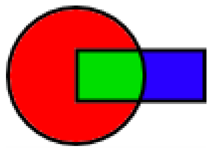
Introduction to Domain Decomposition Methods



David E. Keyes

Department of Applied Physics & Applied Mathematics

Columbia University

As delivered at Kavli Institute for Theoretical Physics, 17 March 2005

Definition and motivation

- **Domain decomposition (DD) is a "divide and conquer" technique for arriving at the solution of problem defined over a domain from the solution of related problems posed on subdomains**
- ***Motivating assumption #1:*** **the solution of the subproblems is qualitatively or quantitatively "easier" than the original**
- ***Motivating assumption #2:*** **the original problem does not fit into the available memory space**
- ***Motivating assumption #3 (parallel context):*** **the subproblems can be solved with some concurrency**

Remarks on definition

- **"Divide and conquer" is not a fully satisfactory description**
  - **"divide, conquer, and *combine*" is better**
  - **combination is often through iterative means**
- **True "divide-and-conquer" (only) algorithms are rare in computing (unfortunately)**
- **It might be preferable to focus on "subdomain composition" rather than "domain decomposition"**

We often think we know all about "two" because two is "one and one". We forget that we have to make a study of "and."

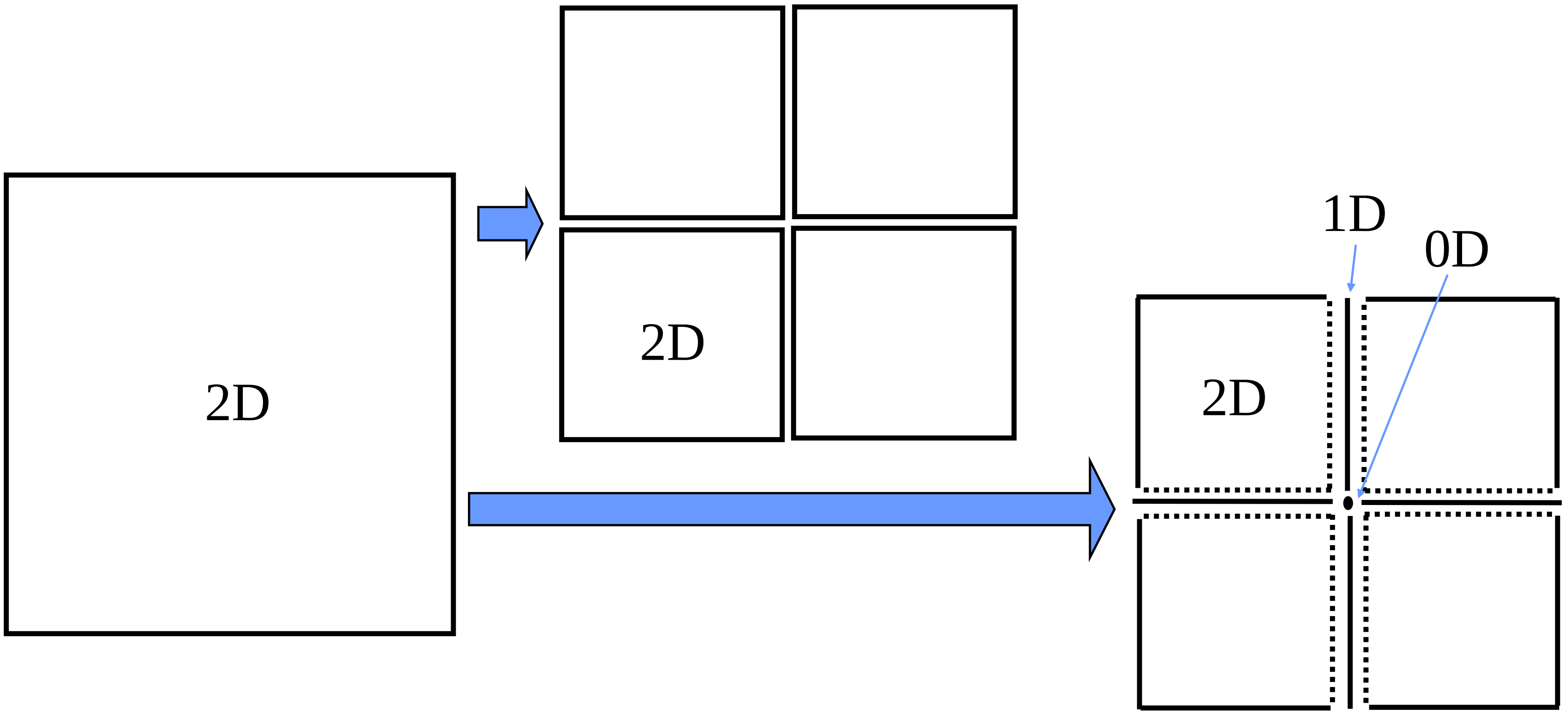
A. S. Eddington (1882-1944)

Remarks on definition

- **Domain decomposition has generic and specific senses within the universe of parallel computing**
  - **generic sense: any data decomposition (considered in contrast to task decomposition)**
  - **specific sense: the domain is the domain of definition of an operator equation (differential, integral, algebraic)**
- **In a generic sense the process of constructing a parallel program consists of**
  - **Decomposition into tasks**
  - **Assignment of tasks to processes**
  - **Orchestration of processes**
    - **Communication and synchronization**
  - **Mapping of processes to processors**

Subproblem structure

- **The subdomains may be of the same or different dimensionality as the original**

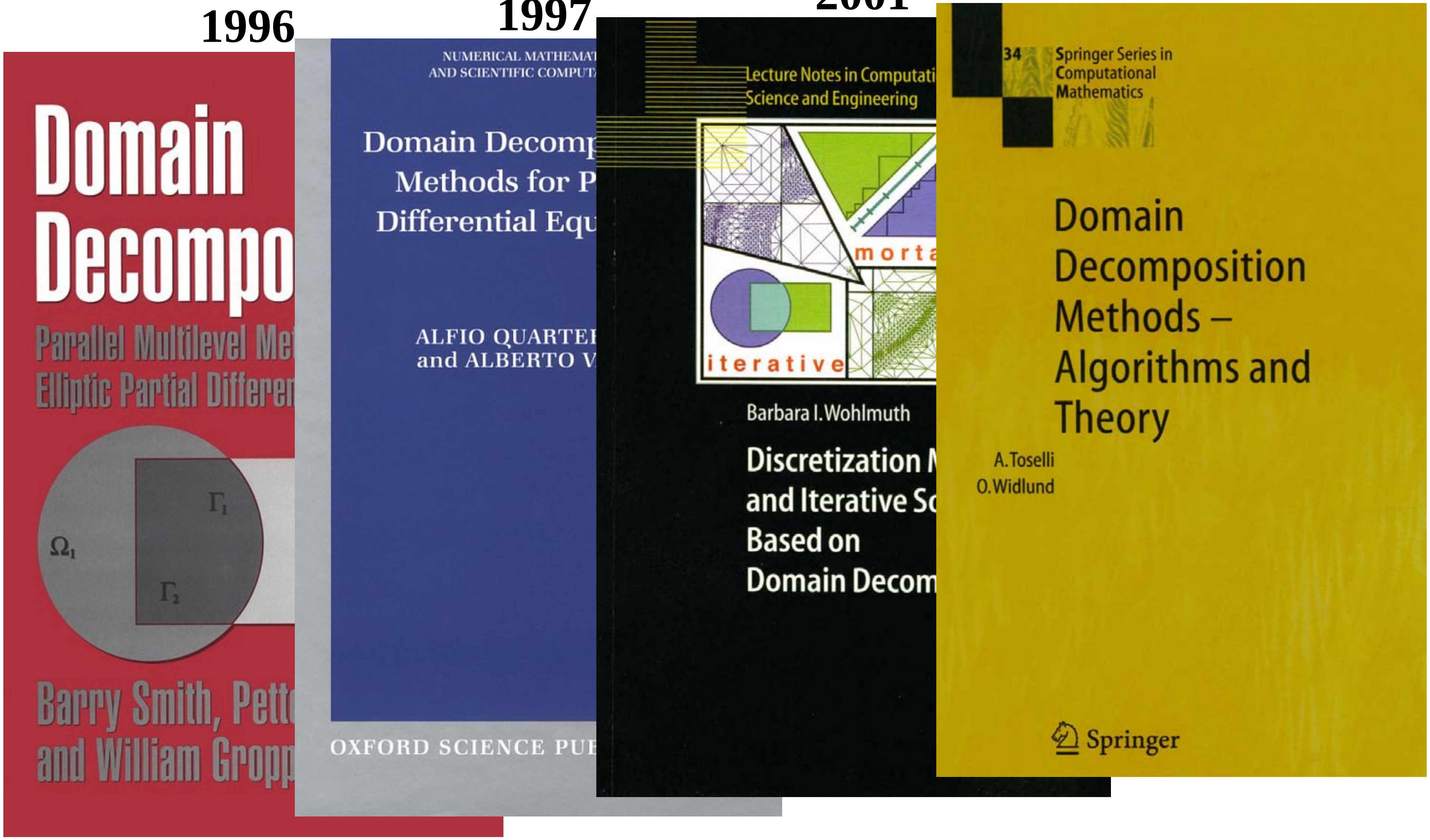


Plan of presentation

- Imperative of domain decomposition (DD) for terascale computing
- Basic DD algorithmic concepts
  - Schwarz
  - Schur
  - Schwarz-Schur combinations
- Basic DD convergence and scaling properties
- En route:
  - mention some “high watermarks” for DD-based simulations

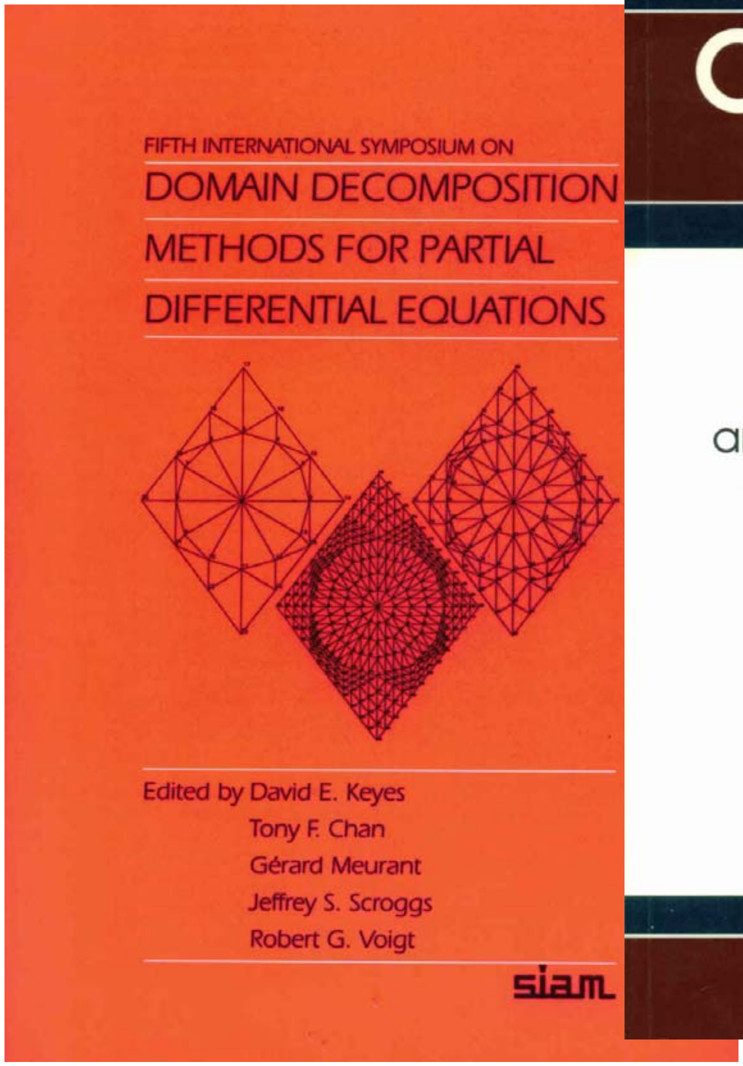
Prime sources for domain decomposition

1996 1997 2001 2004



Other sources for domain decomposition

1992

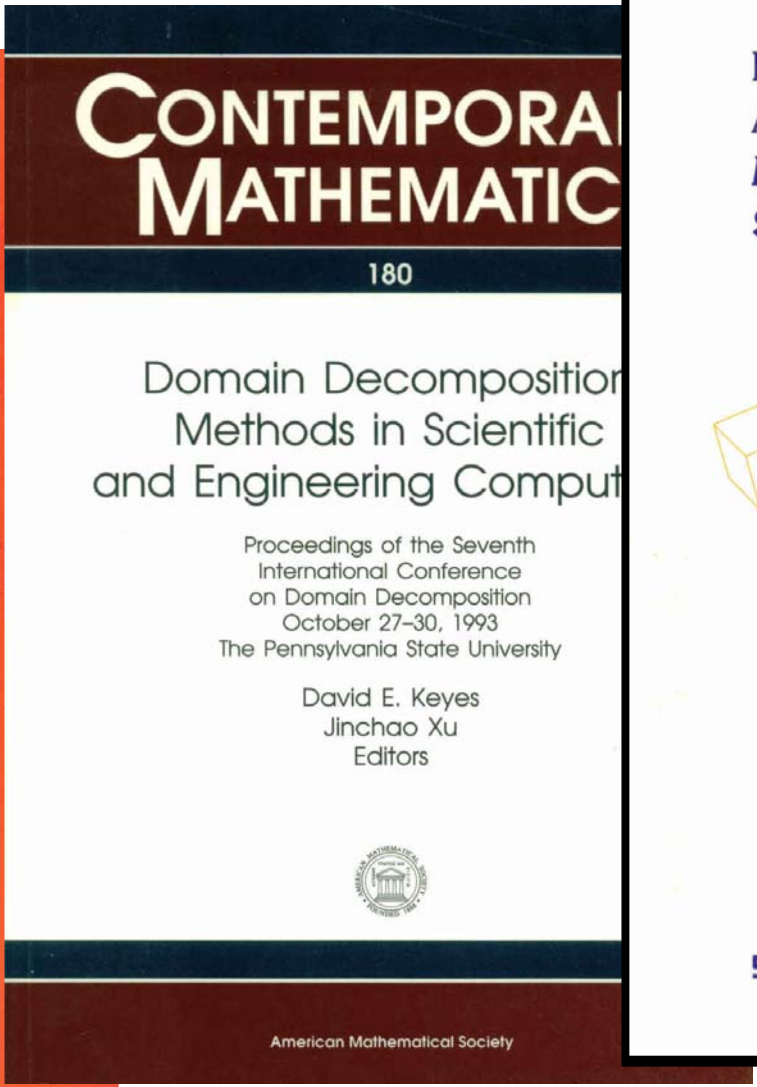


FIFTH INTERNATIONAL SYMPOSIUM ON
DOMAIN DECOMPOSITION
METHODS FOR PARTIAL
DIFFERENTIAL EQUATIONS

Edited by David E. Keyes
Tony F. Chan
Gérard Meurant
Jeffrey S. Scroggs
Robert G. Voigt

siam

1994



CONTEMPORAI
MATHEMATIC

180

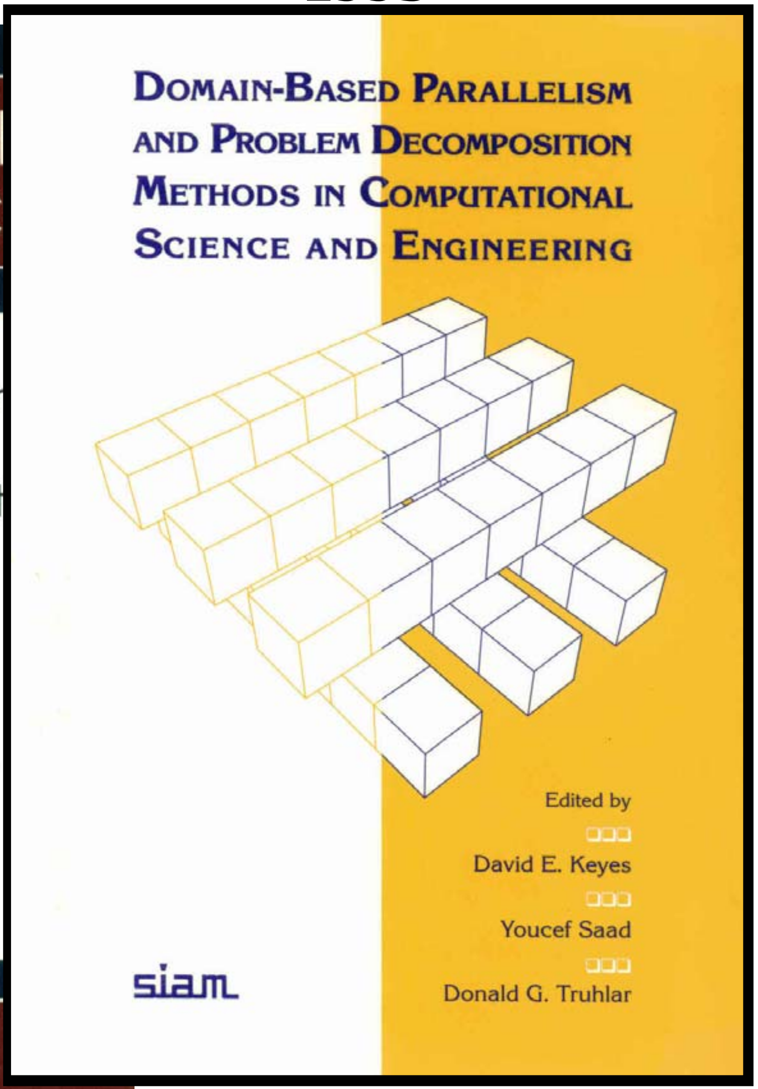
Domain Decompositior
Methods in Scientific
and Engineering Comput

Proceedings of the Seventh
International Conference
on Domain Decomposition
October 27–30, 1993
The Pennsylvania State University

David E. Keyes
Jinchao Xu
Editors

American Mathematical Society

1995



DOMAIN-BASED PARALLELISM
AND PROBLEM DECOMPOSITION
METHODS IN COMPUTATIONAL
SCIENCE AND ENGINEERING

Edited by
David E. Keyes
Youcef Saad
Donald G. Truhlar

siam

+ DDM.ORG and other proceedings volumes, 1988-2004

Platforms for high-end simulation

- ASCI roadmap: go to 100 Teraflop/s by 2005
- Use variety of vendors
  - Compaq
  - Cray
  - Intel
  - IBM
  - SGI
- Rely on commodity processor/memory units, with tightly coupled network
- Massive software project to rewrite physics codes for distributed shared memory??

