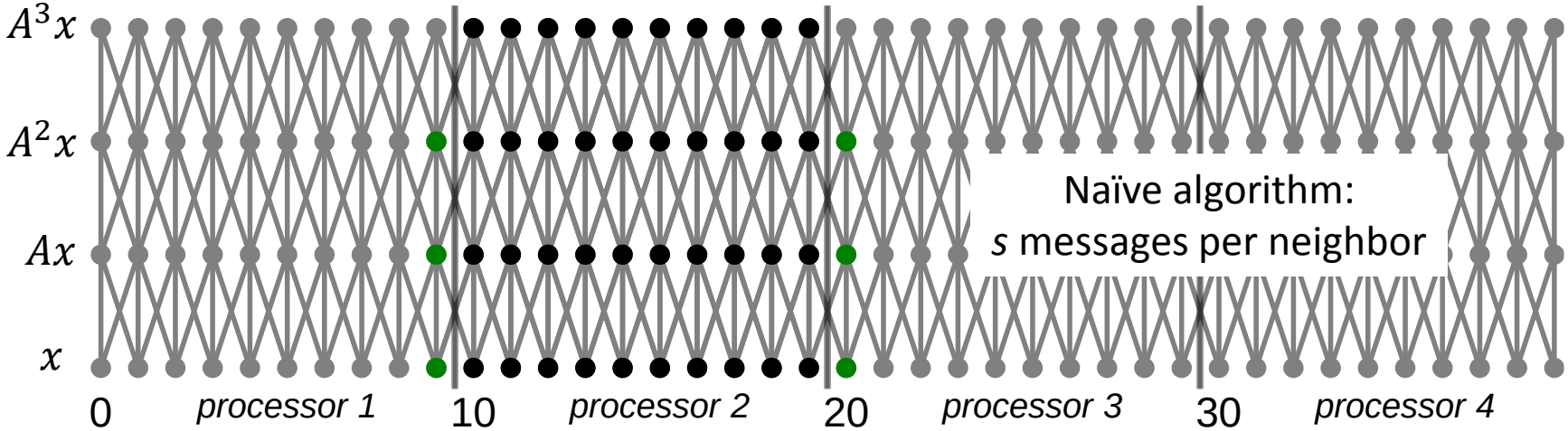
Example: tridiagonal matrix, s = 3, n = 40, p = 4

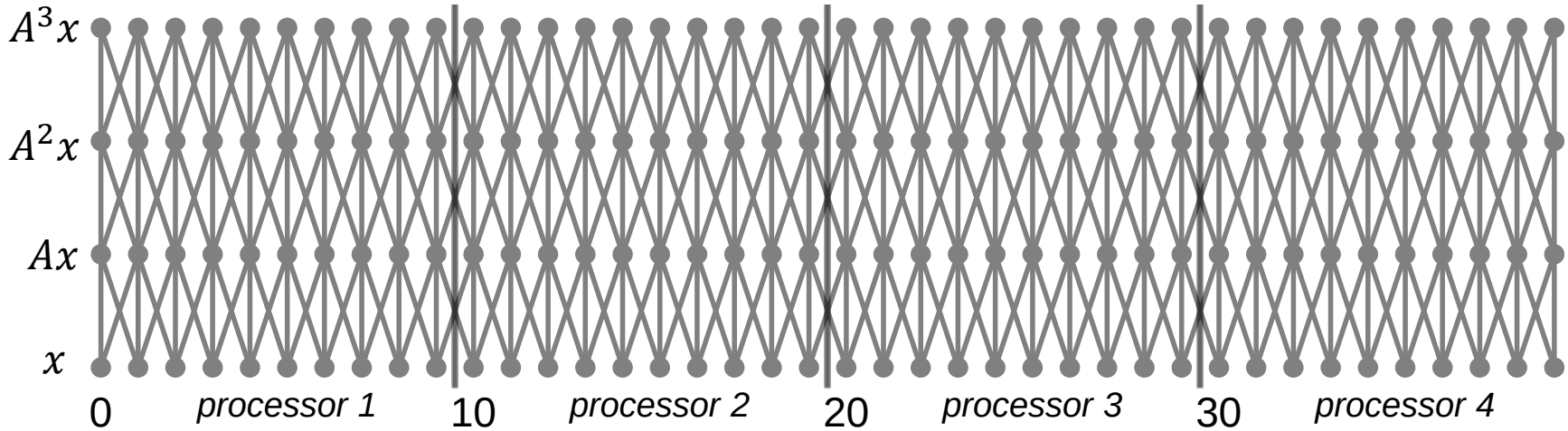




Parallel Matrix Powers Kernel

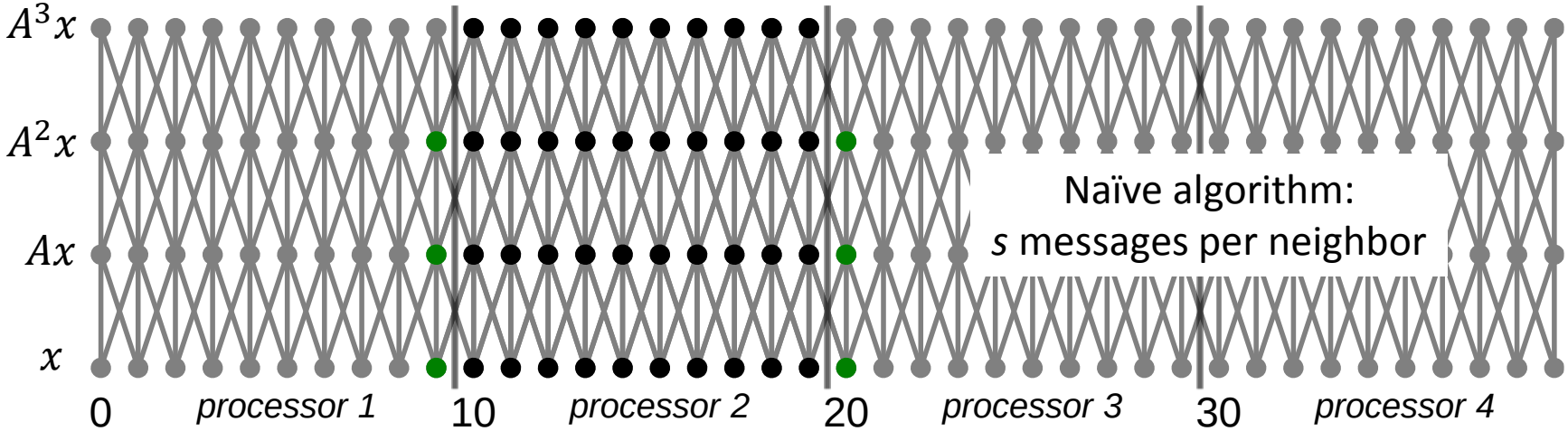
Example: tridiagonal matrix, s = 3, n = 40, p = 4

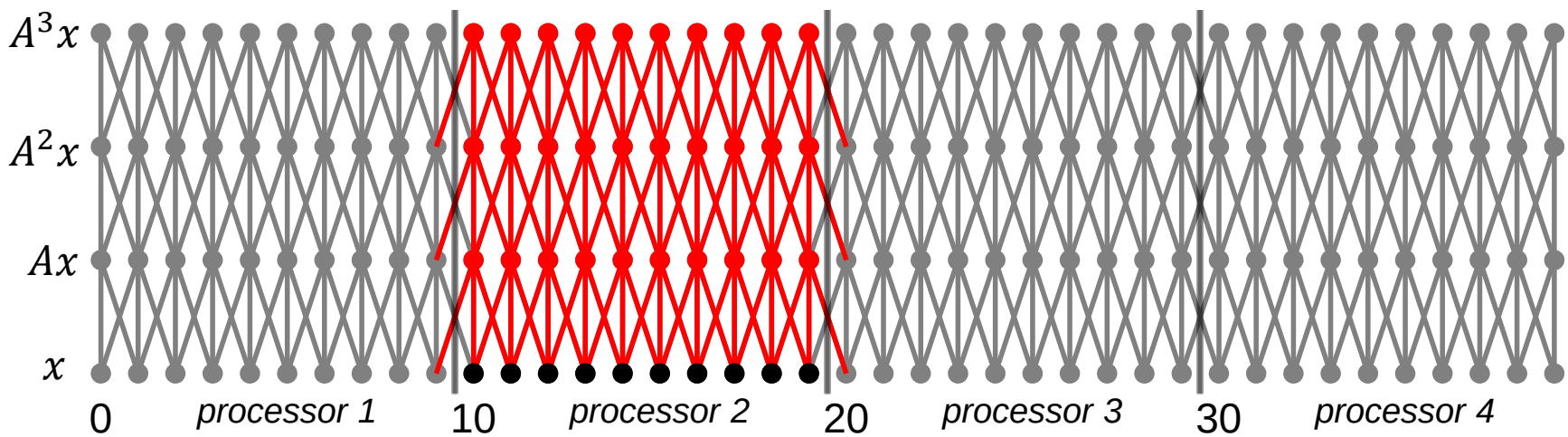




Parallel Matrix Powers Kernel

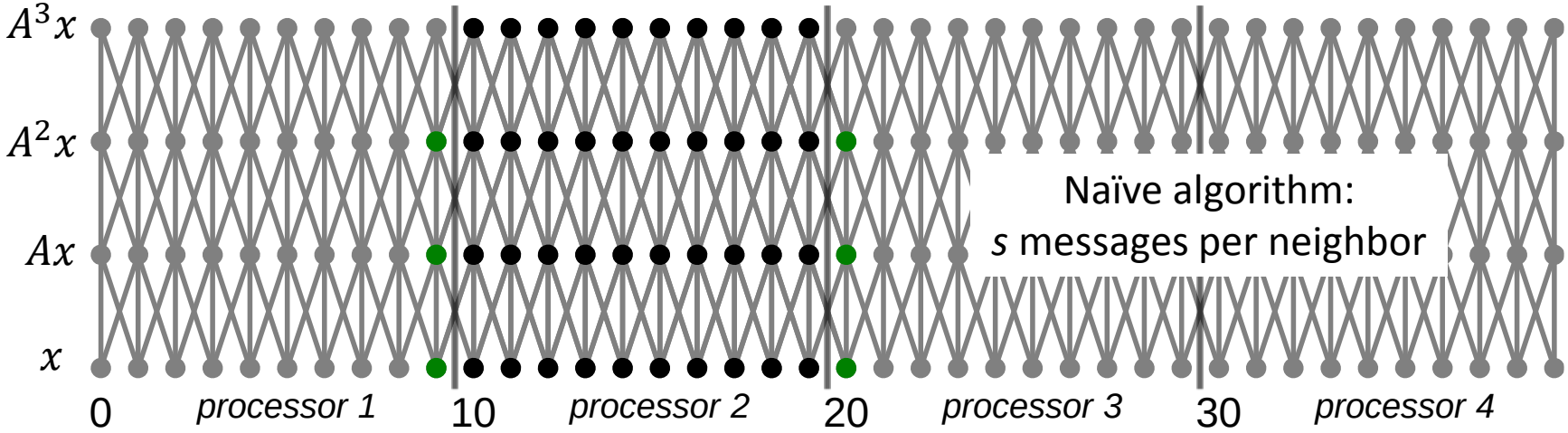
Example: tridiagonal matrix, s = 3, n = 40, p = 4