NPACI Blue Horizon

ASCI White

ASCI Q

ASCI Blue Mountain

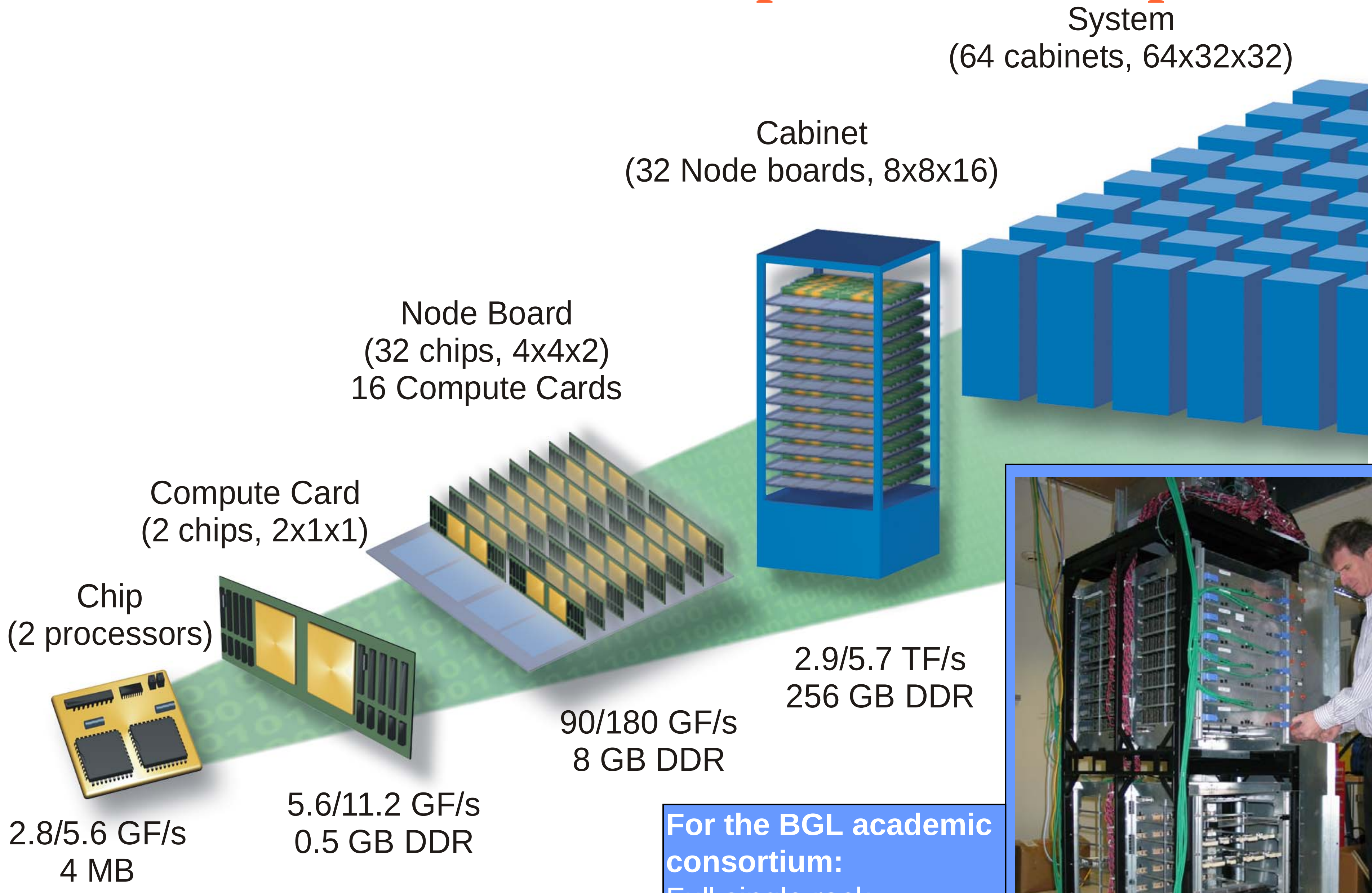
ASCI Cplant

ASCI Blue-Pacific

ASCI Whitecap

ASCI Red

IBM's BlueGene/L, 64K procs, 180 Tflop/s

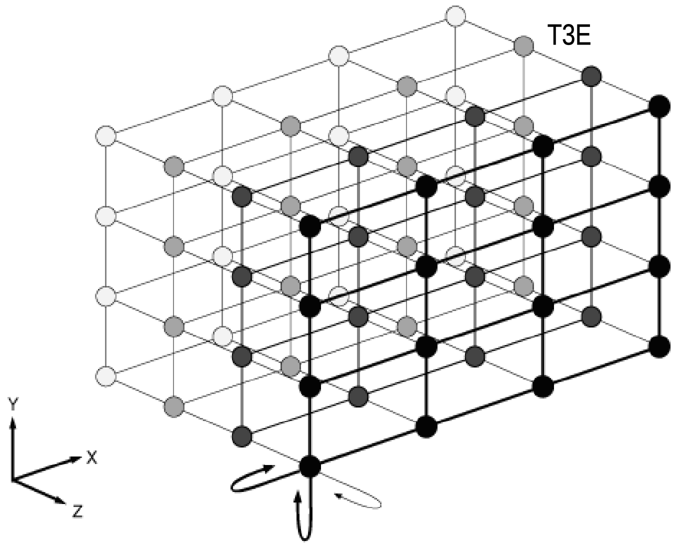


For the BGL academic consortium:
Full single rack
4.8 TFlop/s peak
70% of peak LINPACK
$2M per rack

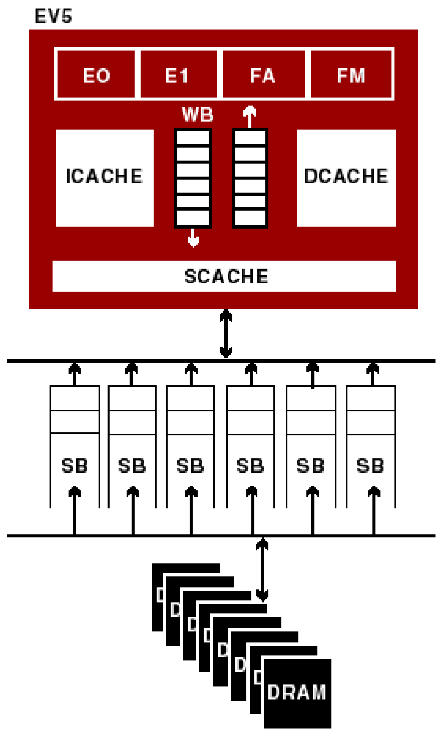
Algorithmic requirements from architecture

- **Must run on physically distributed memory units connected by message-passing network, each serving one or more processors with multiple levels of cache**

"horizontal" aspects



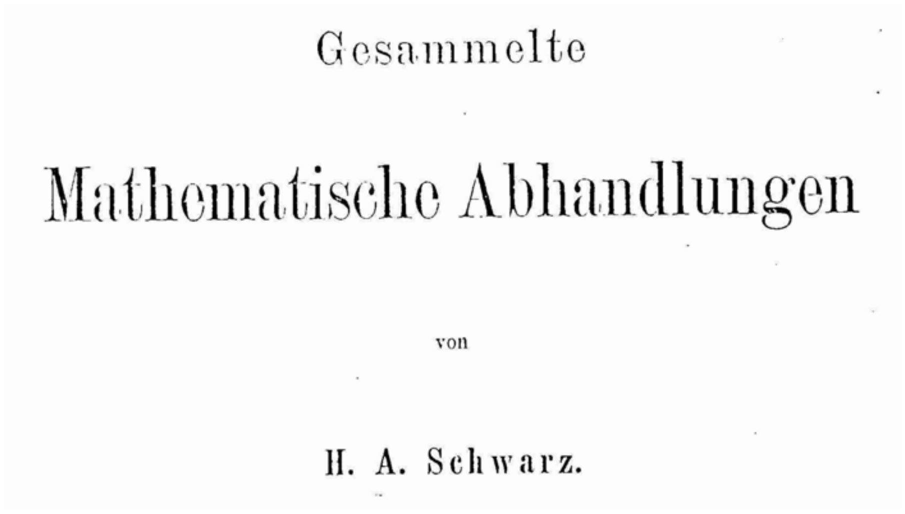
"vertical" aspects



Building platforms is the “easy” part

- Algorithms must be
  - highly concurrent and straightforward to load balance
  - latency tolerant
  - cache friendly (good temporal and spatial locality)
  - highly scalable (in the sense of convergence)
- Domain decomposition “natural” for all of these
- Domain decomposition also “natural” for software engineering
- Fortunate that its theory was built in advance of requirements!

The earliest DD paper?

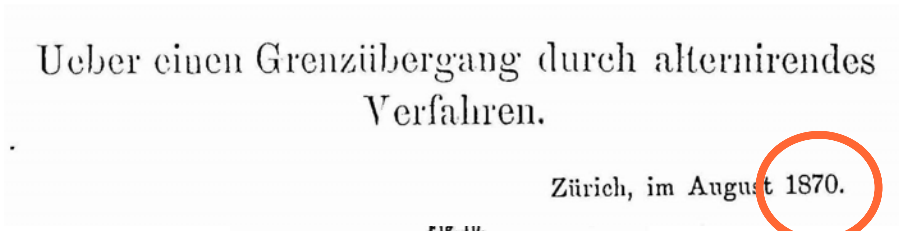


Gesammelte

Mathematische Abhandlungen

von

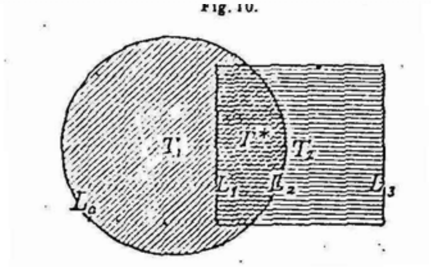
H. A. Schwarz.



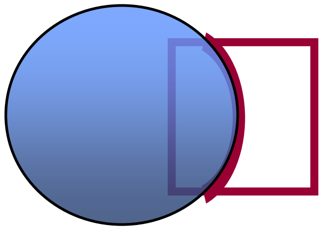
Ueber einen Grenzübergang durch alternirendes Verfahren.

Zürich, im August 1870.

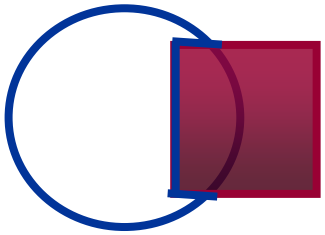
Fig. 10.



What Schwarz proposed…



Solve PDE in circle with BC taken from interior of square



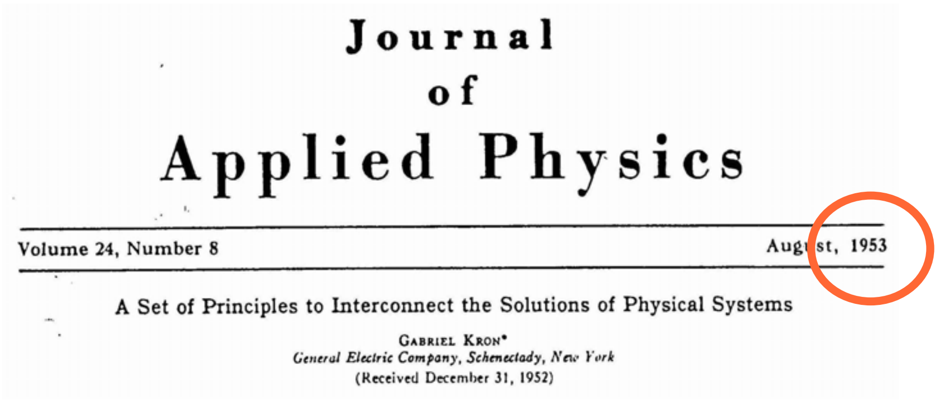
Solve PDE in square with BC taken from interior of circle

And iterate!

Rationale

- Convenient analytic means (separation of variables) are available for the regular problems in the subdomains, but not for the irregular “keyhole” problem defined by their union
- Schwarz iteration defines a functional map from the values defined along (either) artificial interior boundary segment completing a subdomain (arc or segments) to an updated set of values
- A *contraction map* is derived for the error
- Rate of convergence is not necessarily rapid – this was not a concern of Schwarz
- Subproblems are not solved concurrently – neither was this Schwarz’ concern

Other early DD papers

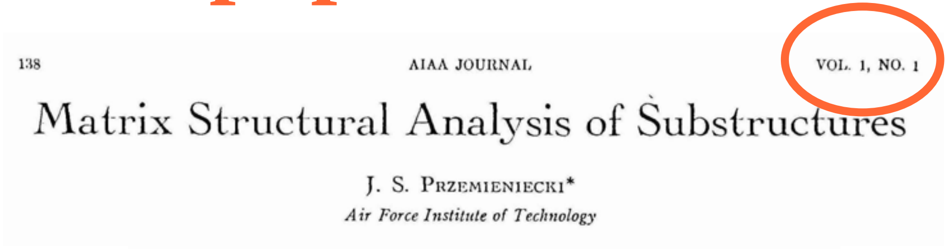


Journal of Applied Physics

Volume 24, Number 8 August, 1953

A Set of Principles to Interconnect the Solutions of Physical Systems

Gabriel Kron*
General Electric Company, Schenectady, New York
(Received December 31, 1952)



138 AIAA JOURNAL VOL. 1, NO. 1

Matrix Structural Analysis of Substructures

J. S. Przemieniecki*
Air Force Institute of Technology

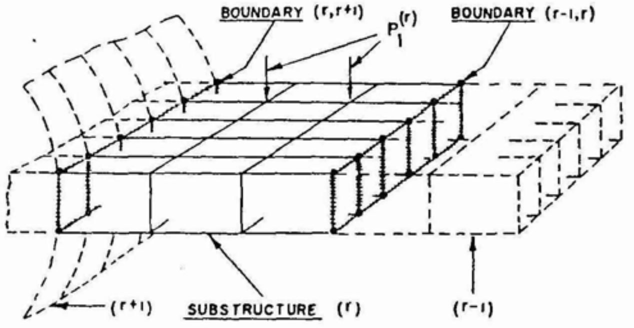
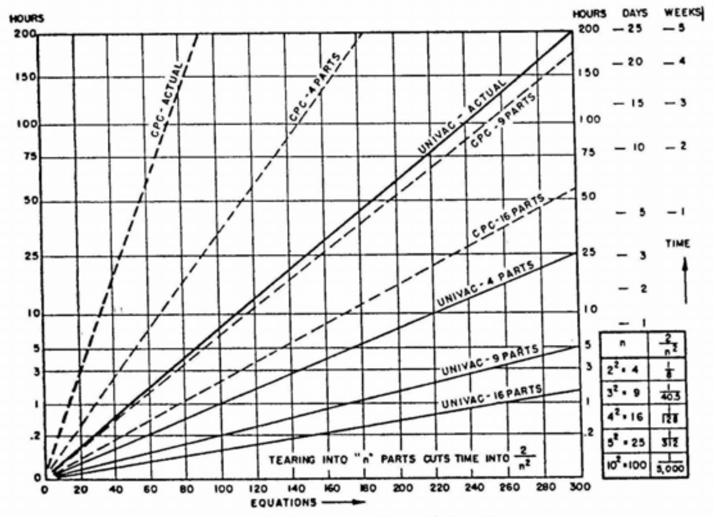


Fig. 1. Typical substructure with fixed boundaries.



n	$\frac{2}{n^2}$
$2^2 = 4$	$\frac{1}{8}$
$3^2 = 9$	$\frac{1}{40.5}$
$4^2 = 16$	$\frac{1}{128}$
$5^2 = 25$	$\frac{1}{312}$
$10^2 = 100$	$\frac{1}{5,000}$

Table I. Computer time for matrix inversion.

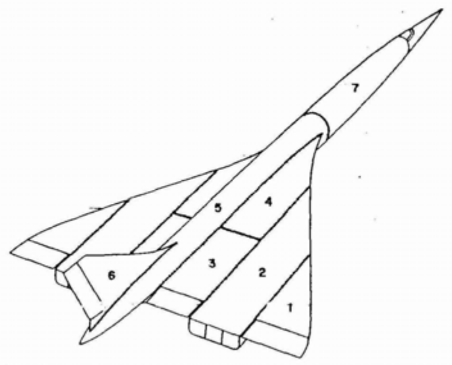


Fig. 3. Typical substructure arrangement for delta aircraft.

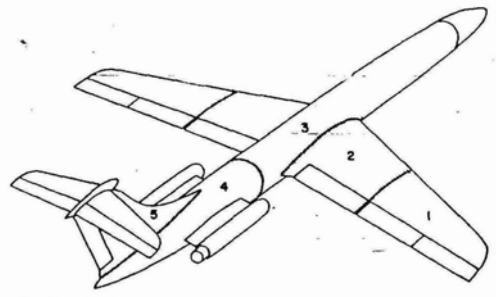


Fig. 4. Typical substructure arrangement for conventional aircraft.

Rationale

- *For Kron:* direct Gaussian elimination has superlinear complexity
  - union of subproblems and the connecting problem (each also superlinear) could be solved in fewer overall operations than one large problem
- *For Przemieniecki:* full airplane structural analysis would not fit in memory of available computers
  - individual subproblems fit in memory

Rationale

- **Let problem size be N, number of subdomains be P, and memory capacity be M**
- **Let problem solution complexity be N^a, $(a \geq 1)$**
- **Then subproblem solution complexity is $(N/P)^a$**
- **Let the cost of connecting the subproblems be $c(N,P)$**
- **Kron wins if** $P\,(N/P)^a + c(N,P) < N^a$

  or $c(N,P) < N^a\,(1-P^{1-a})$
- **Przemieniecki wins if**

$$(N/P) < M$$

NB: Kron does not win directly if a=**1** !

Contemporary interest

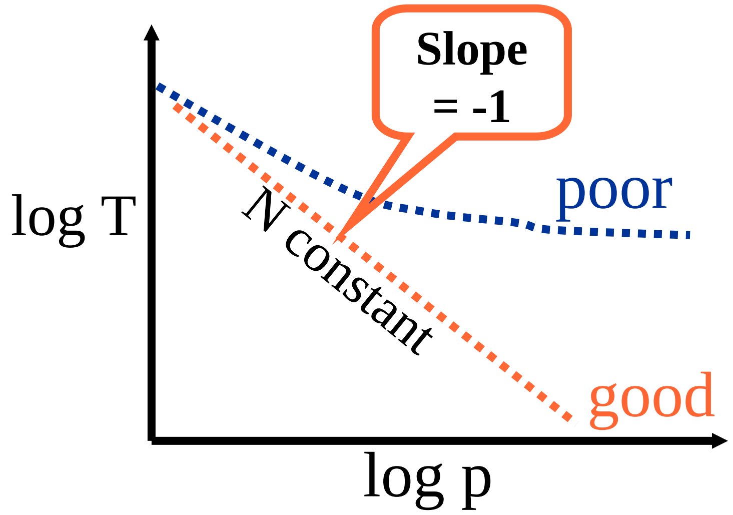
- **Goal is algorithmic scalability:**

  fill up memory of arbitrarily large machines to increase resolution, *while preserving nearly constant *running times* with respect to proportionally smaller problem on one processor**

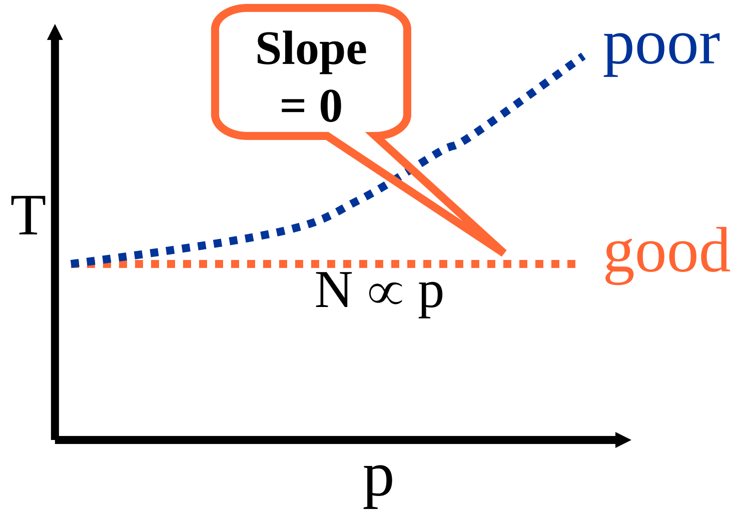
*at worst logarithmically growing

Two definitions of scalability

- **"Strong scaling"**
  - **execution time decreases in inverse proportion to the number of processors**
  - ***fixed size problem overall***

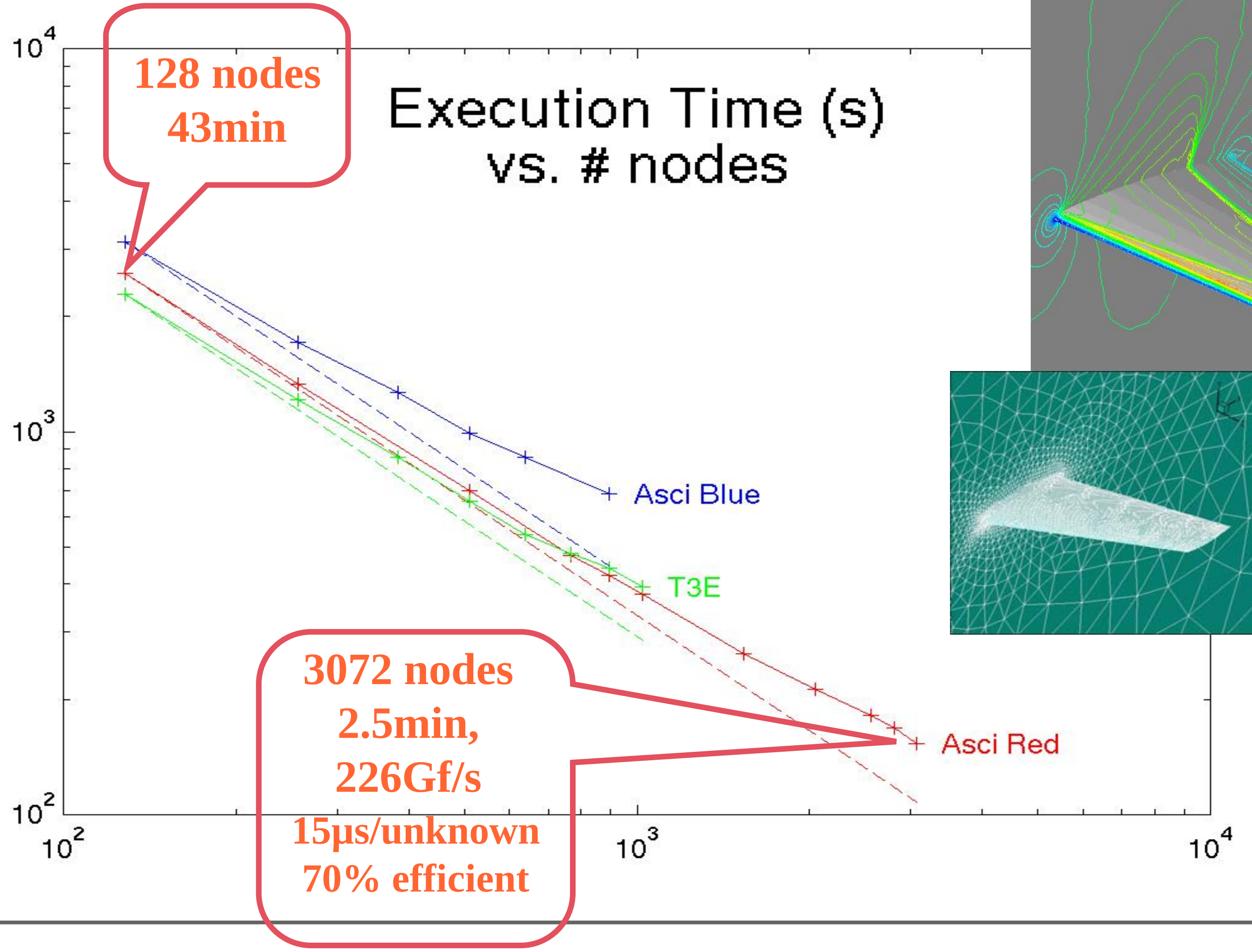


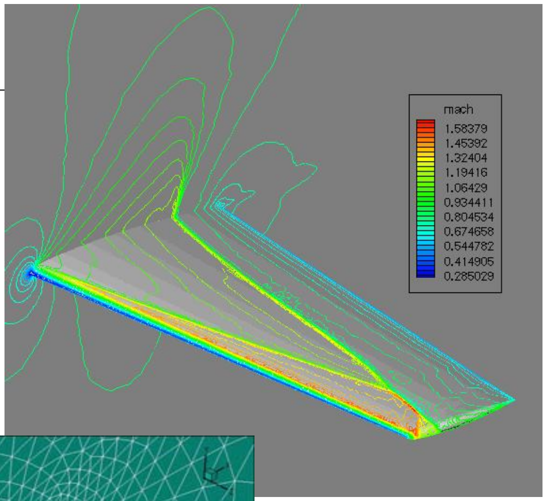
- **"Weak scaling"**
  - **execution time remains constant, as problem size and processor number are increased in proportion**
  - ***fixed size problem per processor***
  - **also known as "Gustafson scaling"**

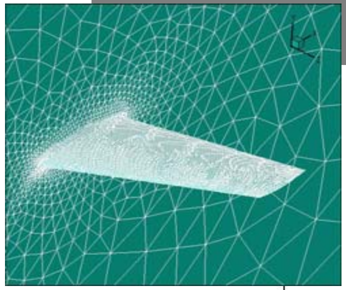


Strong scaling illus. (1999 Bell Prize)

- Newton-Krylov-Schwarz (NKS) algorithm for compressible and incompressible Euler and Navier-Stokes flows
- Used in NASA application FUN3D (M6 wing results below with 11M dof)

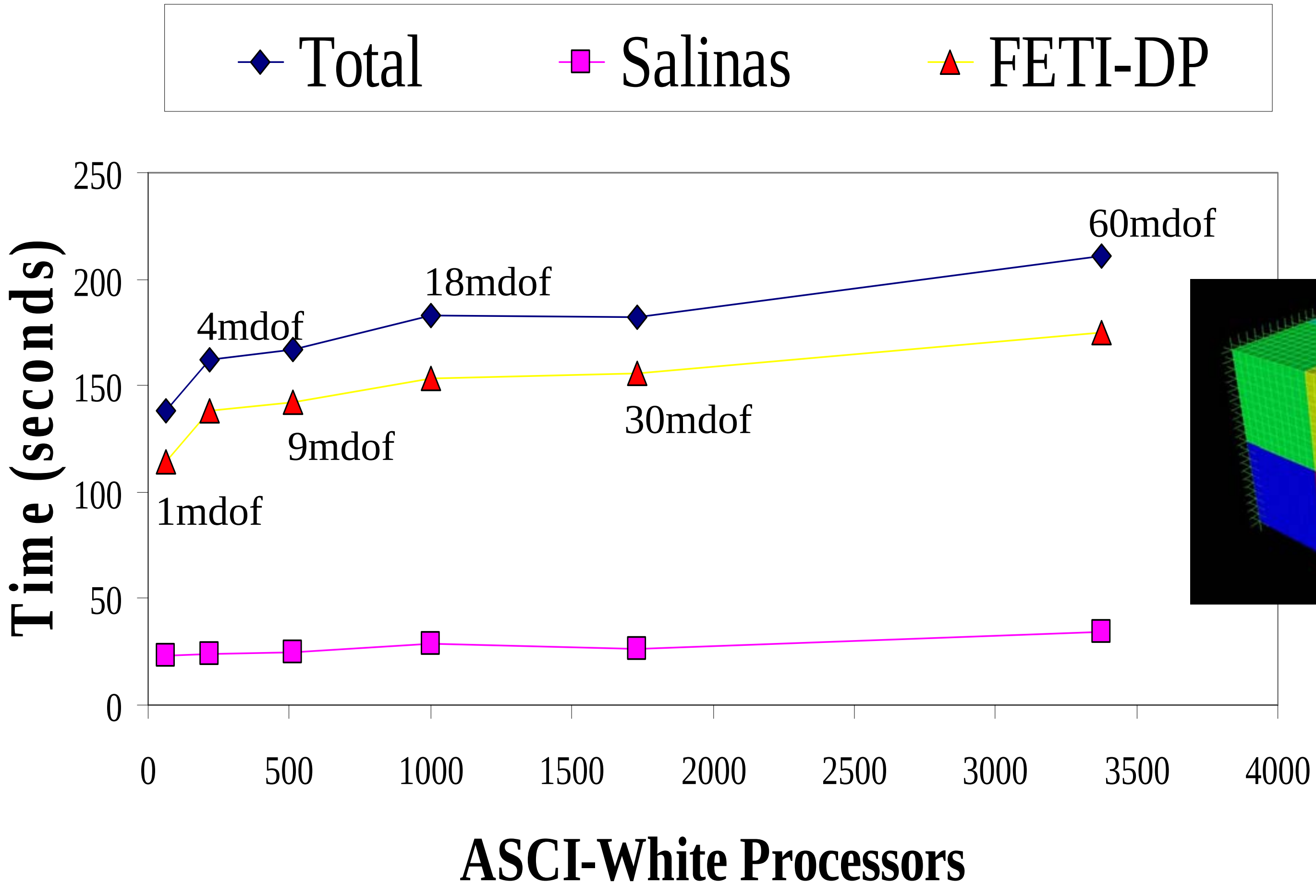


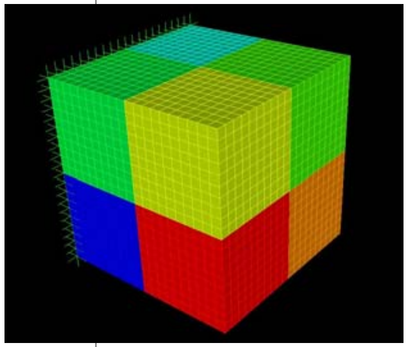




Weak scaling illus. (2002 Bell Prize)

- Finite Element Tearing and Interconnection (FETI) algorithm for solid/shell models
- Used in Sandia applications Salinas, Adagio, Andante





c/o C. Farhat and K. Pierson

Decomposition strategies for $\mathcal{L}u=f$ in Ω

- **Operator decomposition**

$$\mathcal{L} = \sum_k \mathcal{L}_k$$

- **Function space decomposition**

$$f = \sum_k f_k \Phi_k , u = \sum_k u_k \Phi_k$$

- **Domain decomposition**

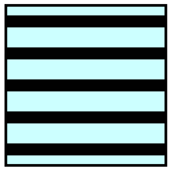
$$\Omega = \bigcup_k \Omega_k$$

Consider the implicitly discretized parabolic case

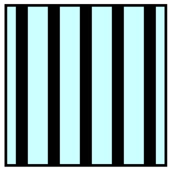
$$[\frac{I}{\tau} + \mathcal{L}_x + \mathcal{L}_y] u^{(k+1)} = \frac{I}{\tau} u^{(k)} + f$$

Operator decomposition

- Consider ADI



$$[\frac{I}{\tau/2} + \mathcal{L}_x]u^{(k+1/2)} = [\frac{I}{\tau/2} - \mathcal{L}_y]u^{(k)} + f$$



$$[\frac{I}{\tau/2} + \mathcal{L}_y]u^{(k+1)} = [\frac{I}{\tau/2} - \mathcal{L}_x]u^{(k+1/2)} + f$$

- Iteration matrix consists of four multiplicative substeps per timestep
  - two sparse matrix-vector multiplies
  - two sets of unidirectional bandsolves
- Parallelism *within* each substep
- But global data exchanges *between* bandsolve substeps