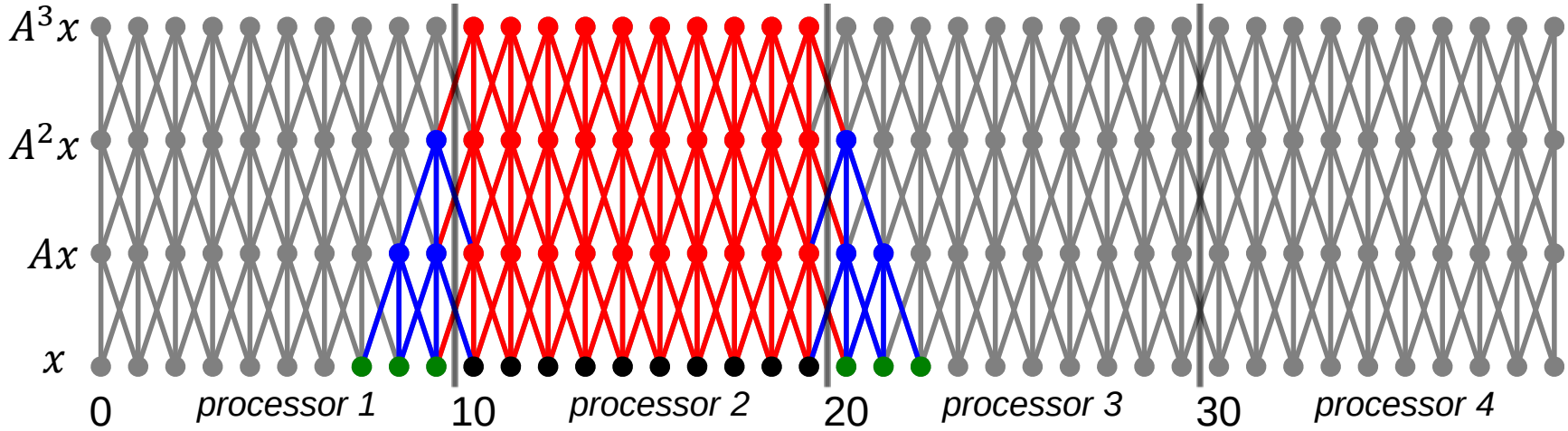




Parallel Matrix Powers Kernel

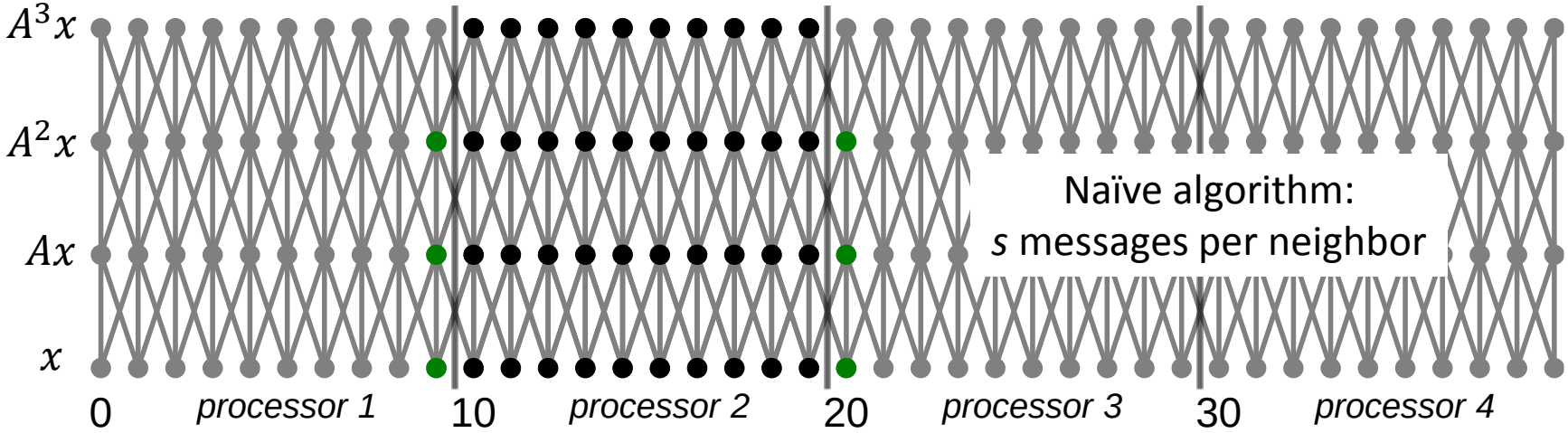
Example: tridiagonal matrix, s = 3, n = 40, p = 4

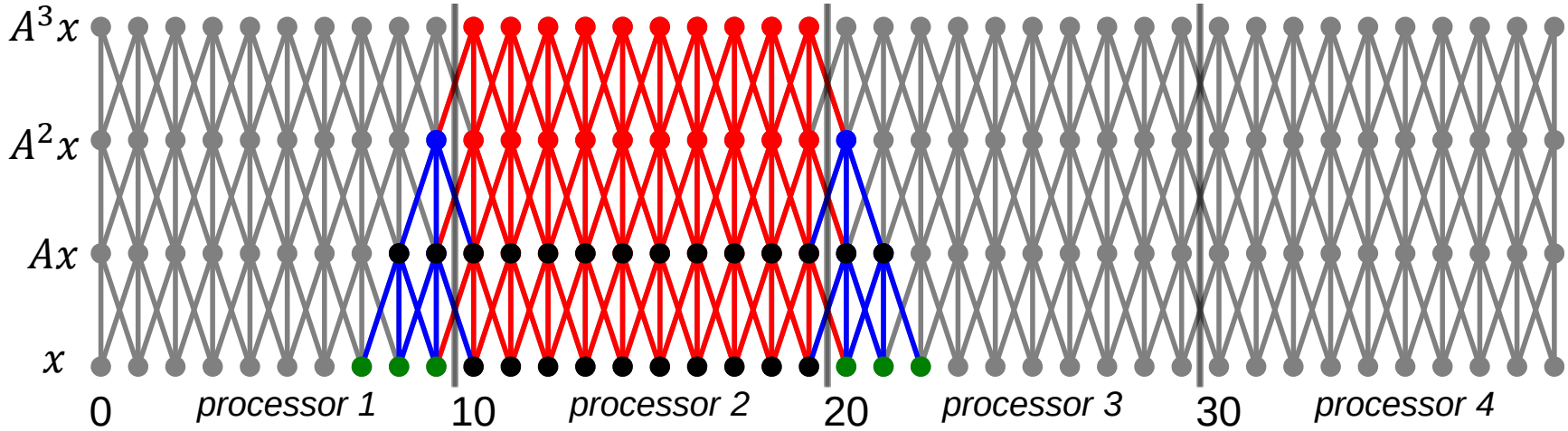




Parallel Matrix Powers Kernel

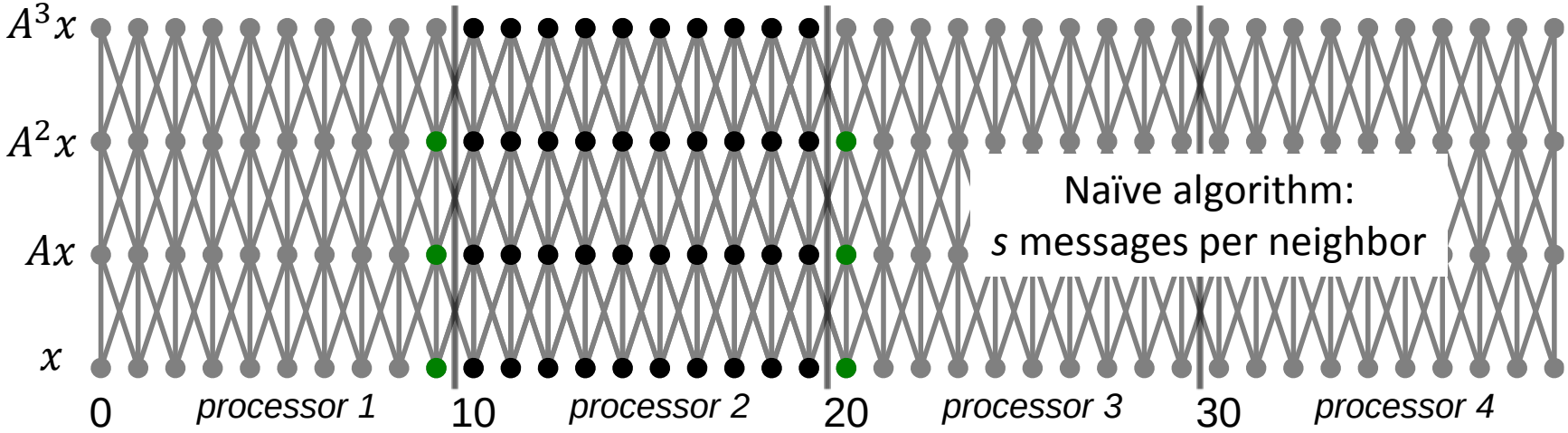
Example: tridiagonal matrix, s = 3, n = 40, p = 4