Function space decomposition

- Consider a spectral Galerkin method

$$u(x,y,t)=\sum_{j=1}^{N}a_j(t)\Phi_j(x,y)$$

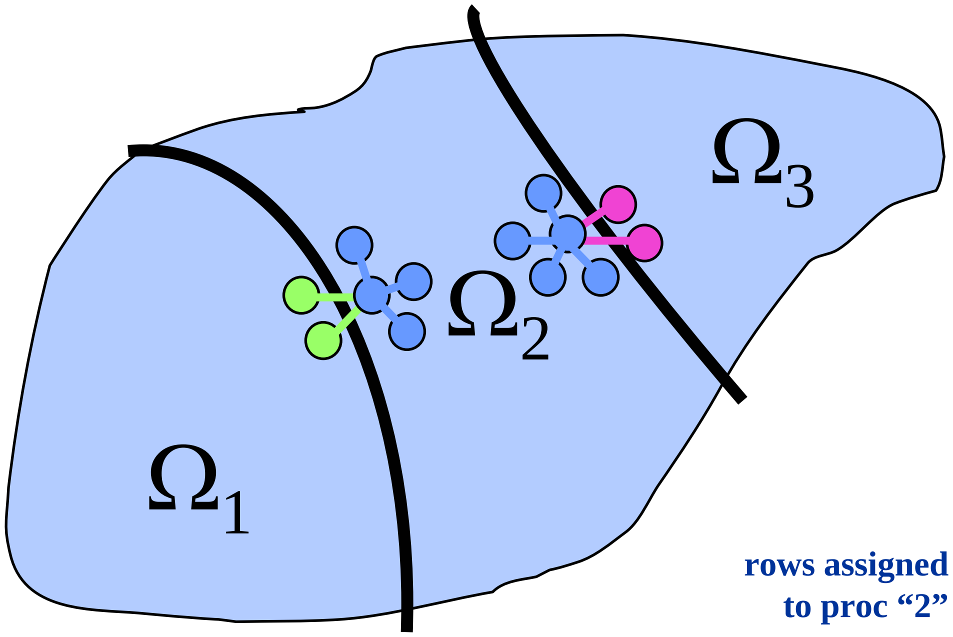
$$\tfrac{d}{dt}(\Phi_i,u)=(\Phi_i,\mathcal{L}u)+(\Phi_i,f),i=1,\ldots,N$$

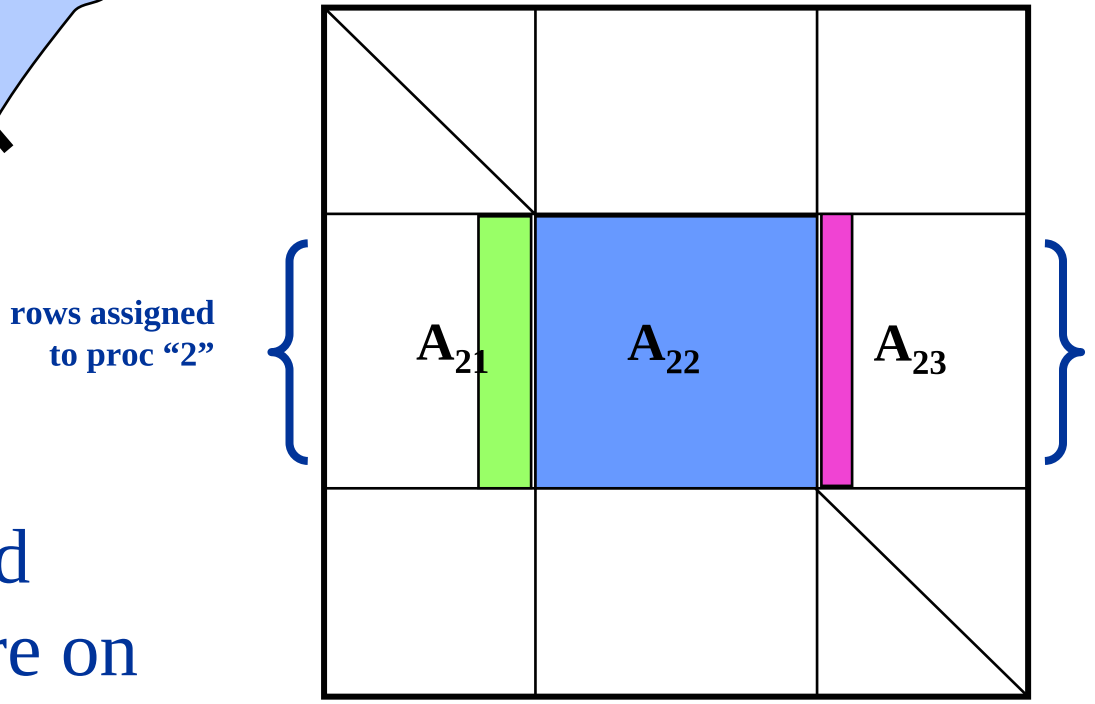
$$\sum_j(\Phi_i,\Phi_j)\tfrac{da_j}{dt}=\sum_j(\Phi_i,\mathcal{L}\Phi_j)a_j+(\Phi_i,f),i=1,\ldots,N$$

$$\tfrac{da}{dt}=M^{-1}Ka+M^{-1}f$$

- Method-of-lines system of ODEs
- Perhaps $M\equiv[(\Phi_j,\Phi_i)],K\equiv[(\Phi_j,\mathcal{L}\Phi_i)]$ are diagonal matrices
- Parallelism across spectral index
- But global data exchanges to *transform back* to physical variables at each step

SPMD parallelism w/domain decomposition

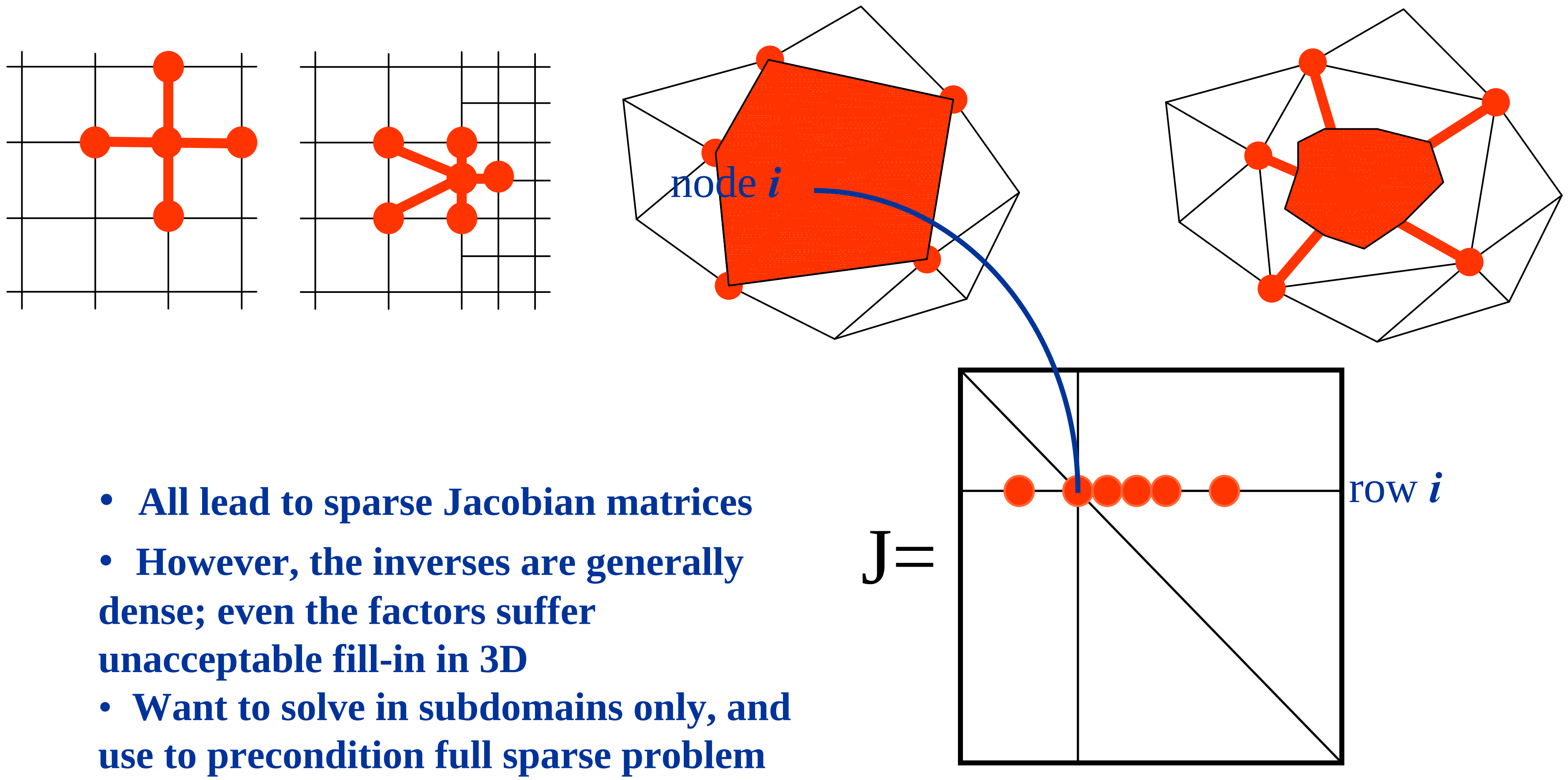




Partitioning of the grid induces block structure on the system matrix (Jacobian)

DD relevant to any local stencil formulation

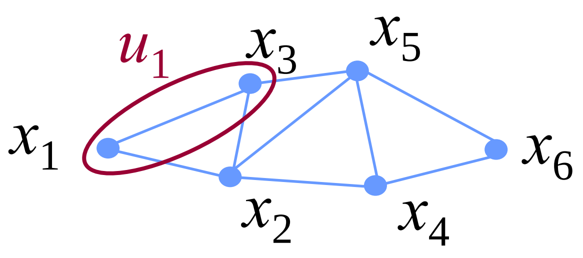
finite differences finite elements finite volumes



- All lead to sparse Jacobian matrices
- However, the inverses are generally dense; even the factors suffer unacceptable fill-in in 3D
- Want to solve in subdomains only, and use to precondition full sparse problem

Digression for notation's sake

- **We need a convenient notation for mapping vectors (representing discrete samples of a continuous field) from full domain to subdomain and back**



$$R_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

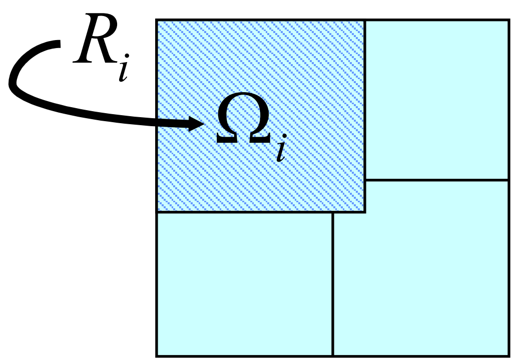
- **Let $\boldsymbol{R_i}$ be a Boolean operator that extracts the elements of the $\boldsymbol{i^{th}}$ subdomain from the global vector**

$$R_1 u = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} \equiv u_1$$

- **Then $\boldsymbol{R_i^T}$ maps the elements of the $\boldsymbol{i^{th}}$ subdomain back into the global vector, padding with zeros**

$$R_1^T = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad R_1^T u_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ 0 \\ x_3 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

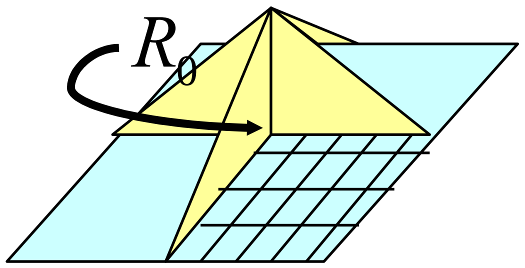
Schwarz domain decomposition method



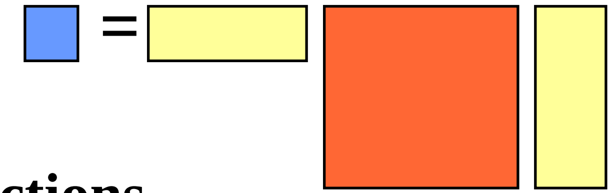
- **Consider restriction and extension operators for subdomains, R_i, R_i^T, and for possible coarse grid, R_0, R_0^T**
- **Replace discretized $Au = f$ with**

$$B^{-1}Au = B^{-1}f$$

$$B^{-1} = R_0^T A_0^{-1} R_0 + \sum_i R_i^T A_i^{-1} R_i$$



$$A_i = R_i A R_i^T$$



- **Solve by a Krylov method**
- **Matrix-vector multiplies with**
  - **parallelism on each subdomain**
  - **nearest-neighbor exchanges, global reductions**
  - **possible small global system (not needed for parabolic case)**

Krylov bases for sparse systems

- E.g., conjugate gradients (CG) for symmetric, positive definite systems, and generalized minimal residual (GMRES) for nonsymmetry or indefiniteness
- Krylov iteration is an algebraic projection method for converting a high-dimensional linear system into a lower-dimensional linear system

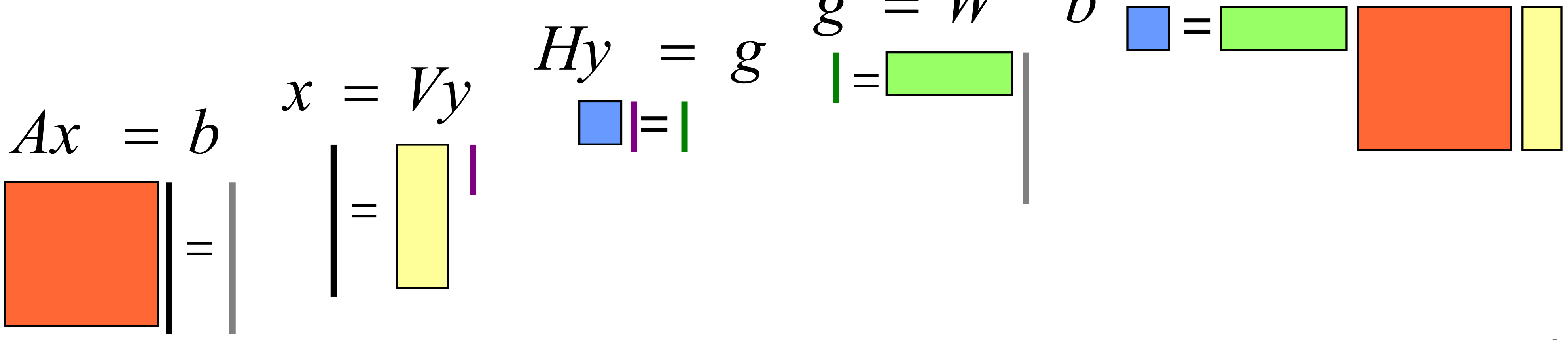
$$H \equiv W^T A V$$

$$g = W^T b$$

$$Hy = g$$

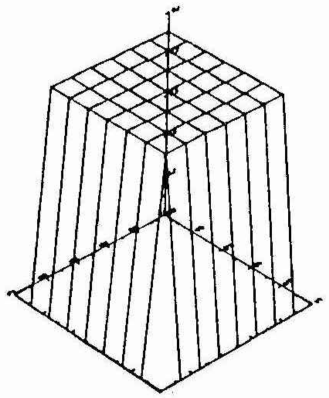
$$x = Vy$$

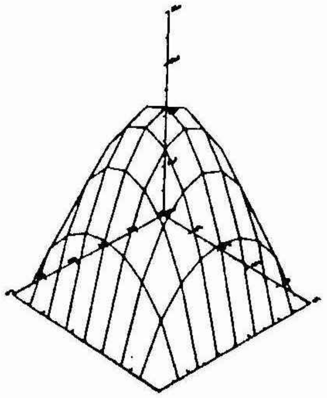
$$Ax = b$$



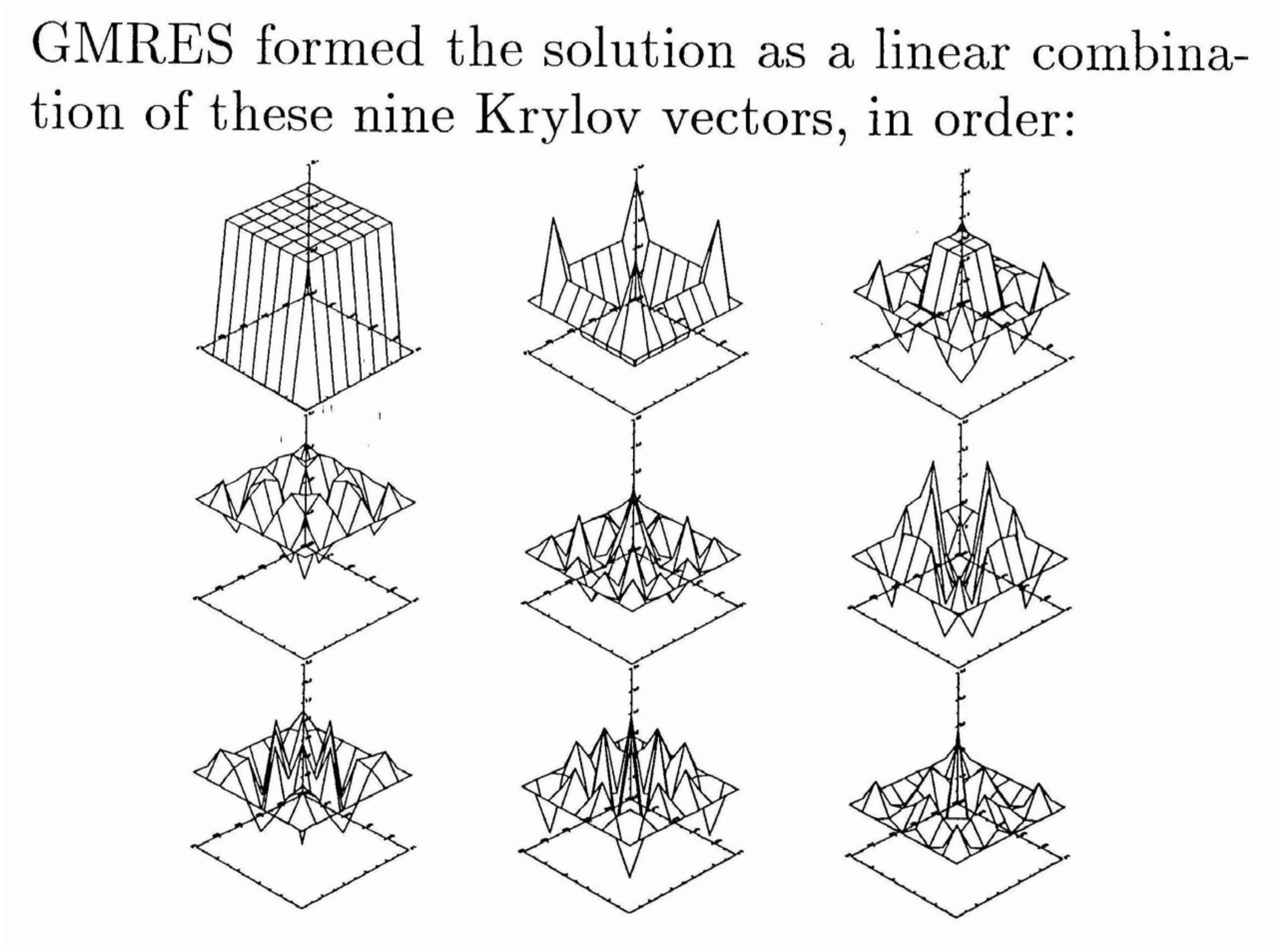
Krylov bases for sparse systems

For an easy-to-visualize example of a Krylov basis, GMRES was used to solve Poisson's equation on an 8×8 grid on a unit square with no preconditioning. The right-hand side and the solution were:

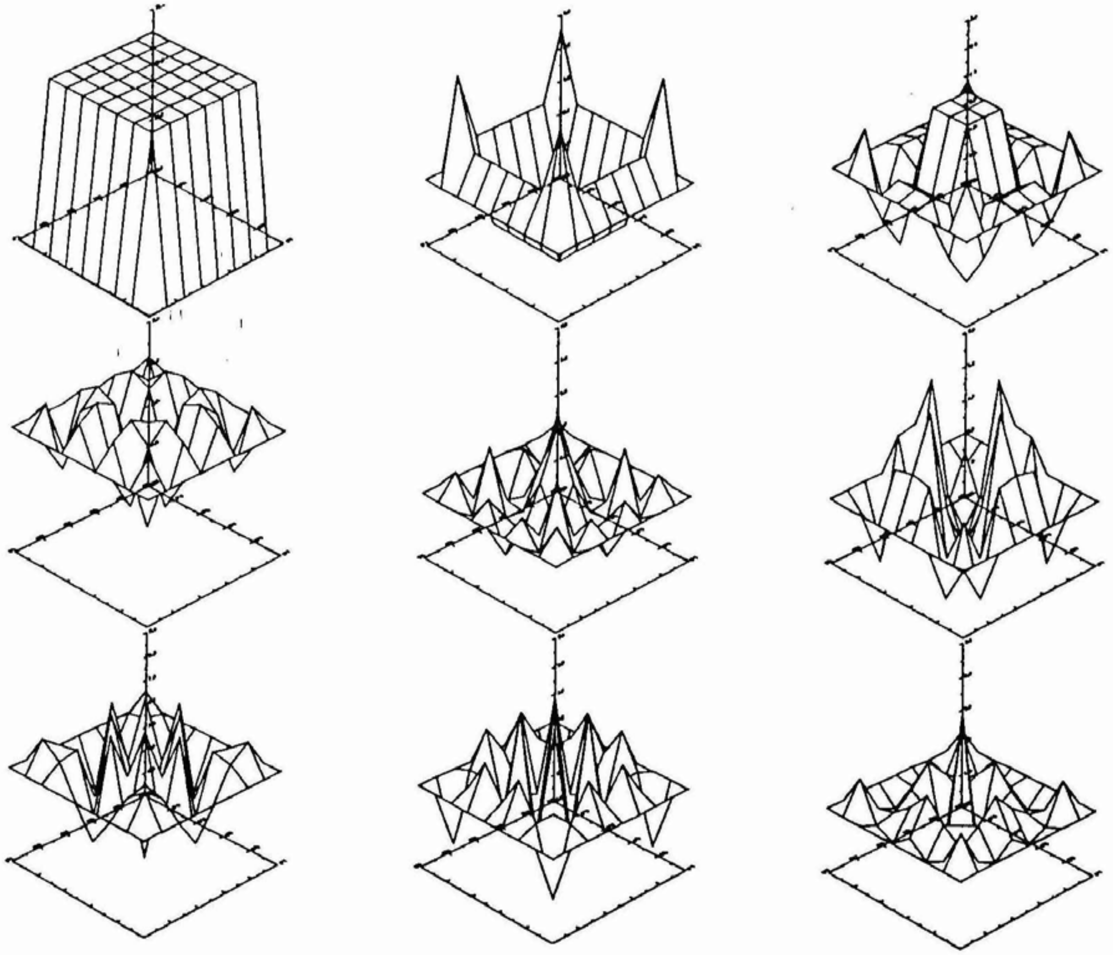




Krylov bases, cont.

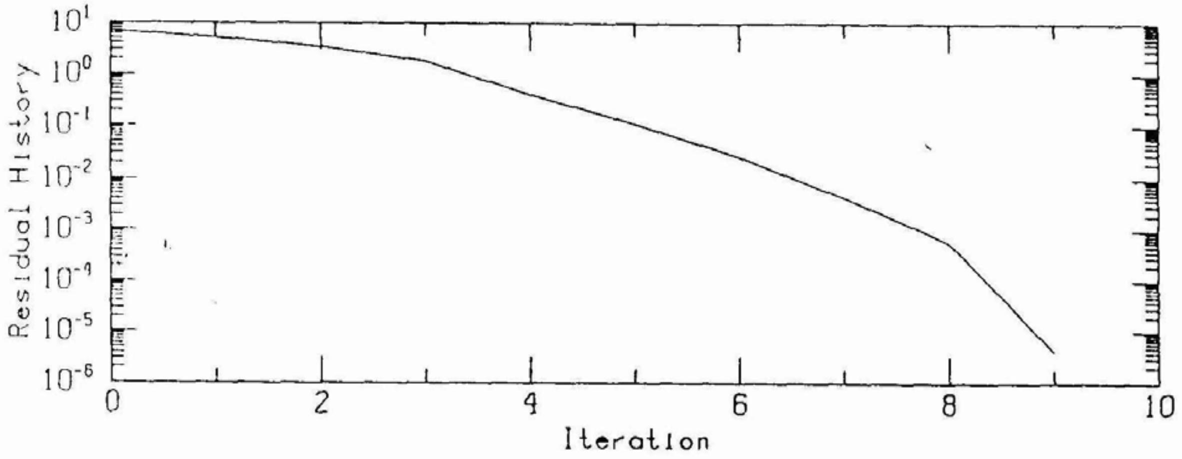


GMRES formed the solution as a linear combination of these nine Krylov vectors, in order:

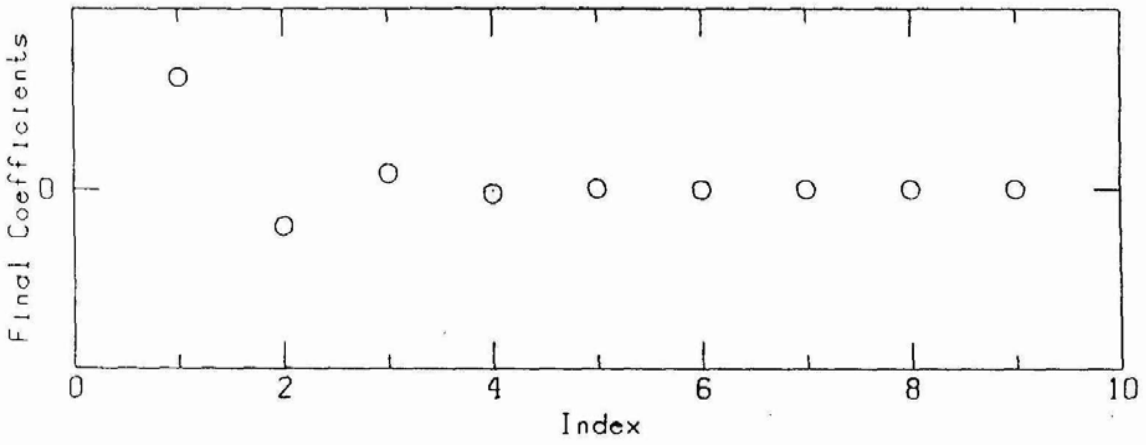


Krylov bases, cont.

The iteration history is:

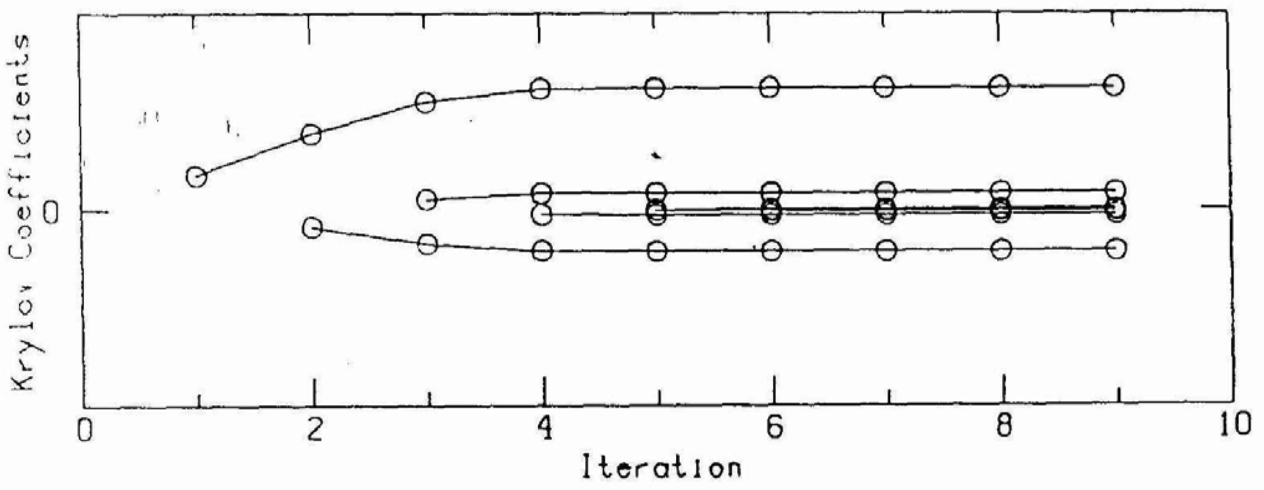


The converged magnitudes of the coefficients are:



Krylov bases, cont.

The iteration history of each coefficient is:



Note that only 9 vectors were needed in this problem of 81 degrees of freedom.

Remember this formula of Schwarz …

For a "good" approximation, B^{-1}, to A^{-1}:

$$B^{-1} = \sum_i R_i^T (R_i A R_i^T)^{-1} R_i$$

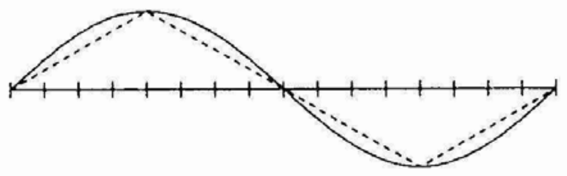
Now, let's compare!

- **Operator decomposition (ADI)**
  - **natural row-based assignment requires *global all-to-all, bulk* data exchanges in each step (for transpose)**
- **Function space decomposition (Fourier)**
  - **Natural mode-based assignment requires *global all-to-all, bulk* data exchanges in each step (for transform)**
- **Domain decomposition (Schwarz)**
  - **Natural domain-based assignment requires *local surface* data exchanges, *global reductions*, and *optional small global* problem**

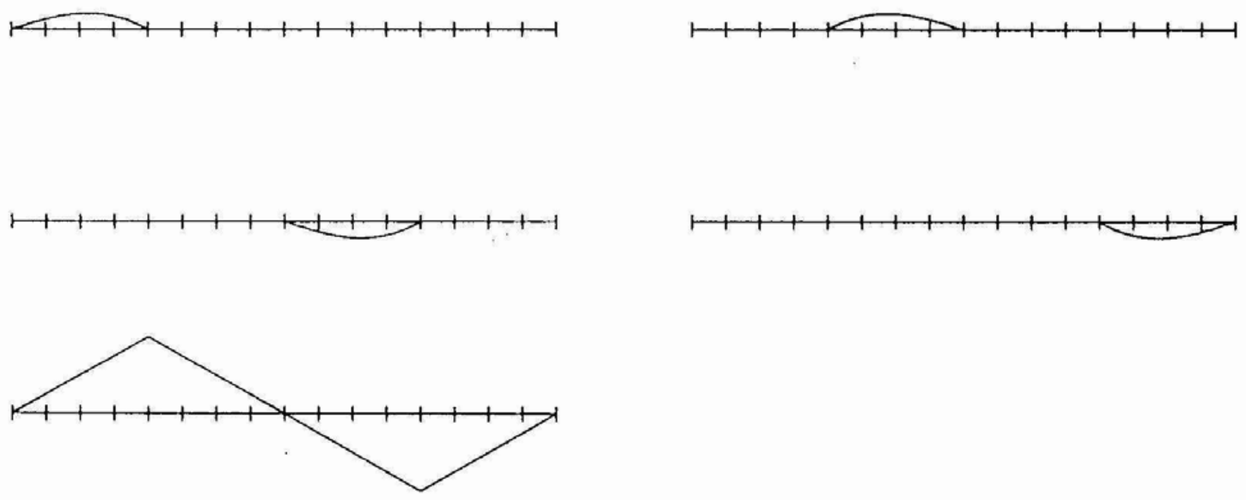
(Of course, domain decomposition can be interpreted as a *special* operator or function space decomposition)

Schwarz subspace decomposition

Consider a one-dimensional example. The function $u(x)$ sketched below

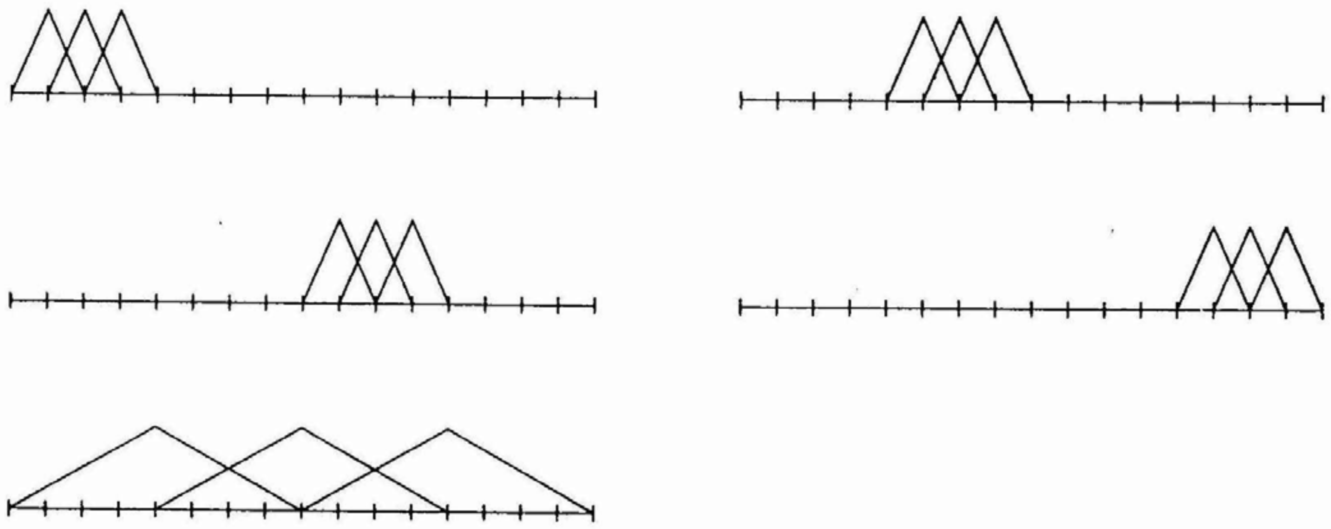


can be decomposed into the sum of the following five functions:



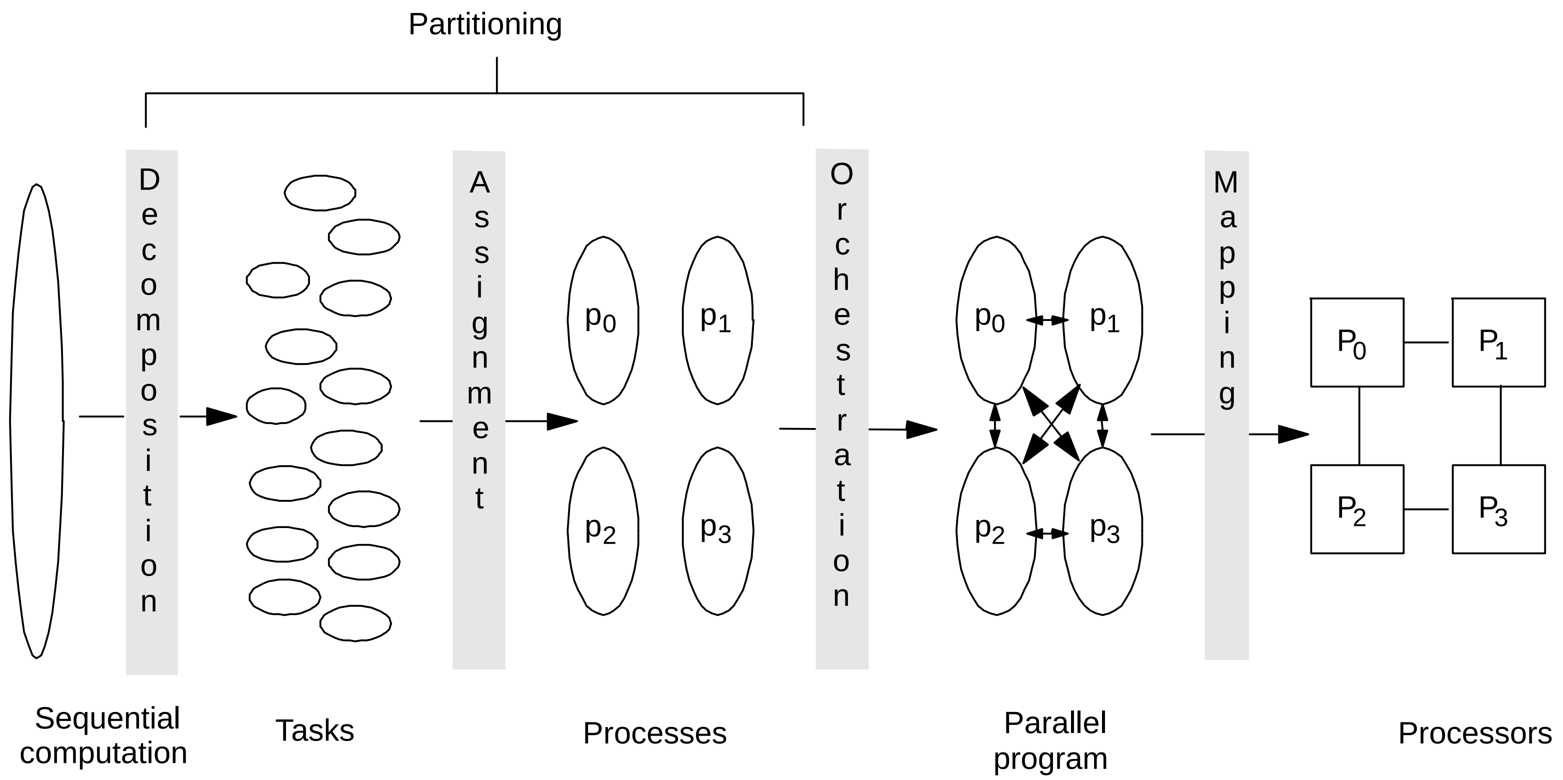
Schwarz subspace decomposition

Piecewise linear finite element bases for each of the five functions are shown below:



The first four of these subspaces are **mutually orthogonal**. The last one is **not orthogonal** to any of the others.

Four steps in creating a parallel program

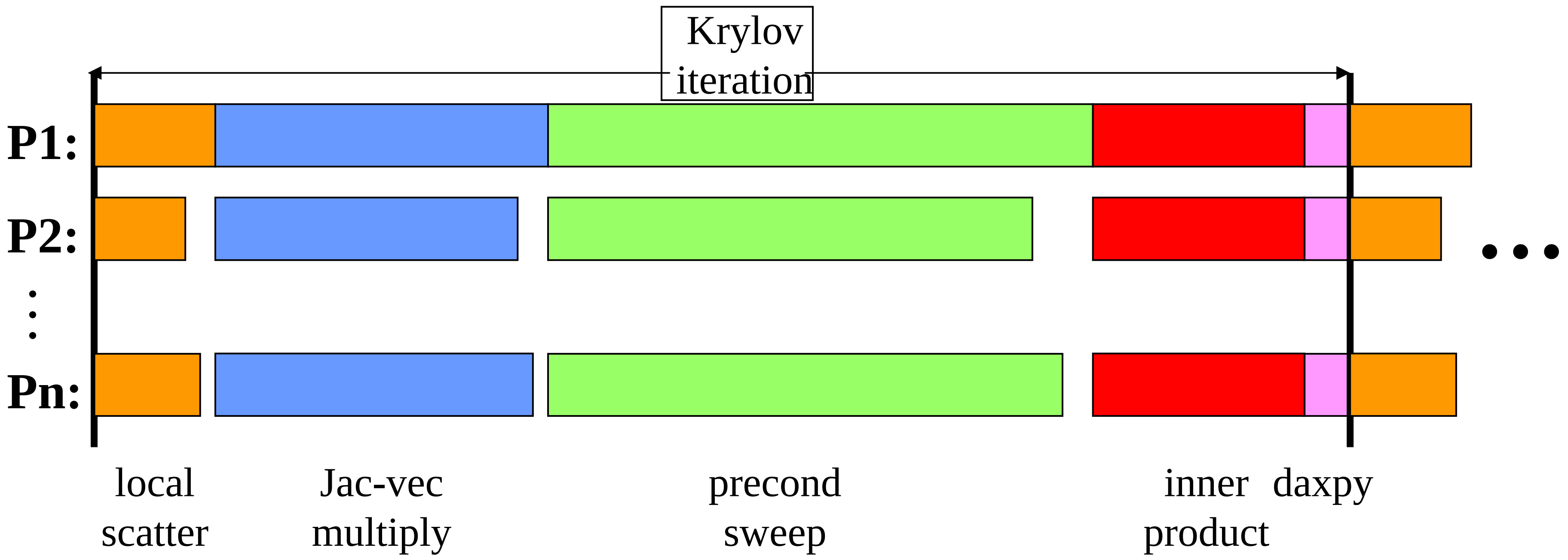


- **Decomposition of computation in tasks**
- **Assignment of tasks to processes**
- **Orchestration of data access, communication, synchronization**
- **Mapping processes to processors**

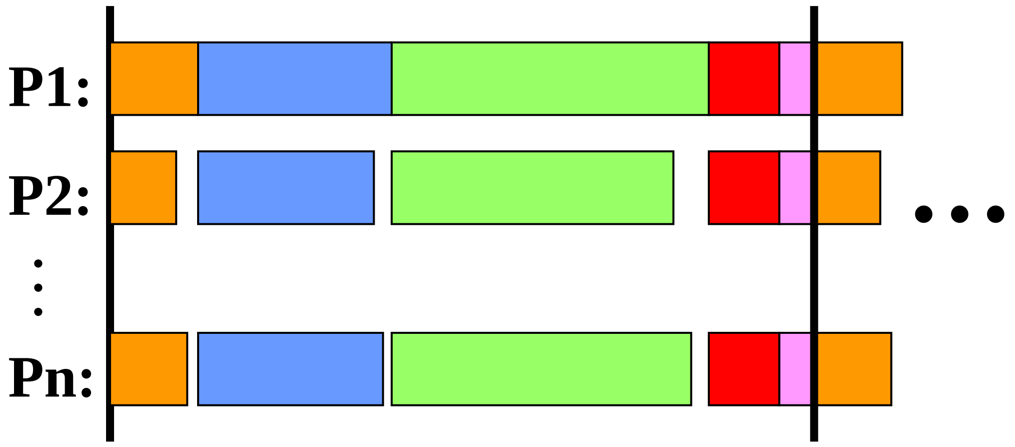
Krylov-Schwarz parallelization summary

- Decomposition into concurrent tasks
  - by domain
- Assignment of tasks to processes
  - typically one subdomain per process
- Orchestration of communication between processes
  - to perform sparse matvec – near neighbor communication
  - to perform subdomain solve – nothing
  - to build Krylov basis – global inner products
  - to construct best fit solution – global sparse solve (redundantly)
- Mapping of processes to processors
  - typically one process per processor

Krylov-Schwarz kernel in parallel

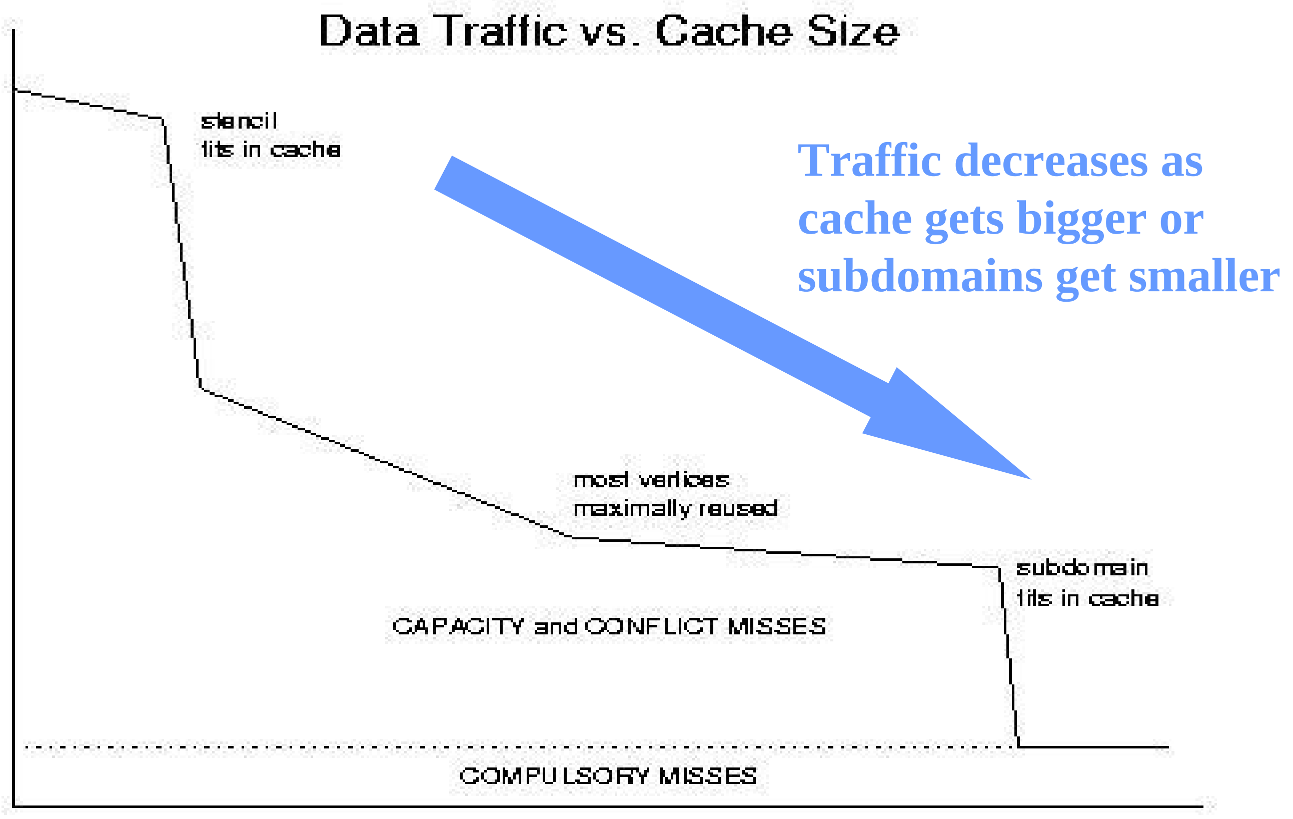


What happens if, for instance, in this (schematicized) iteration, arithmetic speed is ***doubled***, scalar all-gather is ***quartered***, and local scatter is ***cut by one-third***? Each phase is considered separately. Answer is to the right.



Krylov-Schwarz compelling in serial, too

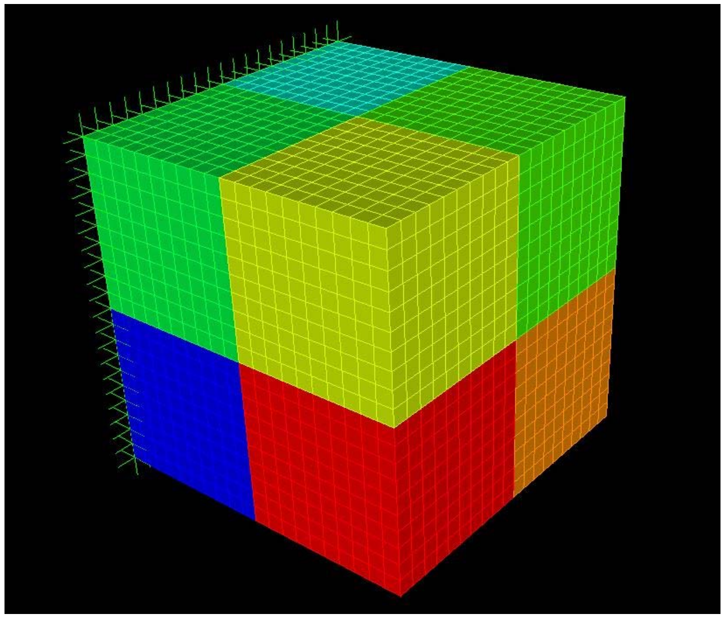
- As successive workingsets "drop" into a level of memory, capacity (and with effort conflict) misses disappear, leaving only compulsory misses, reducing demand on main memory bandwidth
- Cache size is not easily manipulated, but domain size *is*



Estimating scalability of stencil computations

- Given complexity estimates of the leading terms of:
  - the concurrent computation (per iteration phase)
  - the concurrent communication
  - the synchronization frequency
- And a bulk synchronous model of the architecture including:
  - internode communication (network topology and protocol reflecting horizontal memory structure)
  - on-node computation (effective performance parameters including vertical memory structure)
- One can estimate optimal concurrency and optimal execution time
  - on per-iteration basis, or overall (by taking into account any granularity-dependent convergence rate)
  - simply differentiate time estimate in terms of *(N,P)* with respect to *P*, equate to zero and solve for *P* in terms of *N*

Estimating 3D stencil costs (per iteration)



- grid points in each direction ***n***, total work ***N=O(n³)***
- processors in each direction ***p***, total procs ***P=O(p³)***
- memory per node requirements ***O(N/P)***

- concurrent execution time per iteration $A\, n^3/p^3$
- grid points on side of each processor subdomain n/p
- Concurrent neighbor commun. time per iteration $B\, n^2/p^2$
- cost of global reductions in each iteration $C \log p$ or $C\, p^{(1/d)}$
  - ***C*** includes synchronization frequency
- same dimensionless units for measuring ***A***, ***B***, ***C***
  - e.g., cost of scalar floating point multiply-add

3D stencil computation illustration

Rich local network, tree-based global reductions

- **total wall-clock time per iteration**

$$T(n,p) = A\frac{n^3}{p^3} + B\frac{n^2}{p^2} + C\log p$$

- **for optimal** p**,** $\frac{\partial T}{\partial p} = 0$**, or** $-3A\frac{n^3}{p^4} - 2B\frac{n^2}{p^3} + \frac{C}{p} = 0,$

or (with $\theta \equiv \frac{32B^3}{243A^2C}$ **),**

$$p_{opt} = \left(\frac{3A}{2C}\right)^{1/3}\left(\left[1+(1-\sqrt{\theta})\right]^{1/3} + \left[1-(1-\sqrt{\theta})\right]^{1/3}\right)\cdot n$$

- **without "speeddown,"** p **can grow with** n
- **in the limit as** ${B}/{C} \to 0$

$$p_{opt} = \left(\frac{3A}{C}\right)^{1/3}\cdot n$$

3D stencil computation illustration

Rich local network, tree-based global reductions

- **optimal running time**

$$T(n, p_{opt}(n)) = \frac{A}{\rho^3} + \frac{B}{\rho^2} + C\log(\rho n),$$

where

$$\rho = \left(\frac{3A}{2C}\right)^{1/3}\left(\left[1+(1-\sqrt{\theta})\right]^{1/3} + \left[1-(1-\sqrt{\theta})\right]^{1/3}\right)$$

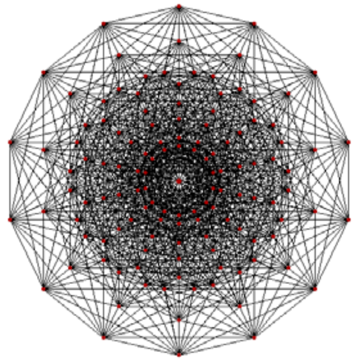
- **limit of infinite neighbor bandwidth, zero neighbor latency (** $B \to 0$ **)**

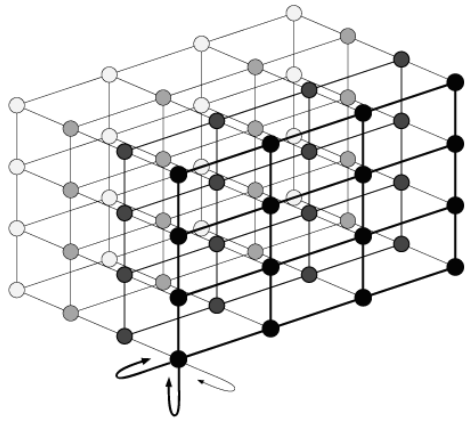
$$T(n, p_{opt}(n)) = C\left[\log n + \frac{1}{3}\log\frac{A}{C} + const.\right]$$

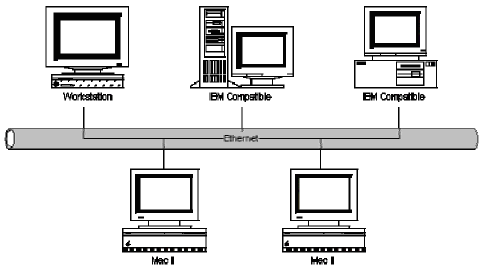
(This analysis is on a per iteration basis; complete analysis multiplies this cost by an iteration count estimate that generally depends on *n* and *p*.)