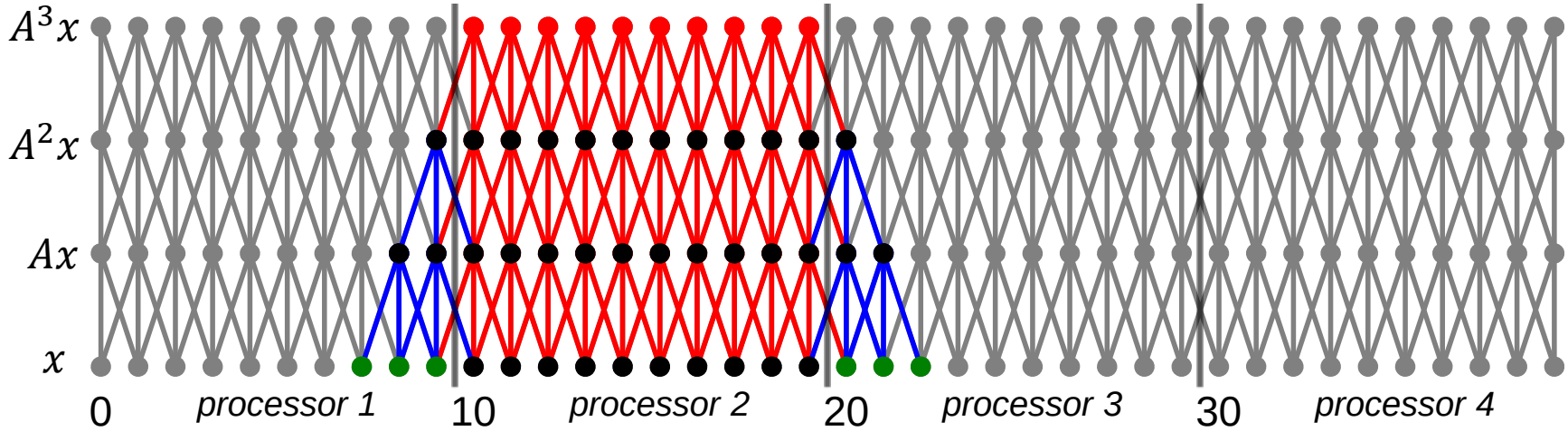




Parallel Matrix Powers Kernel

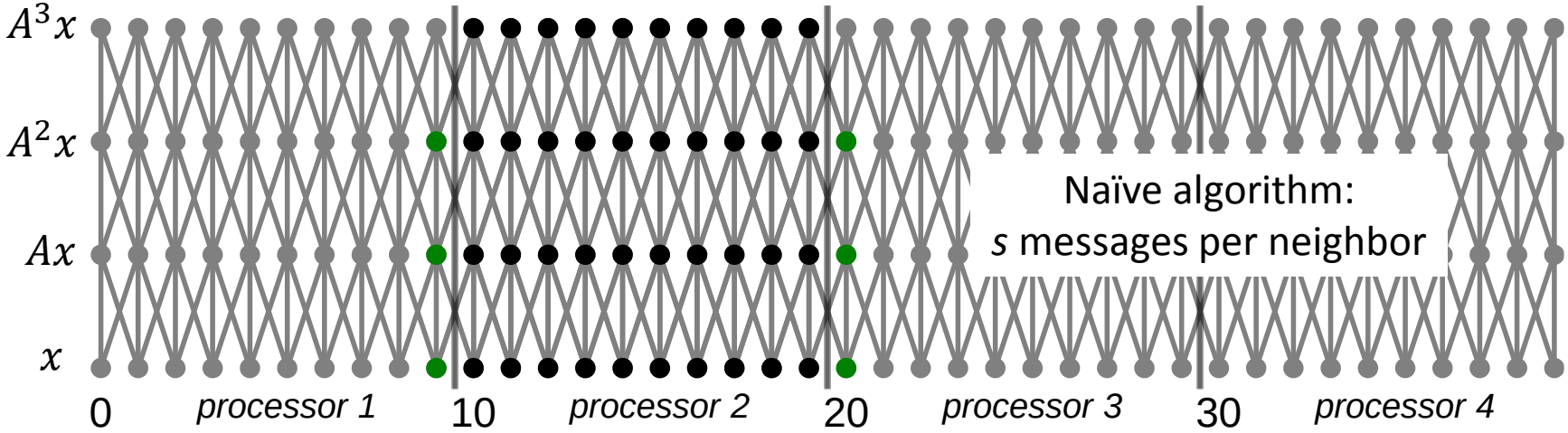
Example: tridiagonal matrix, s = 3, n = 40, p = 4

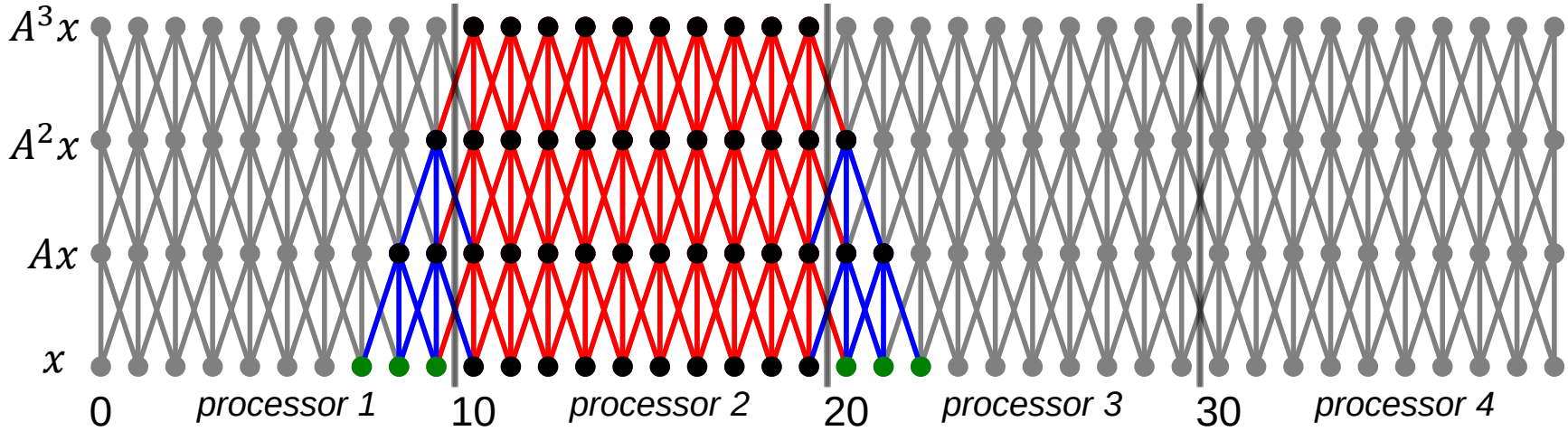




Parallel Matrix Powers Kernel

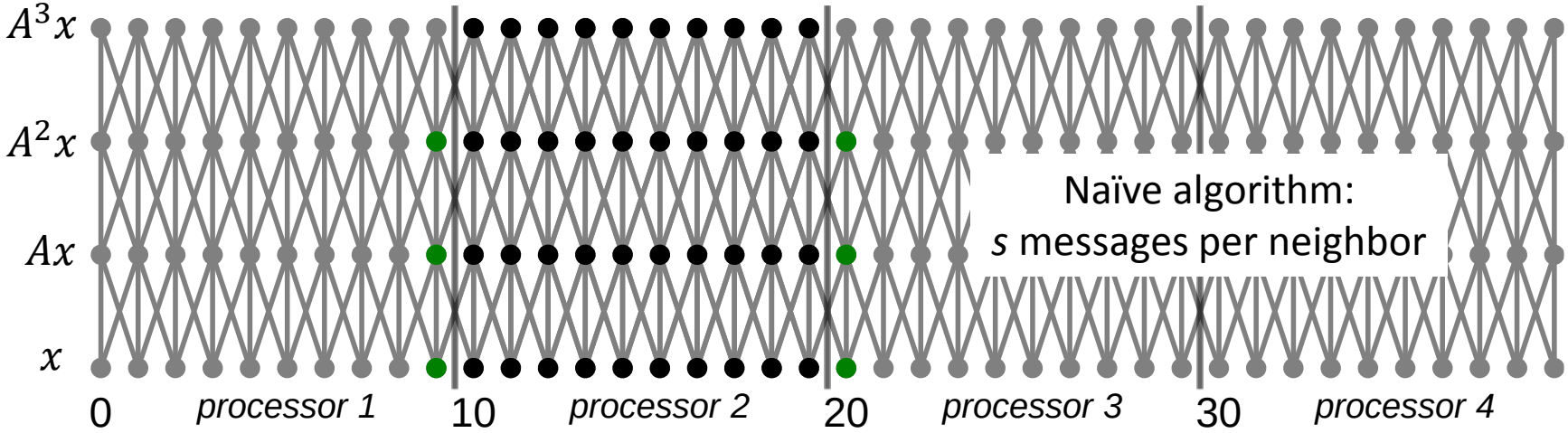
Example: tridiagonal matrix, s = 3, n = 40, p = 4

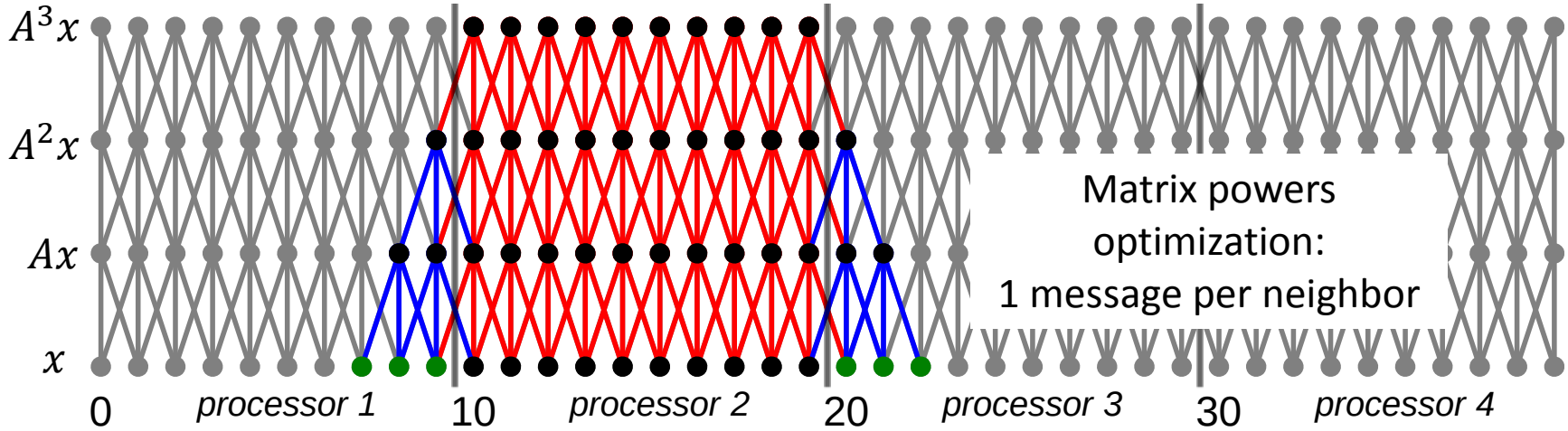




Parallel Matrix Powers Kernel

Example: tridiagonal matrix, s = 3, n = 40, p = 4





Complexity comparison

Example of parallel (per processor) complexity for s iterations of CG vs. s-step CG for a 2D 9-point stencil:

(Assuming each of p processors owns N/p rows of the matrix and $s \leq \sqrt{N/p}$)

	Flops		Words Moved		Messages	
	SpMV	Orth.	SpMV	Orth.	SpMV	Orth.
Classical CG	$\frac{sN}{p}$	$\frac{sN}{p}$	$s\sqrt{N/p}$	$s \log_2 p$	s	$s \log_2 p$
s-step CG	$\frac{sN}{p}$	$\frac{s^2N}{p}$	$s\sqrt{N/p}$	$s^2 \log_2 p$	1	$\log_2 p$

All values in the table meant in the Big-O sense (i.e., lower order terms and constants not included)

Complexity comparison

Example of parallel (per processor) complexity for s iterations of CG vs. s-step CG for a 2D 9-point stencil:

(Assuming each of p processors owns N/p rows of the matrix and $s \leq \sqrt{N/p}$)

	Flops		Words Moved		Messages	
	SpMV	Orth.	SpMV	Orth.	SpMV	Orth.
Classical CG	$\frac{sN}{p}$	$\frac{sN}{p}$	$s\sqrt{N/p}$	$s \log_2 p$	s	$s \log_2 p$
s-step CG	$\frac{sN}{p}$	$\frac{s^2N}{p}$	$s\sqrt{N/p}$	$s^2 \log_2 p$	1	$\log_2 p$

All values in the table meant in the Big-O sense (i.e., lower order terms and constants not included)

Complexity comparison

Example of parallel (per processor) complexity for s iterations of CG vs. s-step CG for a 2D 9-point stencil:

(Assuming each of p processors owns N/p rows of the matrix and $s \leq \sqrt{N/p}$)

	Flops		Words Moved		Messages	
	SpMV	Orth.	SpMV	Orth.	SpMV	Orth.
Classical CG	$\frac{sN}{p}$	$\frac{sN}{p}$	$s\sqrt{N/p}$	$s \log_2 p$	s	$s \log_2 p$
s-step CG	$\frac{sN}{p}$	$\frac{s^2N}{p}$	$s\sqrt{N/p}$	$s^2 \log_2 p$	1	$\log_2 p$

All values in the table meant in the Big-O sense (i.e., lower order terms and constants not included)

s-step GMRES

Classical GMRES

$r_0 = b - Ax_0, v_0 = r_0/\|r_0\|$
for $i = 1{:}\,k$
    $w = Av_{i-1}$
    Orthogonalize w against $[v_0, \dots, v_{i-1}]$ ← e.g., Modified Gram-Schmidt
    Update vector v_i, matrix H
end
Use $H, [v_0, \dots, v_k]$ to construct the solution

s-step GMRES

Classical GMRES

$r_0 = b - Ax_0, v_0 = r_0/\|r_0\|$
for $i = 1{:}k$
    $w = Av_{i-1}$
    Orthogonalize w against $[v_0, \dots, v_{i-1}]$ ← e.g., Modified Gram-Schmidt
    Update vector v_i, matrix H
end
Use $H, [v_0, \dots, v_k]$ to construct the solution

s-step GMRES

$r_0 = b - Ax_0, v_0 = r_0/\|r_0\|$
for $i = 0{:}s{:}k - s$
    Compute W such that span($[v_i, W]$) = $\mathcal{K}_{s+1}(A, v_i)$ ← "matrix powers kernel"
    Make W orthogonal against $[v_0, \dots, v_i]$ ← Block Gram-Schmidt
    Make W orthogonal ← "Tall-Skinny QR"
    Update $[v_{i+1}, \dots, v_{i+s}]$, matrix H
end
Use $H, [v_0, \dots, v_k]$ to construct the solution

Tall-Skinny QR (TSQR)

- TSQR: QR factorization of a tall skinny matrix using Householder transformations
- QR decomposition of m x b matrix W, m >> b
  - P processors, block row layout

Tall-Skinny QR (TSQR)

- TSQR: QR factorization of a tall skinny matrix using Householder transformations
- QR decomposition of m x b matrix W, m >> b
  - P processors, block row layout

- **Classic Parallel Algorithm**
  - Compute Householder vector for each column
  - Number of messages $\propto$ b log P

Tall-Skinny QR (TSQR)

- TSQR: QR factorization of a tall skinny matrix using Householder transformations
- QR decomposition of m x b matrix W, m >> b
  - P processors, block row layout

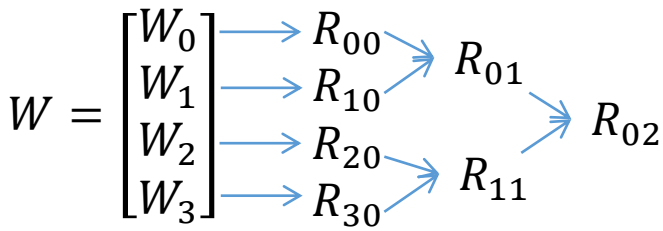
- **Classic Parallel Algorithm**
  - Compute Householder vector for each column
  - Number of messages $\propto$ b log P
- **Communication Avoiding Algorithm**
  - Reduction operation, with QR as operator
  - Number of messages $\propto$ log P

Tall-Skinny QR (TSQR)

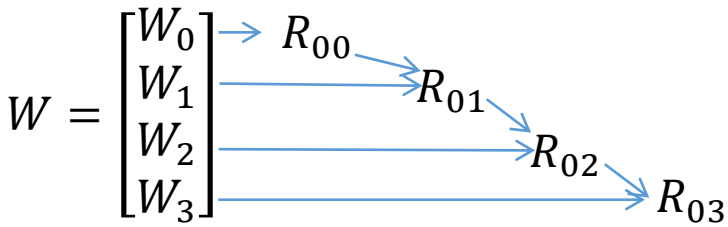
- TSQR: QR factorization of a tall skinny matrix using Householder transformations
- QR decomposition of m x b matrix W, m >> b
  - P processors, block row layout

- **Classic Parallel Algorithm**
  - Compute Householder vector for each column
  - Number of messages ∝ b log P
- **Communication Avoiding Algorithm**
  - Reduction operation, with QR as operator
  - Number of messages ∝ log P

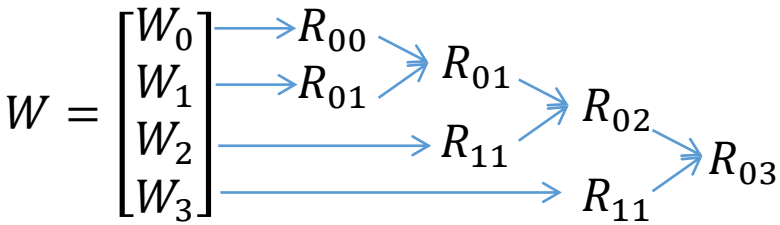
Parallel



Sequential



Dual Core



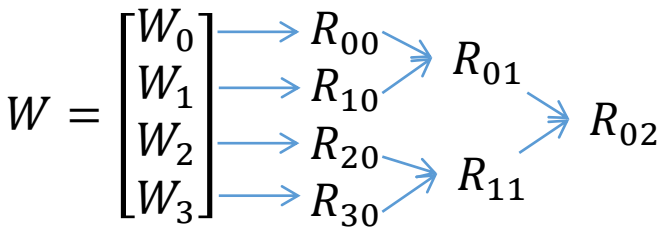
Tall-Skinny QR (TSQR)

- TSQR: QR factorization of a tall skinny matrix using Householder transformations
- QR decomposition of m x b matrix W, m >> b
  - P processors, block row layout

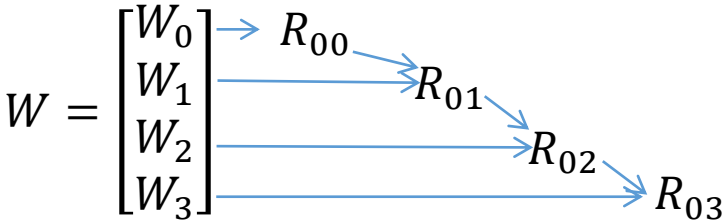
- **Classic Parallel Algorithm**
  - Compute Householder vector for each column
  - Number of messages ∝ b log P
- **Communication Avoiding Algorithm**
  - Reduction operation, with QR as operator
  - Number of messages ∝ log P

TSQR implementations in Intel MKL library, GNU Scientific Library, ScaLAPACK, Spark

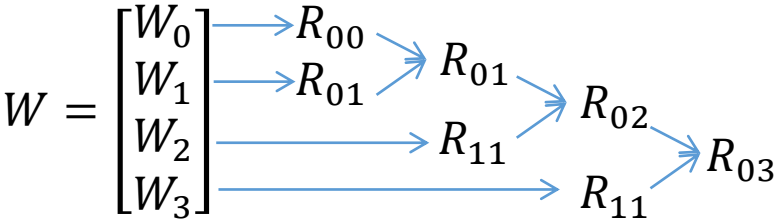
Parallel



Sequential

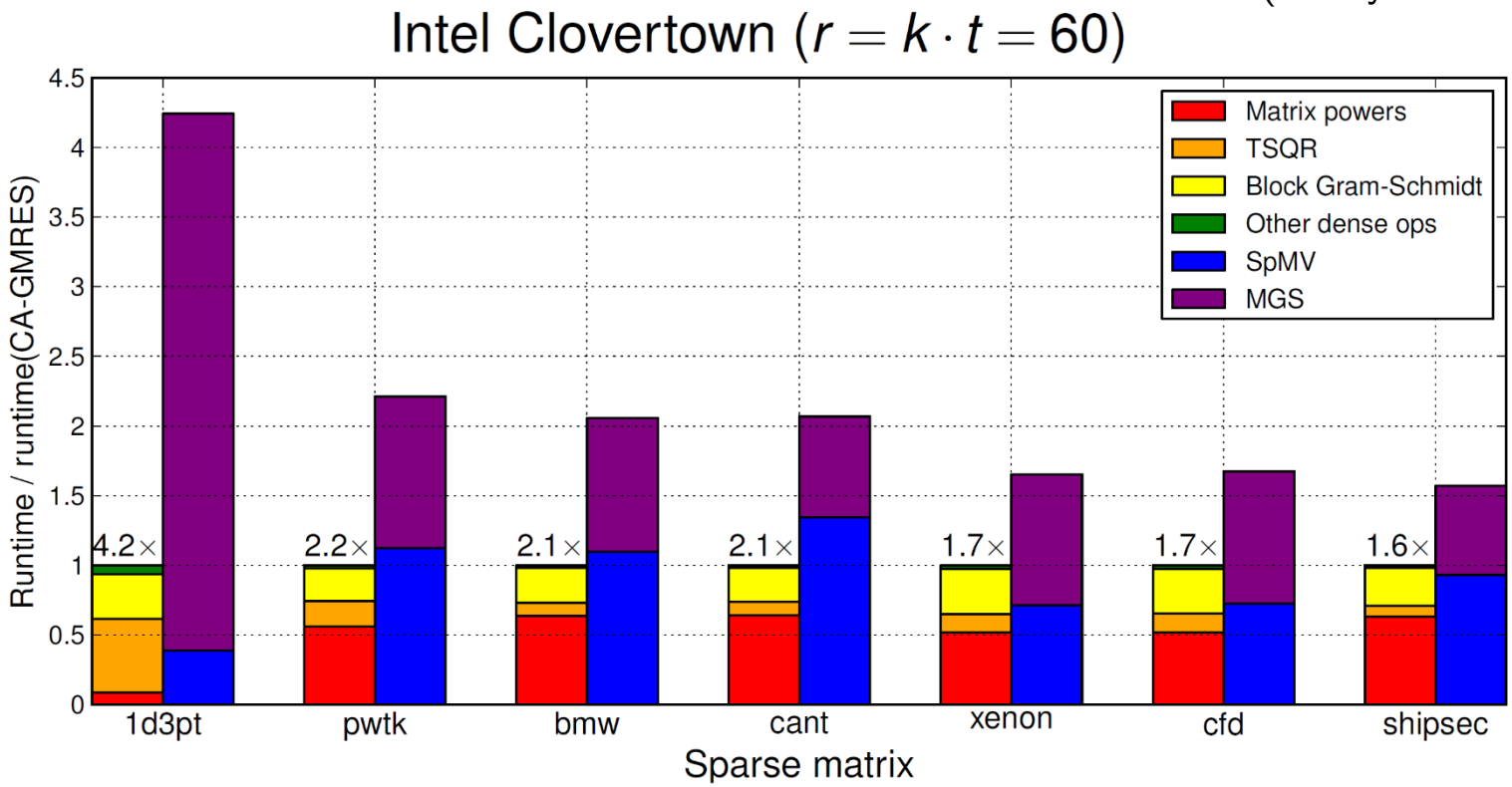


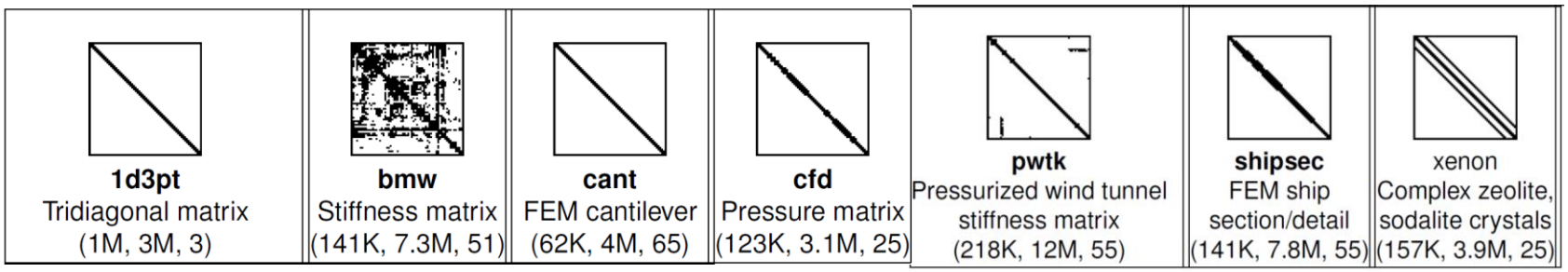
Dual Core



Performance Results

(Mohiyuddin et al, 2009)