Scalability results for DD stencil computations

- **With tree-based (logarithmic) global reductions and scalable nearest neighbor hardware:**
  - **optimal number of processors scales *linearly* with problem size**
- **With 3D torus-based global reductions and scalable nearest neighbor hardware:**
  - **optimal number of processors scales as *three-fourths* power of problem size (almost "scalable")**
- **With common network bus (heavy contention):**
  - **optimal number of processors scales as *one-fourth* power of problem size (not "scalable")**



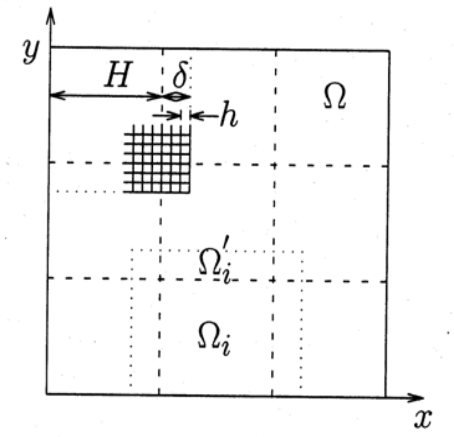




Factoring convergence rate into estimates

- Krylov-Schwarz iterative methods typically converge in a number of iterations that scales as the square-root of the condition number of the Schwarz-preconditioned system

- In terms of N and P, where for d-dimensional isotropic problems, $N=h^{-d}$ and $P=H^{-d}$, for mesh parameter h and subdomain diameter H, iteration counts may be estimated as follows:



Preconditioning Type	in 2D	in 3D
Point Jacobi	$O(N^{1/2})$	$O(N^{1/3})$
Domain Jacobi (δ=0)	$O((NP)^{1/4})$	$O((NP)^{1/6})$
1-level Additive Schwarz	$O(P^{1/2})$	$O(P^{1/3})$
2-level Additive Schwarz	$O(1)$	$O(1)$

Where do these results come from?

- **Point Jacobi result is well known (see any book on the numerical analysis of elliptic problems)**
- **Subdomain Jacobi result has interesting history**
  - **Was derived independently from functional analysis, linear algebra, and graph theory**
- **Schwarz theory is neatly and abstractly summarized in Section 5.2 Smith, Bjorstad & Gropp (1996) and Chapter 2 of Toselli & Widlund (2004)**
  - **condition number, $\kappa \leq \omega\,[1+\rho(\mathcal{E})]\,C_0^2$**
  - **C_0^2 is a splitting constant for the subspaces of the decomposition**
  - **$\rho(\mathcal{E})$ is a measure of the orthogonality of the subspaces**
  - **ω is a measure of the approximation properties of the subspace solvers (can be unity for exact subdomain solves)**
  - **These properties are estimated for different subspaces, different operators, and different subspace solvers and the "crank" is turned**

Comments on the Schwarz results

- **Original basic Schwarz estimates were for:**
  - ***self-adjoint*** **elliptic operators**
  - ***positive definite*** **operators**
  - ***exact*** **subdomain solves,** A_i^{-1}
  - ***two-way*** **overlapping with** R_i, R_i^T
  - ***generous*** **overlap,** $\delta=O(H)$ **(original 2-level result was** $O(1+H/\delta)$**)**
- **Subsequently extended to (within limits):**
  - ***nonself-adjointness*** **(e.g, convection)**
  - ***indefiniteness*** **(e.g., wave Helmholtz)**
  - ***inexact*** **subdomain solves**
  - ***one-way*** **overlap communication ("restricted additive Schwarz")**
  - ***small*** **overlap**

Comments on the Schwarz results, cont.

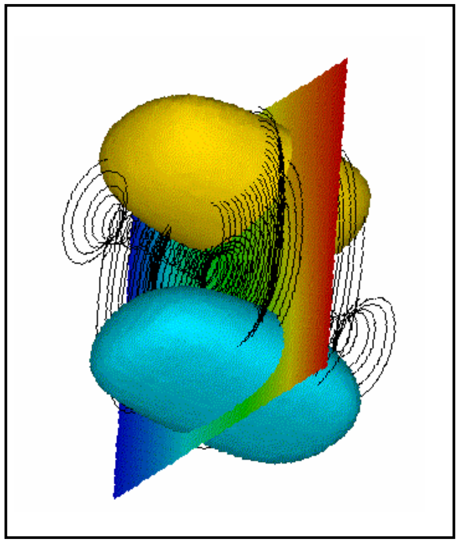
- Theory still requires "sufficiently fine" coarse mesh
  - **However, coarse space need *not* be nested in the fine space or in the decomposition into subdomains**
- Practice is better than one has any right to expect

"In theory, theory and practice are the same ...
In practice they're not!"
— *Yogi Berra*

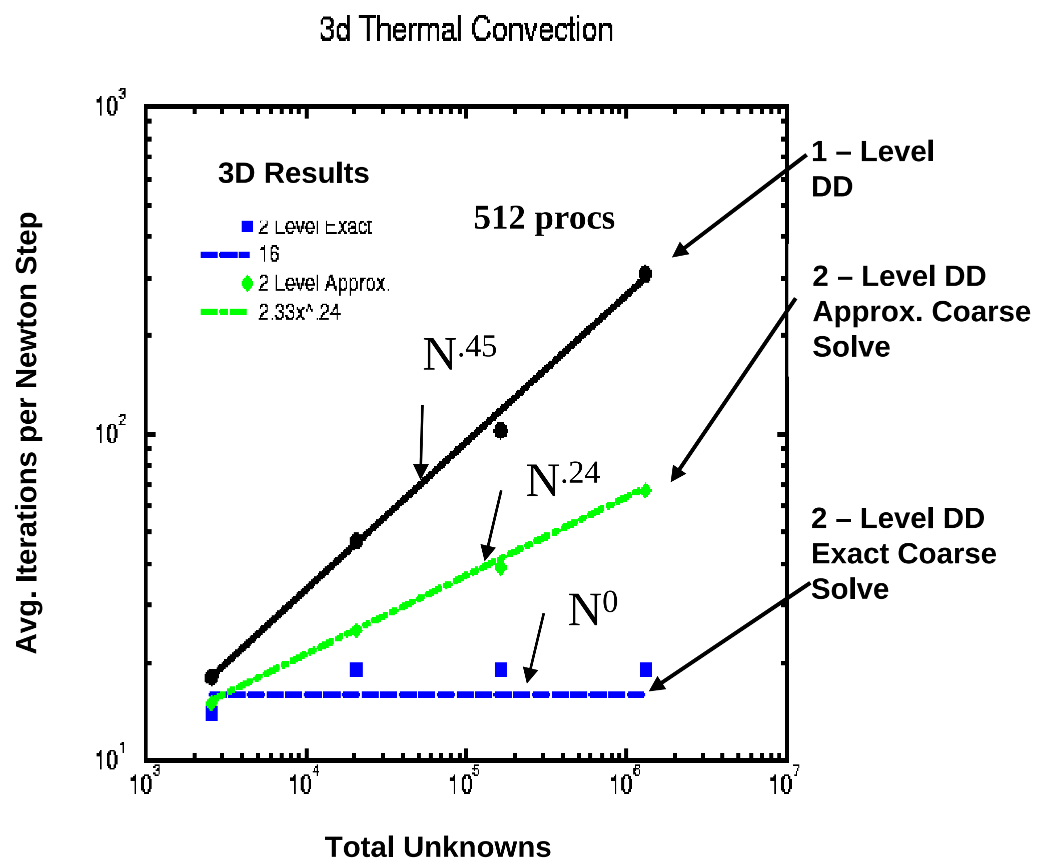
- Wave Helmholtz (e.g., acoustics) is delicate at high frequency:
  - **standard Schwarz Dirichlet boundary conditions can lead to undamped resonances within subdomains,** $u_\Gamma = 0$
  - **remedy involves Robin-type transmission boundary conditions on subdomain boundaries,** $(u + \alpha\, \partial u / \partial n)_\Gamma = 0$

Illustration of 1-level vs. 2-level tradeoff

Thermal Convection Problem (Ra = 1000)



Temperature iso-lines on slice plane, velocity iso-surfaces and streamlines in 3D



c/o J. Shadid and R. Tuminaro

Newton-Krylov solver with Aztec non-restarted GMRES with 1-level domain decomposition preconditioner, ILUT subdomain solver, and ML 2-level DD with Gauss-Seidel subdomain solver. Coarse Solver: “Exact” = SuperLU (1 proc), “Approx” = one step of ILU (8 proc. in parallel)

"Unreasonable effectiveness" of Schwarz

- **When does the sum of partial inverses equal the inverse of the sums? When the decomposition is right! Let $\{r_i\}$ be a complete set of orthonormal row eigenvectors for A :** $r_i A = a_i r_i$ **or** $a_i = r_i A r_i^T$

Then

$$A = \Sigma_i r_i^T a_i r_i$$

and

$$A^{-1} = \Sigma_i r_i^T a_i^{-1} r_i = \Sigma_i r_i^T (r_i A r_i^T)^{-1} r_i$$

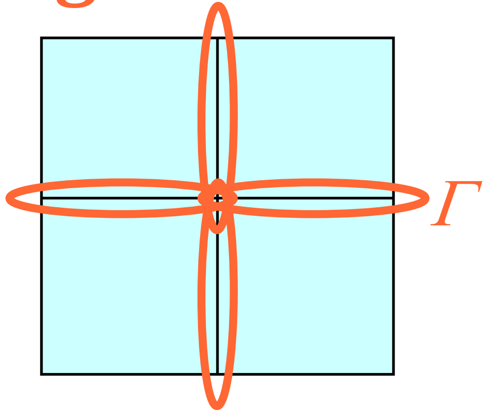
— the Schwarz formula!

- **Good decompositions are a compromise between conditioning and parallel complexity, in practice**

Schur complement substructuring

- **Given a partition**

$$\begin{bmatrix} A_{ii} & A_{i\Gamma} \\ A_{\Gamma i} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_i \\ f_\Gamma \end{bmatrix}$$



- **Condense:**

$$S u_\Gamma = g \qquad S \equiv A_{\Gamma\Gamma} - A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma} \qquad g \equiv f_\Gamma - A_{\Gamma i} A_{ii}^{-1} f_i$$

- **Properties of the Schur complement:**
  - **smaller than original A, but generally dense**
  - **expensive to form, to store, to factor, and to solve**
  - **better conditioned than original A, for which $\kappa(A)=O(h^{-2})$**
  - **for a single interface, $\kappa(S)=O(h^{-1})$**
- **Therefore, solve iteratively, with action of S on each Krylov vector**

Schur preconditioning

- **Note the factorization of the system matrix**

$$A = \begin{bmatrix} A_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix} \begin{bmatrix} I & A_{ii}^{-1} A_{i\Gamma} \\ 0 & S \end{bmatrix}$$

- **Hence a perfect preconditioner is**

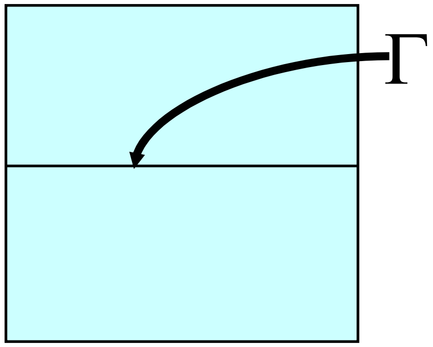
$$A^{-1} = \begin{bmatrix} I & A_{ii}^{-1} A_{i\Gamma} \\ 0 & S \end{bmatrix}^{-1} \begin{bmatrix} A_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} I & -A_{ii}^{-1} A_{i\Gamma} S^{-1} \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} A_{ii}^{-1} & 0 \\ -A_{\Gamma i} A_{ii}^{-1} & I \end{bmatrix}$$

Schur preconditioning

- **Let** M^{-1} **be any good preconditioner for** S
- **Let**

$$B^{-1} = \begin{bmatrix} I & \tilde{A}_{ii}^{-1} A_{i\Gamma} \\ 0 & M \end{bmatrix}^{-1} \begin{bmatrix} \tilde{A}_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix}^{-1}$$



- **Then** B^{-1} **is a good preconditioner for** A**, for recall**

$$A^{-1} = \begin{bmatrix} I & A_{ii}^{-1} A_{i\Gamma} \\ 0 & S \end{bmatrix}^{-1} \begin{bmatrix} A_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix}^{-1}$$

Schur preconditioning

- So, instead of $M^{-1}Su_\Gamma = M^{-1}g$, use full system

$$B^{-1}\begin{bmatrix} A_{ii} & A_{i\Gamma} \\ A_{\Gamma i} & A_{\Gamma\Gamma} \end{bmatrix}\begin{bmatrix} u_i \\ u_\Gamma \end{bmatrix} = B^{-1}\begin{bmatrix} f_i \\ f_\Gamma \end{bmatrix}$$

- Here, solves with A_{ii} may be done approximately since all degrees of freedom are retained
- Once this simple block decomposition is understood, everything boils down to two more profound questions:
  - How to approximate S cheaply
  - How should the relative quality of M and $\tilde{A}_{ii}$ compare

Schur preconditioning

- How to approximate S cheaply?
  - Many techniques for a single interface
  - Factorizations of narrow band approximations
  - Spectral (FFT-implementable) decompositions
  - Algebraic “probing” of a specified sparsity pattern for inverse
- For separator sets more complicated than a single interface, we componentize, creating the preconditioner of the union from the sum of preconditioners of the individual pieces

Schwarz-on-Schur

- **Beyond a simple interface, preconditioning the Schur complement is complex in and of itself; Schwarz is used on the reduced problem**
- **Neumann-Neumann**

$$M^{-1} = \Sigma_i D_i R_i^T S_i^{-1} R_i D_i$$

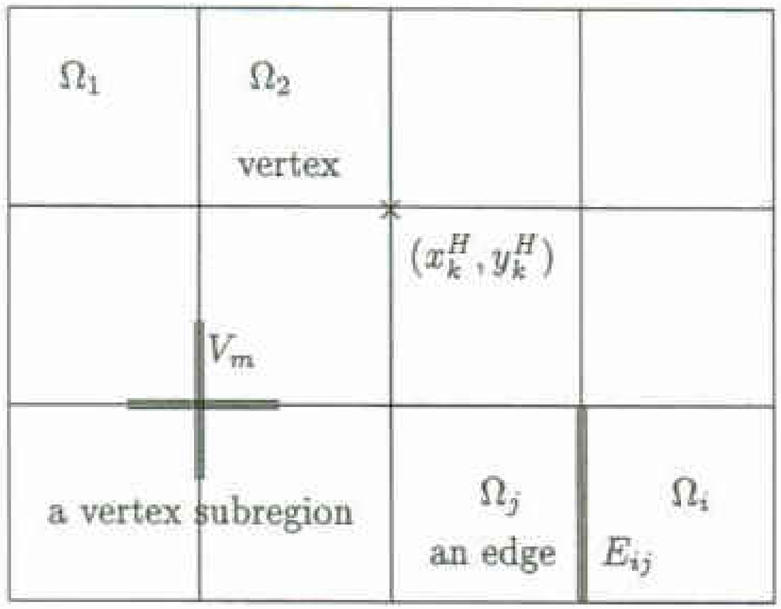
- **Balancing Neumann-Neumann**

$$M^{-1} = M_0^{-1} + (I - M_0^{-1} S)(\Sigma_i D_i R_i^T S_i^{-1} R_i D_i)(I - S M_0^{-1})$$

- ***Numerous* other variants allow inexact subdomain solves, combining additive Schwarz-like preconditioning of the separator set components with inexact subdomain solves on the subdomains**

Schwarz-on-Schur

- **As an illustration of the algorithmic structure, we consider the 2D Bramble-Pasciak-Schatz (1984) preconditioner for the case of many subdomains**



$$\begin{bmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{bmatrix} \begin{bmatrix} u_I \\ u_B \end{bmatrix} = \begin{bmatrix} f_I \\ f_B \end{bmatrix}$$

$$A_{II} = \text{blockdiag}\,(A_{ii}) = \begin{bmatrix} A_{11} & & 0 \\ & \ddots & \\ 0 & & A_{pp} \end{bmatrix}$$

$$A \equiv \begin{bmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{IB}^T A_{II}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{II} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{II}^{-1} A_{IB} \\ 0 & I \end{bmatrix}$$