Performance and Applications

- Performance studies
  - s-step GMRES on hybrid CPU/GPU arch. (Yamazaki et al., 2014)
  - comparison of s-step and pipelined GMRES (Yamazaki et al., 2017)

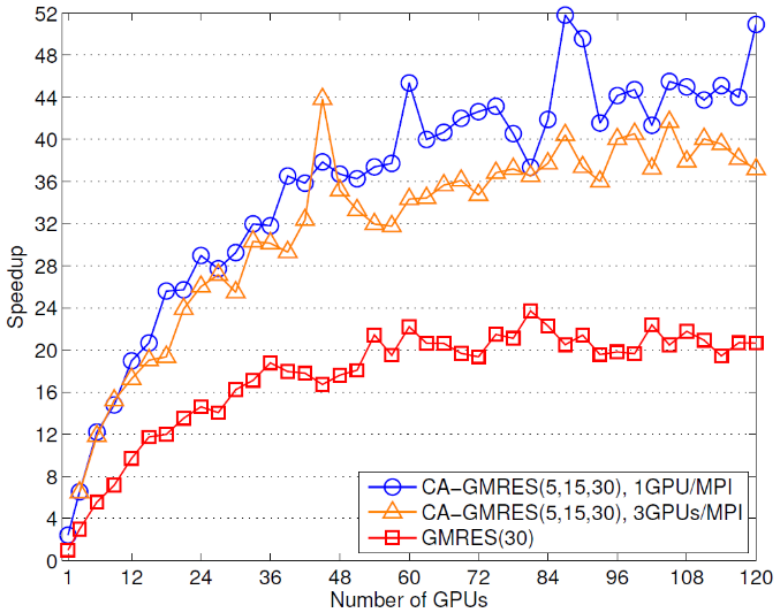


Fig. 6. Parallel Strong Scaling of CA-GMRES and GMRES on 120 distributed GPUs (over GMRES on one GPU), for the `G3_Circuit` matrix.

Performance and Applications

- Performance studies
  - s-step GMRES on hybrid CPU/GPU arch. (Yamazaki et al., 2014)
  - comparison of s-step and pipelined GMRES (Yamazaki et al., 2017)

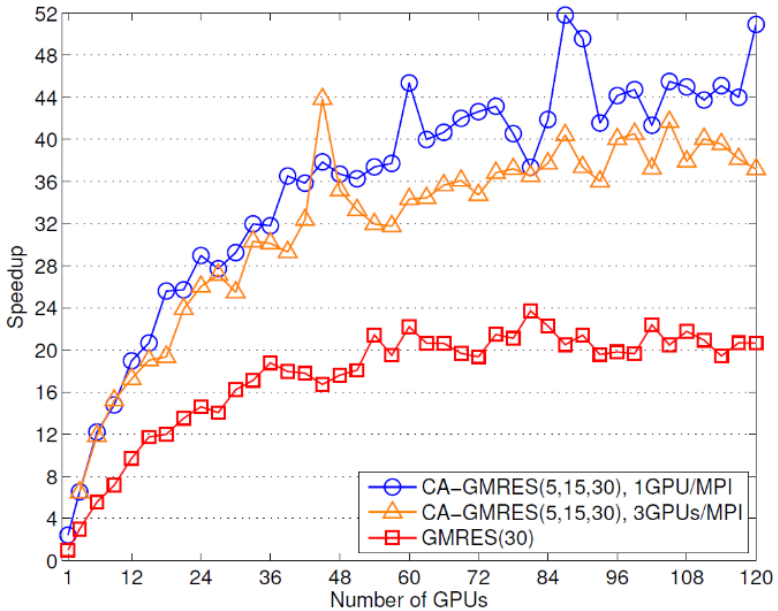


Fig. 6. Parallel Strong Scaling of CA-GMRES and GMRES on 120 distributed GPUs (over GMRES on one GPU), for the `G3_Circuit` matrix.

- Example applications: s-step BICGSTAB used in
  - combustion, cosmology [Williams, C., et al., IPDPS, 2014]
  - geoscience dynamics [Anciaux-Sedrakian et al., 2016]
  - far-field scattering [Zhang et al., 2016]
  - wafer defect detection [Zhang et al., 2016]

Performance and Applications

- Performance studies
  - s-step GMRES on hybrid CPU/GPU arch. (Yamazaki et al., 2014)
  - comparison of s-step and pipelined GMRES (Yamazaki et al., 2017)

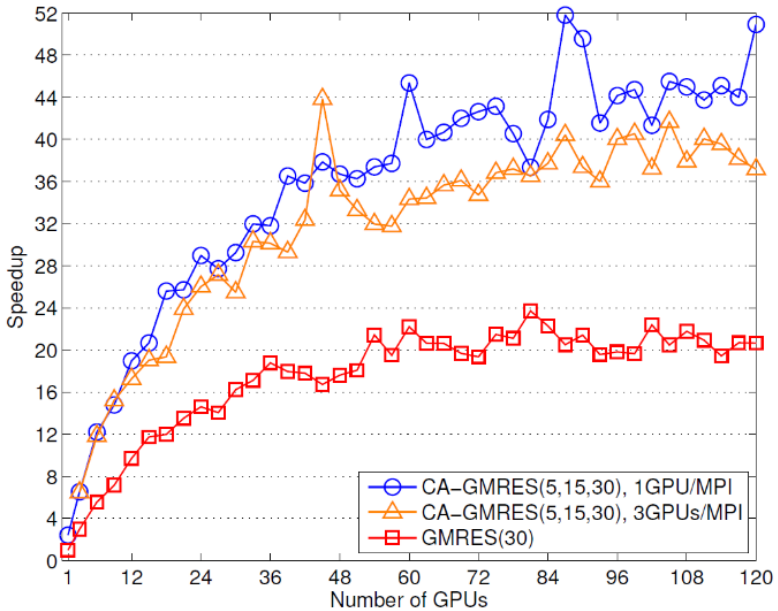


Fig. 6. Parallel Strong Scaling of CA-GMRES and GMRES on 120 distributed GPUs (over GMRES on one GPU), for the `G3_Circuit` matrix.

- Example applications: s-step BICGSTAB used in
  - combustion, cosmology [Williams, C., et al., IPDPS, 2014]
  - geoscience dynamics [Anciaux-Sedrakian et al., 2016]
  - far-field scattering [Zhang et al., 2016]
  - wafer defect detection [Zhang et al., 2016]

up to **4.2x** on 24K cores on Cray XE6

Alternative Approaches

- Enlarged Krylov subspace methods (Grigori, Moufawad, Nataf, 2016)
  - Split vector into t parts based on domain decomposition of A; enlarge Krylov subspace by t dimensions each iteration
    - Faster convergence, more parallelizable

- Combined s-step pipelined methods
  - (ℓ, s)-GMRES (Yamazaki, Hoemmen, Luszczek, Dongarra, 2017)
  - Hybrid approach which combines ideas of s-step and pipelined methods; reduces number of global synchronizations and also overlaps them with other work

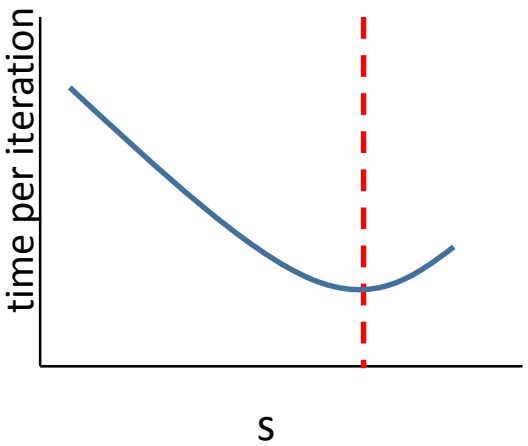
Practical Implementation Challenges

- How to pick parameters? (pipeline depth in pipelined method; s in s-step method)
  - Choice must take into account matrix structure, machine, partition, as well as numerical properties (more on this next time!)

- Preconditioning
  - Must consider overlap in pipelined methods (if enough to overlap with)
  - For s-step, can diminish potential gain from matrix powers kernel if preconditioner is dense (but still win from savings in Allreduce)

Choosing s

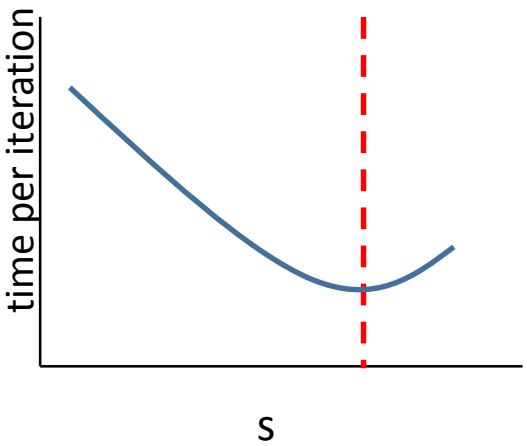
- How do we expect communication costs to change as s increases?
- Initially decrease, but at some point, start increasing
  - Point depends on sparsity structure of matrix, partition of matrix, and latency/bandwidth parameters of the machine
- Bandwidth cost can start to dominate
- For s large enough, the extra entries we need go past our neighbors boundaries
    - more messages required -> increased latency cost



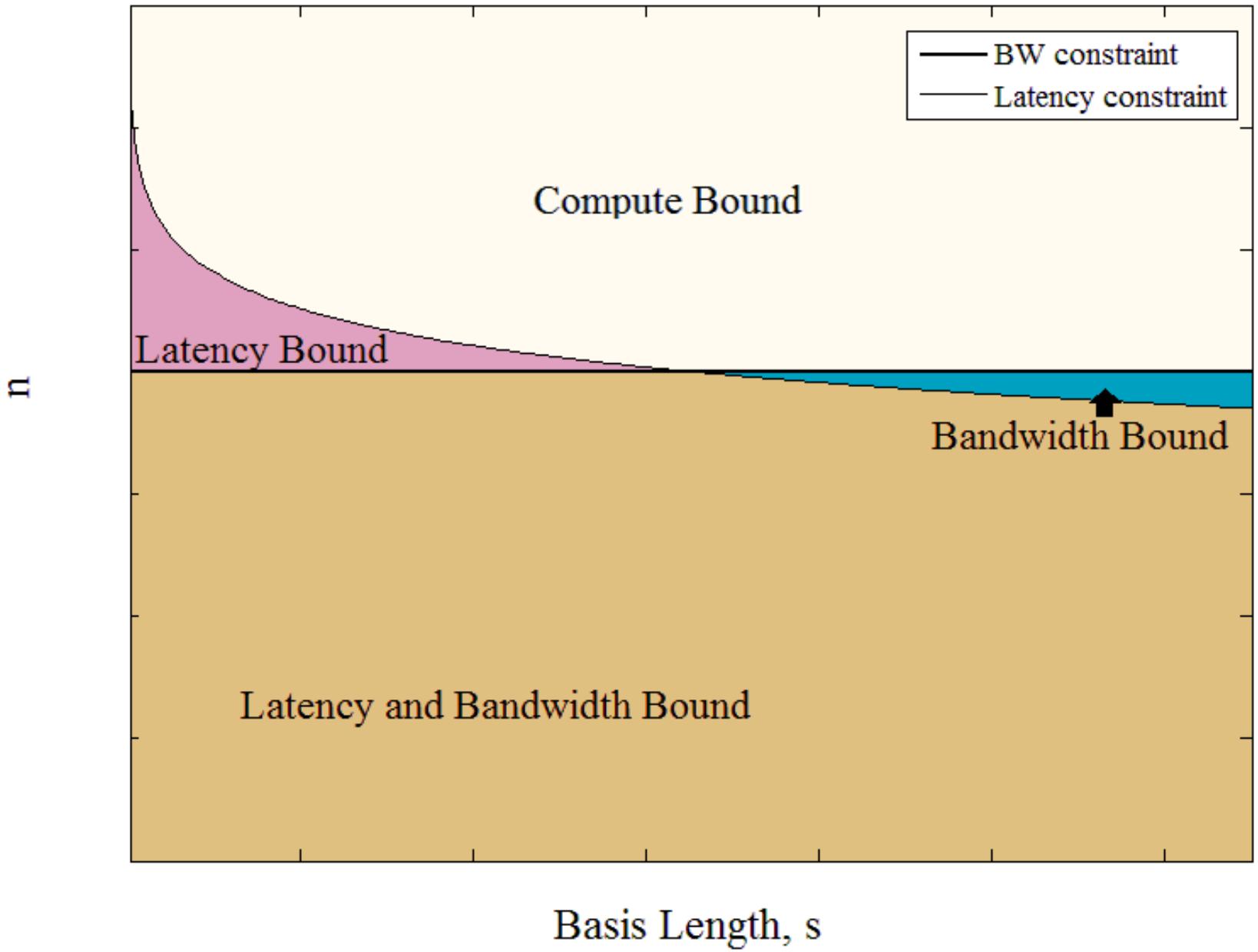
	Flops		Words Moved		Messages	
	SpMV	Orth.	SpMV	Orth.	SpMV	Orth.
Classical CG	$\frac{sN}{p}$	$\frac{sN}{p}$	$s\sqrt{N/p}$	$s \log_2 p$	s	$s \log_2 p$
s-step CG	$\frac{sN}{p}$	$\frac{s^2N}{p}$	$s\sqrt{N/p}$	$s^2 \log_2 p$	1	$\log_2 p$

Choosing s

- How do we expect communication costs to change as s increases?
- Initially decrease, but at some point, start increasing
  - Point depends on sparsity structure of matrix, partition of matrix, and latency/bandwidth parameters of the machine
- Bandwidth cost can start to dominate
- For s large enough, the extra entries we need go past our neighbors boundaries
    - more messages required -> increased latency cost
- For GMRES, best s for matrix powers may not be best s for TSQR kernel
  - Choice of s requires co-tuning



	Flops		Words Moved		Messages	
	SpMV	Orth.	SpMV	Orth.	SpMV	Orth.
Classical CG	$\frac{sN}{p}$	$\frac{sN}{p}$	$s\sqrt{N/p}$	$s \log_2 p$	s	$s \log_2 p$
s-step CG	$\frac{sN}{p}$	$\frac{s^2N}{p}$	$s\sqrt{N/p}$	$s^2 \log_2 p$	1	$\log_2 p$


BW constraint
Latency constraint
Compute Bound
Latency Bound
Bandwidth Bound
Latency and Bandwidth Bound
n
Basis Length, s

Lower Bound Tradeoffs for Matrix Powers

- Solomonik, C., Knight, Demmel (2014): Lower bounds on tradeoffs between three basic costs of a parallel algorithm: synchronization, data movement, and computational cost.
- By considering critical path, tradeoffs give lower bounds on the execution time which are dependent on the problem size but independent of the number of processors (assuming homogeneity)

Lower Bound Tradeoffs for Matrix Powers

- Solomonik, C., Knight, Demmel (2014): Lower bounds on tradeoffs between three basic costs of a parallel algorithm: synchronization, data movement, and computational cost.
- By considering critical path, tradeoffs give lower bounds on the execution time which are dependent on the problem size but independent of the number of processors (assuming homogeneity)
- Theorem: Any parallel execution of an s-dimensional Krylov basis computation for a $(2m+1)^d$-point stencil on a d-dimensional regular mesh requires

$\Omega(m^d b^d s)$ flops, $\Omega(m^d b^{d-1} s)$ words, $\Omega(s/b)$ messages,

for some $b \in \{1, \dots, s\}$

Lower Bound Tradeoffs for Matrix Powers

- Solomonik, C., Knight, Demmel (2014): Lower bounds on tradeoffs between three basic costs of a parallel algorithm: synchronization, data movement, and computational cost.
- By considering critical path, tradeoffs give lower bounds on the execution time which are dependent on the problem size but independent of the number of processors (assuming homogeneity)
- Theorem: Any parallel execution of an s-dimensional Krylov basis computation for a $(2m+1)^d$-point stencil on a d-dimensional regular mesh requires

$\Omega(m^d b^d s)$ flops, $\Omega(m^d b^{d-1} s)$ words, $\Omega(s/b)$ messages,

for some $b \in \{1, \dots, s\}$

- Matrix powers kernel attains this lower bound when $n^d/p \geq m^d b^d$ where n^d is # mesh points

Performance Modeling to Estimate Parameters

- Goal: estimate best blocking factor b for matrix powers computation

Performance Modeling to Estimate Parameters

- Goal: estimate best blocking factor b for matrix powers computation
- Cost model:

$$\text{Time} = \gamma \times \text{flops} + \beta \times \text{words moved} + \alpha \times \text{\# messages}$$

Performance Modeling to Estimate Parameters

- Goal: estimate best blocking factor b for matrix powers computation
- Cost model:

Time = $\gamma \times$ flops + $\beta \times$ words moved + $\alpha \times$ # messages

- Choose b to minimize

$$\text{Time} \sim \gamma\, m^d b^d s + \beta m^d b^{d-1} s + \alpha\, s/b$$

Performance Modeling to Estimate Parameters

- Goal: estimate best blocking factor b for matrix powers computation
- Cost model:

$$\text{Time} = \gamma \times \text{flops} + \beta \times \text{words moved} + \alpha \times \#\ \text{messages}$$

- Choose b to minimize

$$\text{Time} \sim \gamma\, m^d b^d s + \beta m^d b^{d-1} s + \alpha\, s/b$$

- Latency/BW tradeoff point : $b \sim \frac{\alpha^{1/d}}{m\beta^{1/d}}$

Performance Modeling to Estimate Parameters

- Goal: estimate best blocking factor b for matrix powers computation
- Cost model:

$$\text{Time} = \gamma \times \text{flops} + \beta \times \text{words moved} + \alpha \times \text{\# messages}$$

- Choose b to minimize

$$\text{Time} \sim \gamma\, m^d b^d s + \beta m^d b^{d-1} s + \alpha\, s/b$$

- Latency/BW tradeoff point : $b \sim \frac{\alpha^{1/d}}{m\beta^{1/d}}$
- Starting place for parameter selection – to get close to optimal answer, would need more accurate model of time, costs including constants

Matrix Partitioning

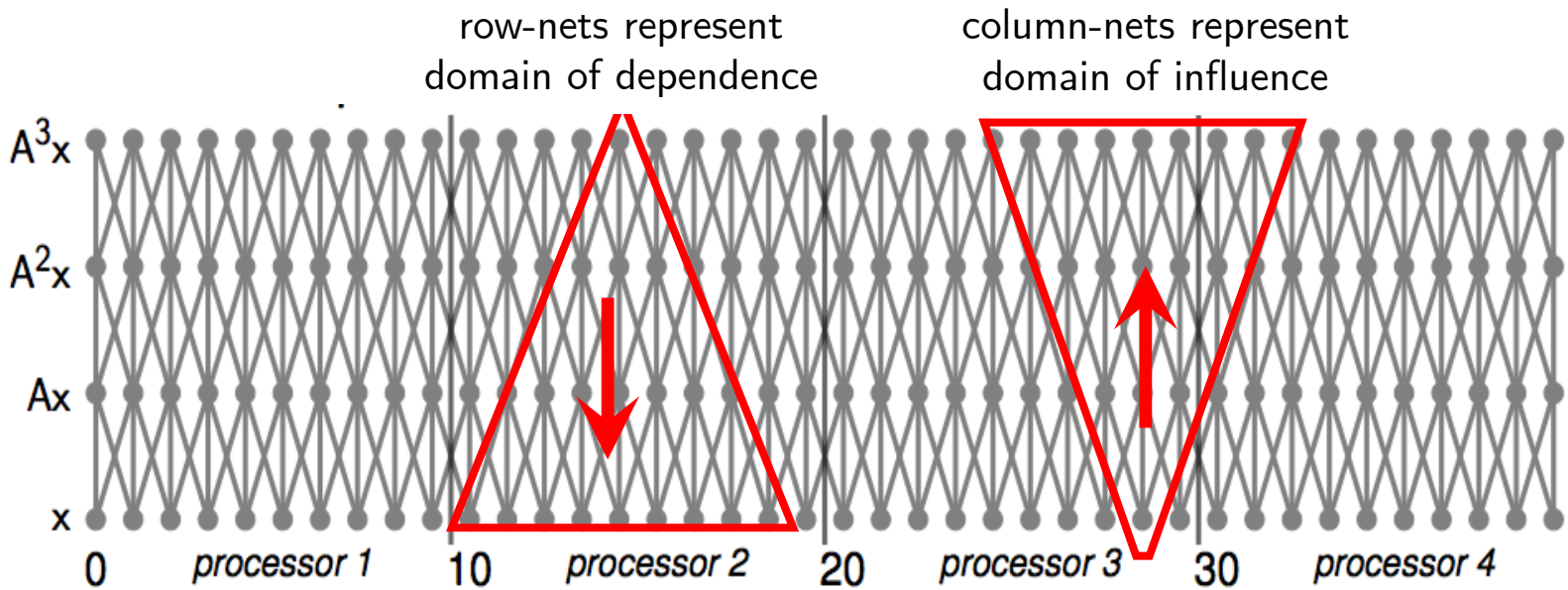
- For computing matrix powers (i.e., constructing the basis matrix in s-step methods, we really want to partition the structure of A^s rather than A
  - Analogous to single SpMV, can construct a hypergraph model such that the minimum cut gives a partition with minimum communication volume

- Load balancing
  - The parallel matrix powers kernel involves redundantly computing entries of the vectors on different processors
  - Entries which need to be redundantly computed determined by partition

Hypergraph Partitioning for Matrix Powers

Parallel communication for $V = [x, Ax, A^2x, \ldots, A^s x]$, given a sparse tiling of A	=	Parallel communication for $y = A^s x$, given 1D rowwise layout of A^s

(assuming no cancellation and nonzero diagonal)

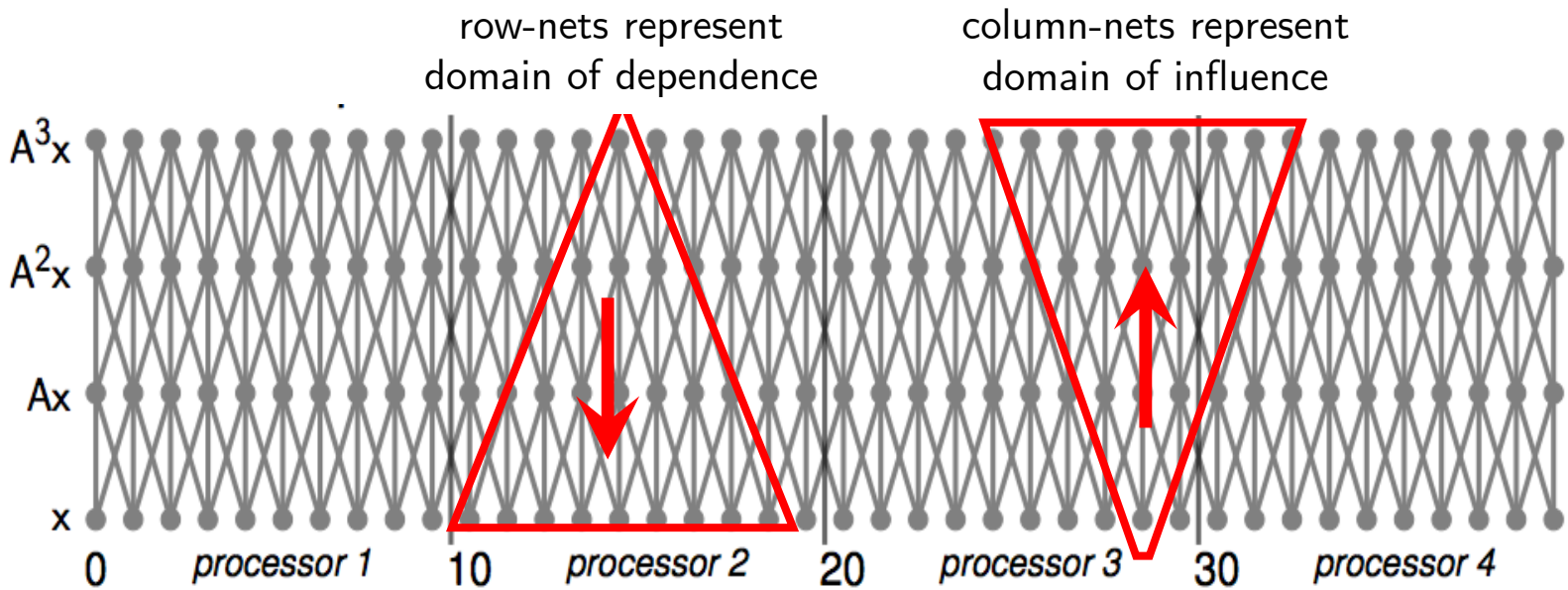


- "s-level" row- and column-nets encode the structure of A^s

Hypergraph Partitioning for Matrix Powers

Parallel communication for $V = [x, Ax, A^2x, \ldots, A^s x]$, given a sparse tiling of A	=	Parallel communication for $y = A^s x$, given 1D rowwise layout of A^s

(assuming no cancellation and nonzero diagonal)



- "s-level" row- and column-nets encode the structure of A^s
- But expensive to compute (s × Boolean sparse matrix-matrix multiplies)
  - Only worth it if A has particularly irregular sparsity structure (e.g., number of nonzeros per column in A^i grows at various rates) and same matrix will be reused
  - Potential use of randomized algorithms to estimate nnz/column in A^i

Preconditioning for s-step variants

- Preconditioners improve spectrum of system to improve convergence rate
  - E.g., instead of $Ax = b$, solve $M^{-1}Ax = M^{-1}b$, where $M^{-1} \approx A^{-1}$
  - Essential in practice

Preconditioning for s-step variants

- Preconditioners improve spectrum of system to improve convergence rate
  - E.g., instead of $Ax = b$, solve $M^{-1}Ax = M^{-1}b$, where $M^{-1} \approx A^{-1}$
  - Essential in practice

- In s-step variants, general preconditioning is a challenge
  - Except for very simple cases, ability to exploit temporal locality (in matrix powers computation) across iterations is diminished by preconditioning
    - Still potential gain from blocking inner products/avoiding global synchronization
  - If possible to avoid communication at all, usually necessitates significant modifications to the algorithm

Preconditioning for s-step variants

- Preconditioners improve spectrum of system to improve convergence rate
  - E.g., instead of $Ax = b$, solve $M^{-1}Ax = M^{-1}b$, where $M^{-1} \approx A^{-1}$
  - Essential in practice

- In s-step variants, general preconditioning is a challenge
  - Except for very simple cases, ability to exploit temporal locality (in matrix powers computation) across iterations is diminished by preconditioning
    - Still potential gain from blocking inner products/avoiding global synchronization
  - If possible to avoid communication at all, usually necessitates significant modifications to the algorithm

- Tradeoff: speed up convergence, but increase time per iteration due to communication!
  - For each specific app, must evaluate tradeoff between **preconditioner quality** and **sparsity** of the system

Preconditioning for s-step KSMs

- Much recent/ongoing work in developing communication-avoiding preconditioned methods

- Many approaches shown to be compatible
  - Diagonal
  - Sparse Approx. Inverse (SPAI) – for s-step BICGSTAB by Mehri (2014)
  - HSS preconditioning (Hoemmen, 2010); for banded matrices (Knight, C., Demmel, 2014); same general technique for any system that can be written as sparse + low-rank
  - Deflation for s-step CG (C., Knight, Demmel, 2014), for s-step GMRES (Yamazaki et al., 2014)
  - CA-ILU(0) – Moufawad and Grigori (2013)
  - Domain decomposition – avoid introducing additional communication by "underlapping" subdomains (Yamazaki et al., 2014)

"Underlapping" Domain Decomposition

(Yamazaki et al., 2014)

- Variant of an additive Schwarz preconditioner, modified to ensure consistent interfaces between the subdomains without additional communication beyond what is required by sparsity structure of A

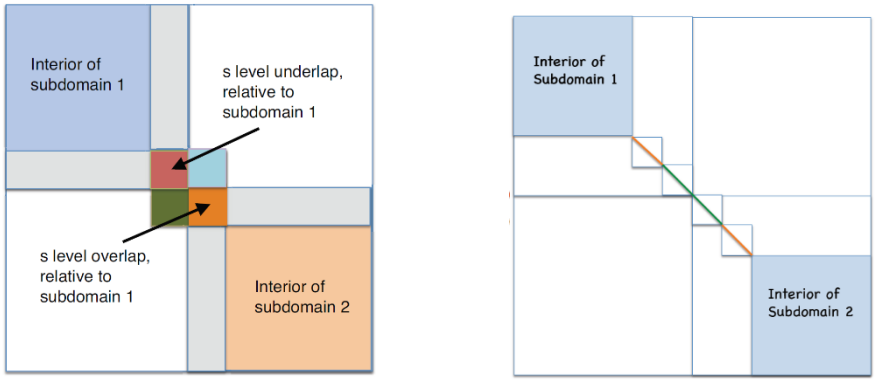
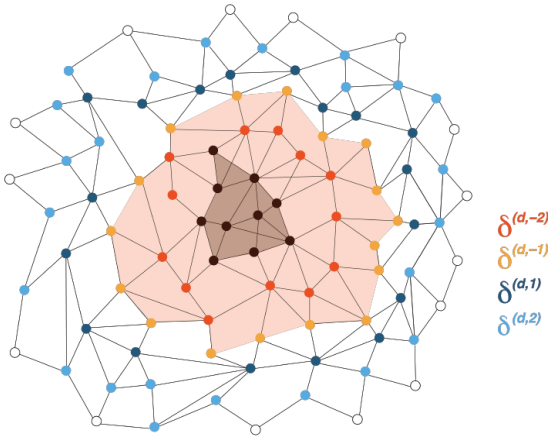


Fig. 8. Matrix Partitioning for the CA Preconditioner for two subdomains. The underlap and the overlap relative to subdomain 1 are shown.



In order to "localize" effects of preconditioner,

- form "interior" by removing s-level "underlap"
- apply "local" preconditioner on "interior"
  - ILU(k), SAI(k), Jacobi, GaussSeidel, etc. on "interior"
- apply diagonal Jacobi on "underlap"

"Underlapping" Domain Decomposition

(Yamazaki et al., 2014)

- Variant of an additive Schwarz preconditioner, modified to ensure consistent interfaces between the subdomains without additional communication beyond what is required by sparsity structure of A

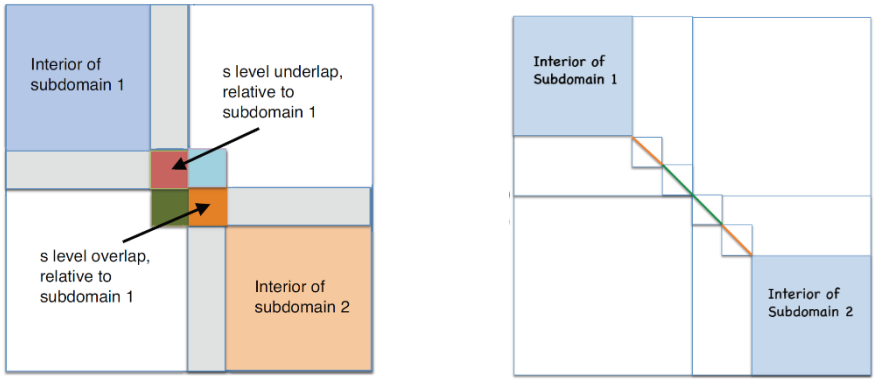
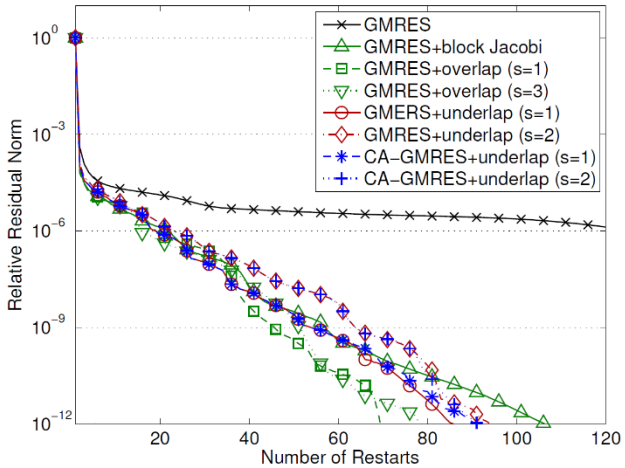
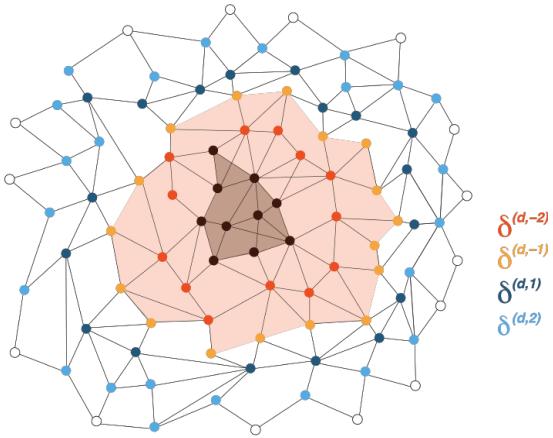


Fig. 8. Matrix Partitioning for the CA Preconditioner for two subdomains. The underlap and the overlap relative to subdomain 1 are shown.



(b) G3_Circuit matrix, with restart = 30.

Fig. 11. Solution Convergence, using Different Domain Decomposition Preconditioners with Local ILU(0)'s on 6 GPUs.



In order to "localize" effects of preconditioner,

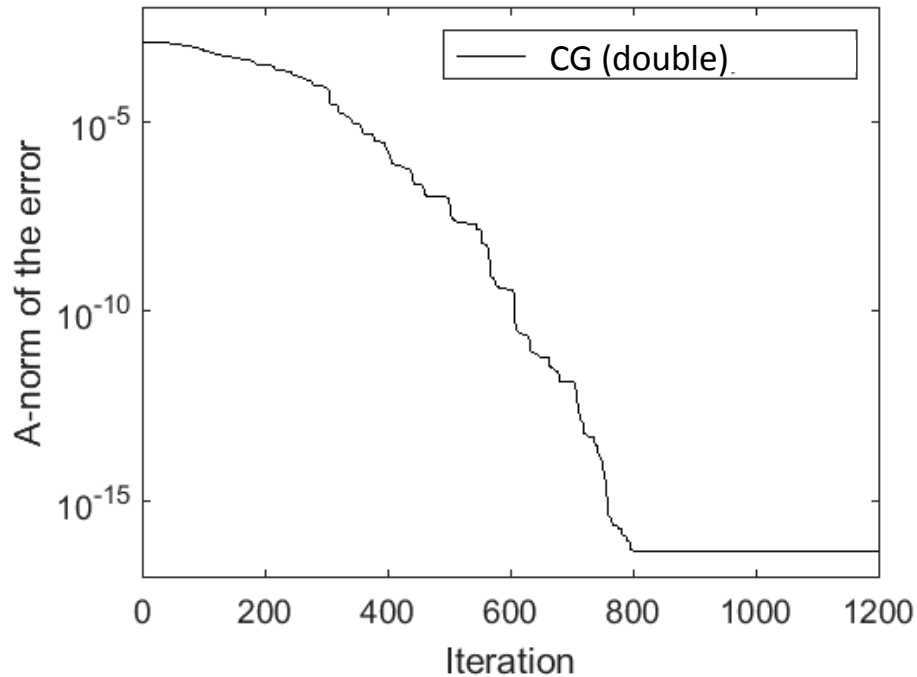
- form "interior" by removing s-level "underlap"
- apply "local" preconditioner on "interior"
  - ILU(k), SAI(k), Jacobi, GaussSeidel, etc. on "interior"
- apply diagonal Jacobi on "underlap"

The effects of finite precision

Well-known that roundoff error has two effects:

1. Delay of convergence
   - No longer have exact Krylov subspace
   - Can lose numerical rank deficiency
   - Residuals no longer orthogonal - Minimization of $\|x - x_i\|_A$ no longer exact

2. Loss of attainable accuracy
   - Rounding errors cause true residual $b - Ax_i$ and updated residual r_i deviate!



A: bcsstk03 from SuiteSparse,
b: equal components in the eigenbasis of $A, \|b\| = 1$
$N = 112, \kappa(A) \approx 7e6$

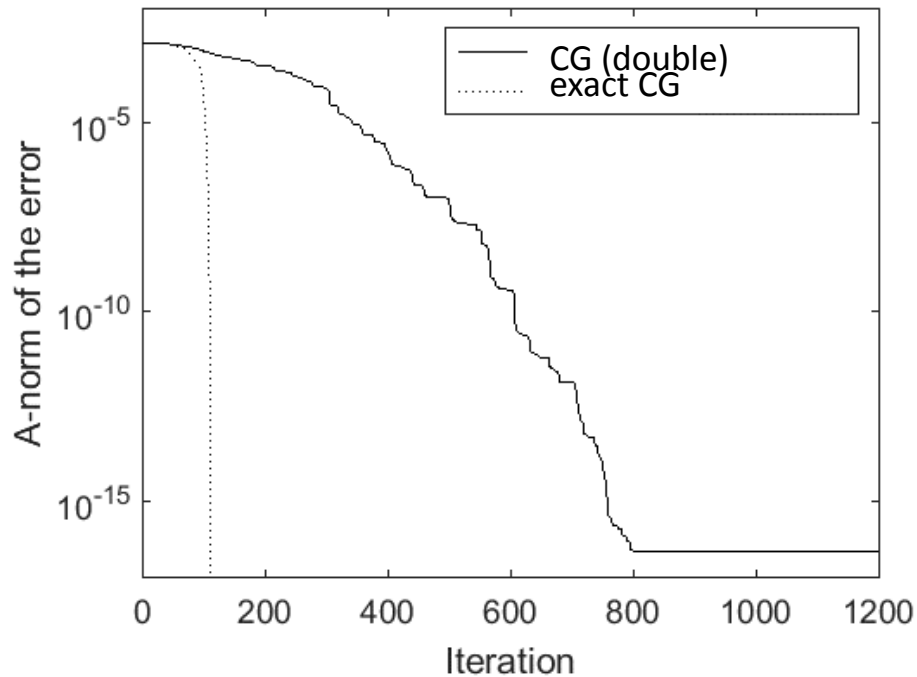
Much work on these results for CG; See Meurant and Strakoš (2006) for a thorough summary of early developments in finite precision analysis of Lanczos and CG

The effects of finite precision

Well-known that roundoff error has two effects:

1. Delay of convergence
   - No longer have exact Krylov subspace
   - Can lose numerical rank deficiency
   - Residuals no longer orthogonal - Minimization of $\|x - x_i\|_A$ no longer exact

2. Loss of attainable accuracy
   - Rounding errors cause true residual $b - Ax_i$ and updated residual r_i deviate!



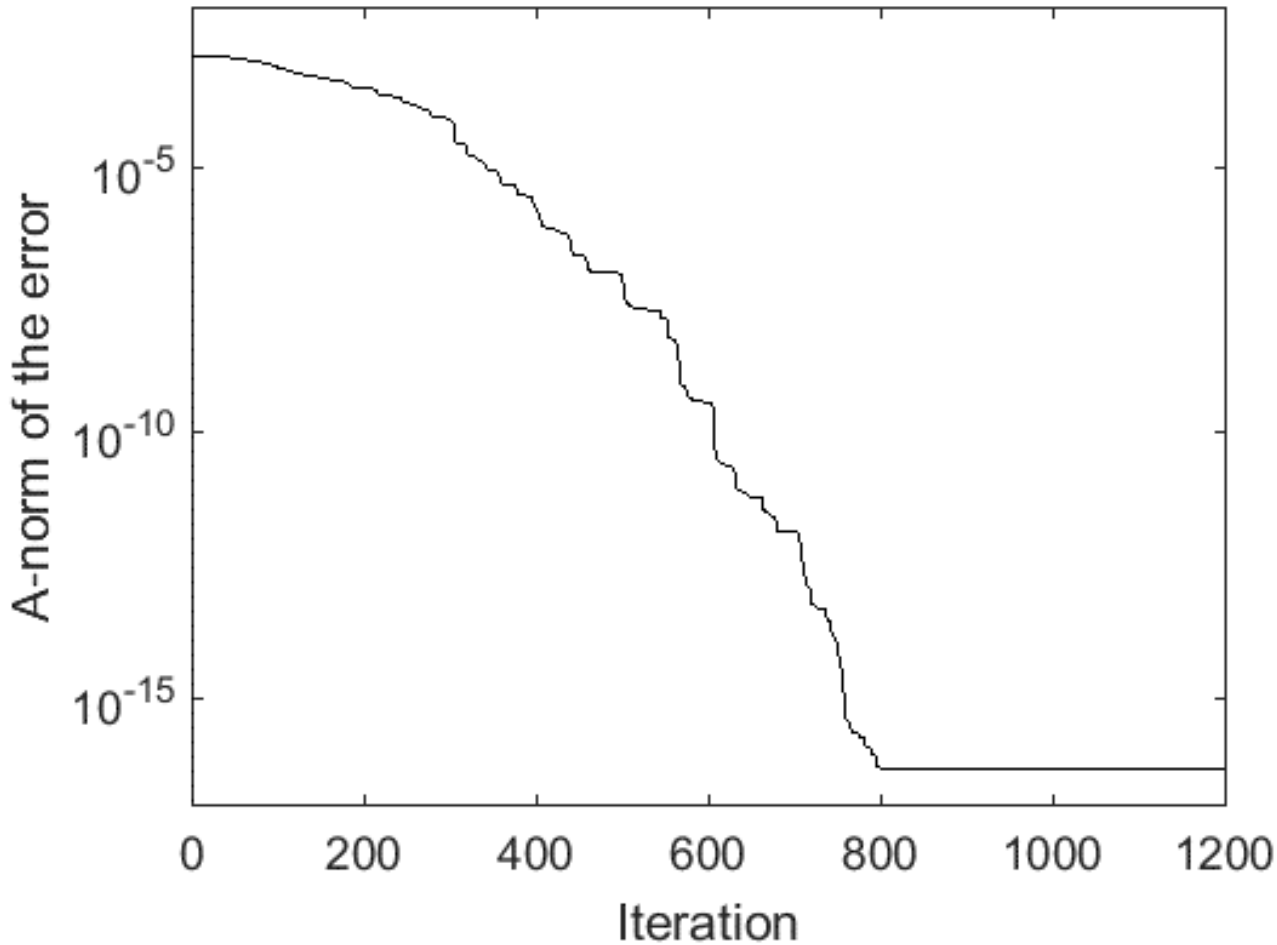
A: bcsstk03 from SuiteSparse,
b: equal components in the eigenbasis of A, $\|b\| = 1$
$N = 112, \kappa(A) \approx 7e6$

Much work on these results for CG; See Meurant and Strakoš (2006) for a thorough summary of early developments in finite precision analysis of Lanczos and CG

Conjugate Gradient method for solving Ax = b
double precision ($\varepsilon = 2^{-53}$)

$$\|x_i - x\|_A = \sqrt{(x_i - x)^T A (x_i - x)}$$

$$x_i = x_{i-1} + \alpha_i p_i$$
$$r_i = r_{i-1} - \alpha_i A p_i$$
$$p_i = r_i + \beta_i p_i$$



Conjugate Gradient method for solving Ax = b
double precision ($\varepsilon = 2^{-53}$)

$$\|x_i - x\|_A = \sqrt{(x_i - x)^T A (x_i - x)}$$

$$x_i = x_{i-1} + \alpha_i p_i$$
$$r_i = r_{i-1} - \alpha_i A p_i$$
$$p_i = r_i + \beta_i p_i$$