Schwarz-on-Schur

- **For this case** $\kappa(S) = O(H^{-1}h^{-1})$ **, which is not as good as the single interface case, for which** $\kappa(S) = O(h^{-1})$
- **The Schur complement has the block structure**

$$S = \begin{bmatrix} S_{EE} & S_{EV} \\ S_{EV}^T & S_{VV} \end{bmatrix}$$

for which the following block diagonal preconditioner improves conditioning only to $O(H^{-2}\log^2(Hh^{-1}))$

$$M^{-1} = \begin{bmatrix} S_{EE}^{-1} & 0 \\ 0 & S_{VV}^{-1} \end{bmatrix}$$

- **Note that we can write** M^{-1} **equivalently as**

$$M^{-1} = \sum_i R_{E_i}^T S_{E_iE_i}^{-1} R_{E_i} + \sum_j R_{V_j}^T S_{V_jV_j}^{-1} R_{V_j}$$

Schwarz-on-Schur

- If we replace the diagonal vertex term of M^{-1} with a coarse grid operator

$$M^{-1} = \sum_i R_{E_i}^T S_{E_i E_i}^{-1} R_{E_i} + R_H^T A_H^{-1} R_H$$

then

$$\kappa(M^{-1}S) = C(1+\log^2(Hh^{-1}))$$

where C may still retain dependencies on other bad parameters, such as jumps in the diffusion coefficients

- The edge term can be replaced with cheaper components
- There are numerous variations in 2D and 3D that conquer various additional weaknesses

Schwarz polynomials

- Polynomials of Schwarz projections that are combinations of additive and multiplicative may be appropriate for certain implementations
- We may solve the fine subdomains concurrently and follow with a coarse grid (redundantly/cooperatively)

$$u \leftarrow u + \Sigma_i B_i^{-1}(f - Au)$$
$$u \leftarrow u + B_0^{-1}(f - Au)$$

- This leads to algorithm "Hybrid II" in S-B-G'96:

$$B^{-1} = B_0^{-1} + (I - B_0^{-1}A)(\Sigma_i B_i^{-1})$$

- Convenient for "SPMD" (single prog/multiple data)

Onward to nonlinearity

- Linear versus nonlinear problems
  - Solving linear algebraic problems often constitutes 90% of the running time of a large simulation
  - The nonlinearity is often a fairly straightforward outer loop, in that it introduces no new types of messages or synchronizations, and has overall many fewer synchronizations than the preconditioned Krylov method or other linear solver inside it
- We can wrap Newton, Picard, fixed-point or other iterations outside, linearize, and apply what we know
- We consider both Newton-outside and Newton-inside methods

Newton-Krylov-Schur-Schwarz: a solver "workhorse"

$$F(u) \approx F(u_c) + F'(u_c)\delta u = 0$$

$$u = u_c + \lambda\, \delta u$$

$$J\delta u = -F$$

$$\delta u = \underset{x \in V \equiv \{F, JF, J^2F, \cdots\}}{\operatorname{argmin}} \{Jx + F\}$$

$$B^{-1} J \delta u = -B^{-1} F$$

$$B^{-1} = \left(\begin{bmatrix} \tilde{A}_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix} \begin{bmatrix} I & \tilde{A}_{ii}^{-1} A_{i\Gamma} \\ 0 & M \end{bmatrix} \right)^{-1}$$

$$\tilde{A}^{-1} = \sum_i R_i^T (R_i A R_i^T)^{-1} R_i$$

Newton

nonlinear solver

asymptotically quadratic

Krylov

accelerator

spectrally adaptive

Schur

preconditioner

parallelizable by structure

Schwarz

preconditioner

parallelizable by domain

Jacobian-free Newton-Krylov

- **In the Jacobian-Free Newton-Krylov (JFNK) method, a Krylov method solves the linear Newton correction equation, requiring Jacobian-vector products**
- **These are approximated by the Fréchet derivatives**

$$J(u)v \approx \frac{1}{\varepsilon}[F(u+\varepsilon v) - F(u)]$$

(where ε is chosen with a fine balance between approximation and floating point rounding error) or automatic differentiation, so that the actual Jacobian elements are *never explicitly needed*

- **One builds the Krylov space on a true $F'(u)$ (to within numerical approximation)**

How to accommodate preconditioning

- **Krylov iteration is expensive in memory and in function evaluations, so subspace dimension *k* must be kept small in practice, through preconditioning the Jacobian with an approximate inverse, so that the product matrix has low condition number in**

$$(B^{-1}A)x = B^{-1}b$$

- **Given the ability to apply the action of B^{-1} to a vector, preconditioning can be done on either the left, as above, or the right, as in, e.g., for matrix-free:**

$$JB^{-1}v \approx \frac{1}{\varepsilon}[F(u+\varepsilon B^{-1}v)-F(u)]$$

Philosophy of Jacobian-free NK

- To *evaluate* the linear residual, we use the true $F'(u)$, giving a true Newton step and asymptotic quadratic Newton convergence
- To *precondition* the linear residual, we do anything convenient that uses understanding of the dominant physics/mathematics in the system and respects the limitations of the parallel computer architecture and the cost of various operations:
  - Jacobian blocks decomposed for parallelism (Schwarz)
  - Jacobian of lower-order discretization
  - Jacobian with “lagged” values for expensive terms
  - Jacobian stored in lower precision
  - Jacobian of related discretization
  - operator-split Jacobians
  - physics-based preconditioning

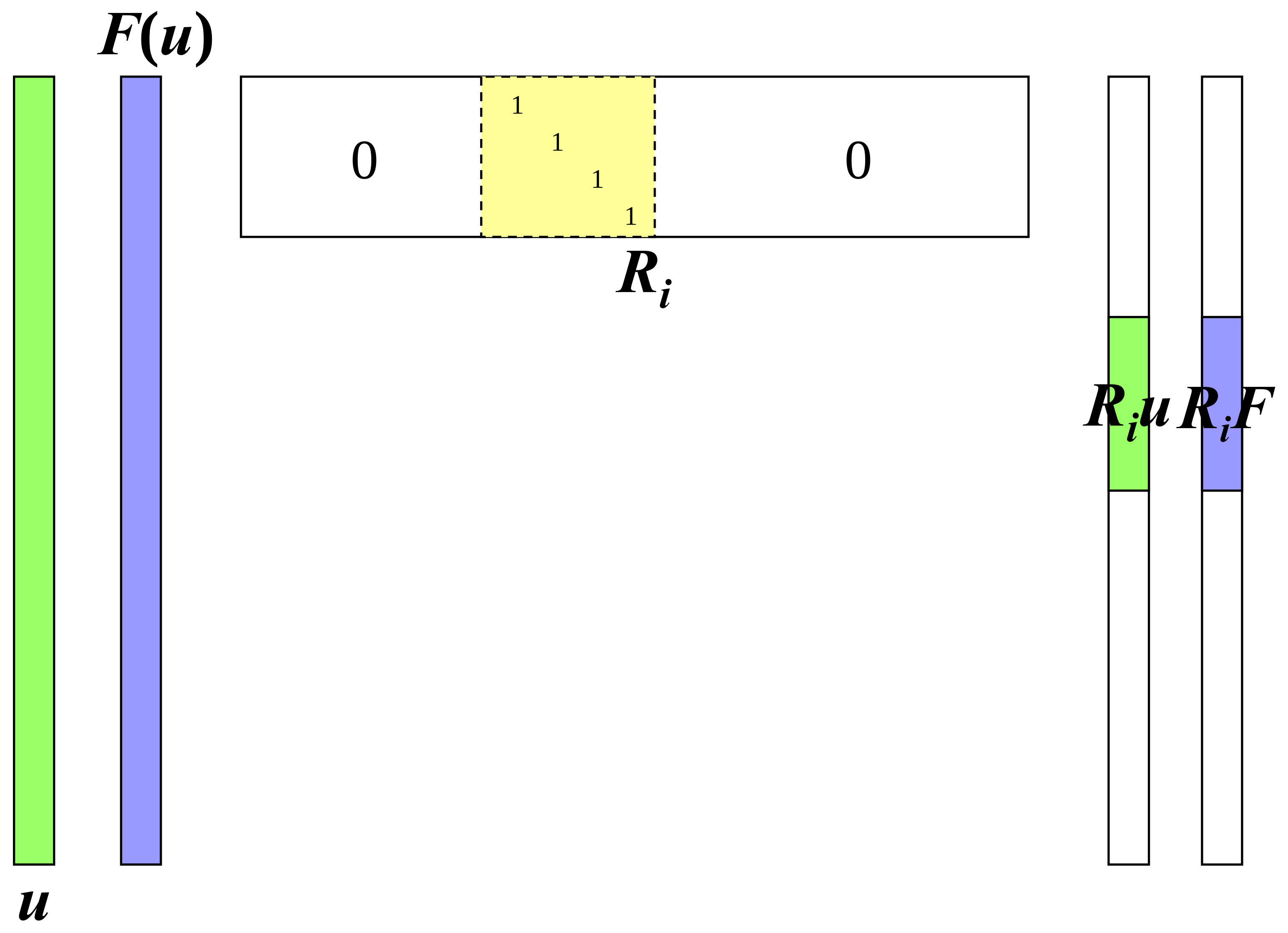
Nonlinear Schwarz preconditioning

- Nonlinear Schwarz has Newton both *inside* and *outside* and is fundamentally Jacobian-free
- It replaces $F(u)=0$ with a new nonlinear system possessing the same root, $\Phi(u)=0$
- Define a correction $\delta_i(u)$ to the i^{th} partition (e.g., subdomain) of the solution vector by solving the following local nonlinear system:

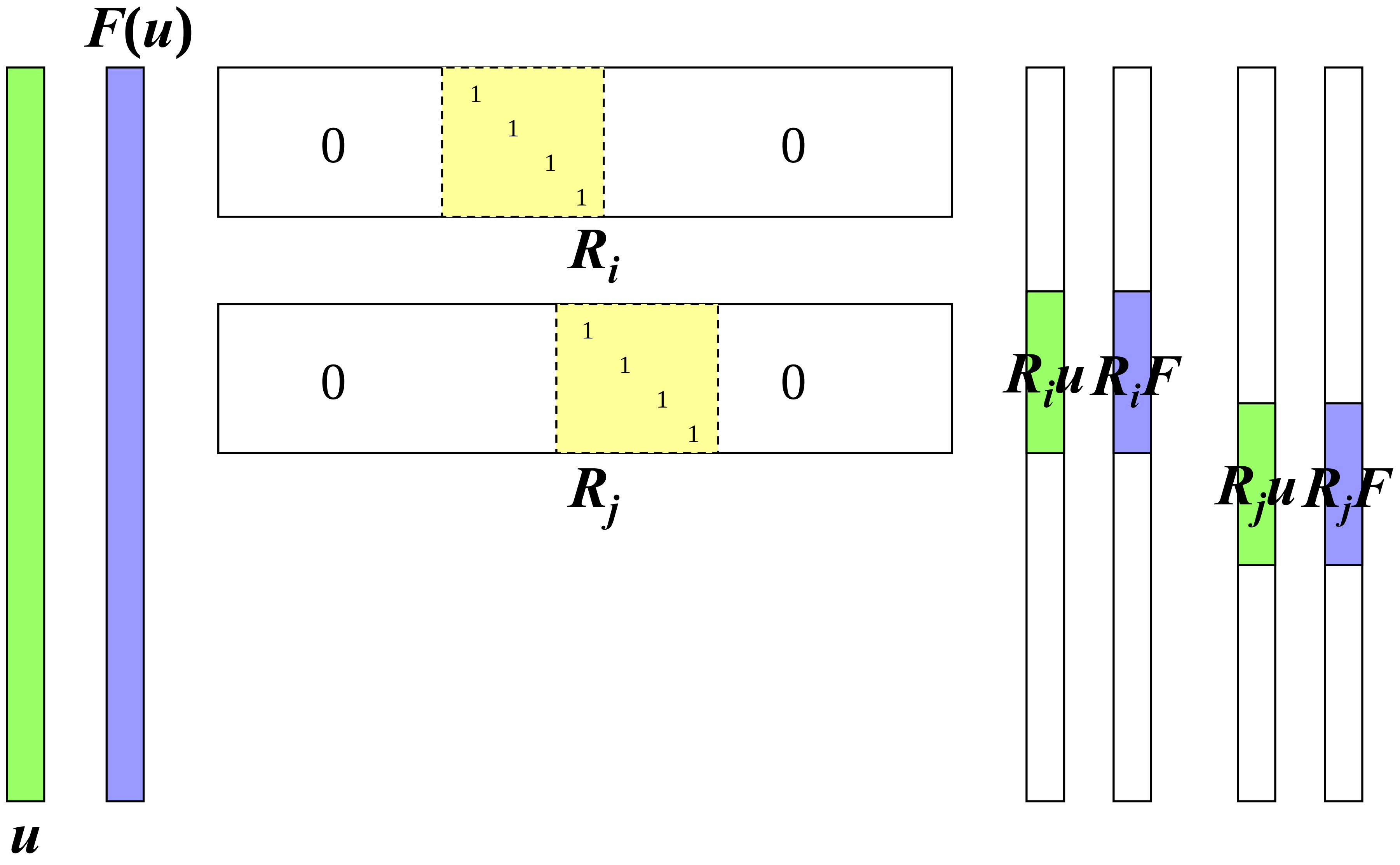
$$R_i F(u+\delta_i(u))=0$$

  where $\delta_i(u)\in\Re^n$ is nonzero only in the components of the i^{th} partition
- Then sum the corrections: $\Phi(u)=\sum_i \delta_i(u)$ to get an implicit function of u

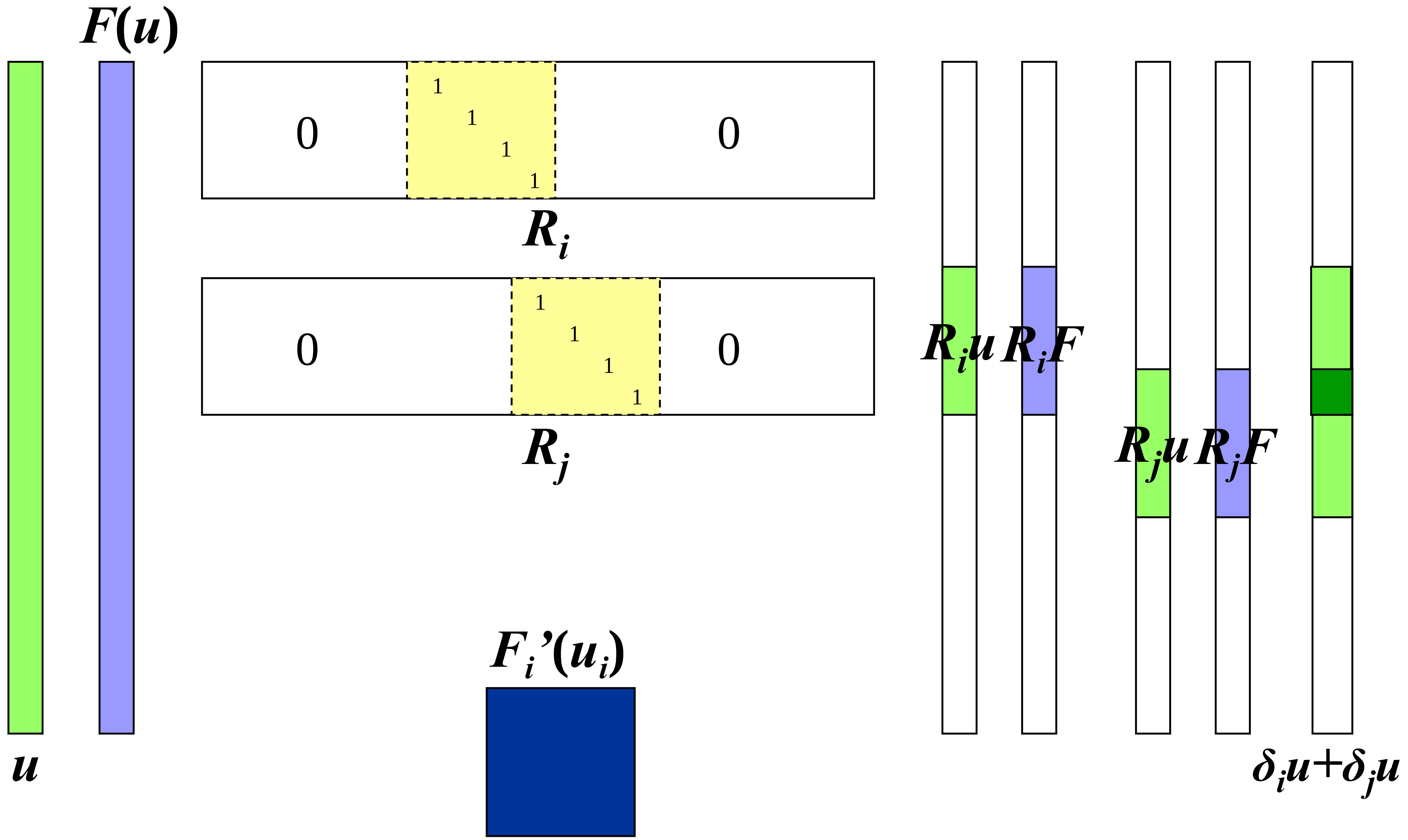
Nonlinear Schwarz – picture



Nonlinear Schwarz – picture



Nonlinear Schwarz – picture



Nonlinear Schwarz, cont.

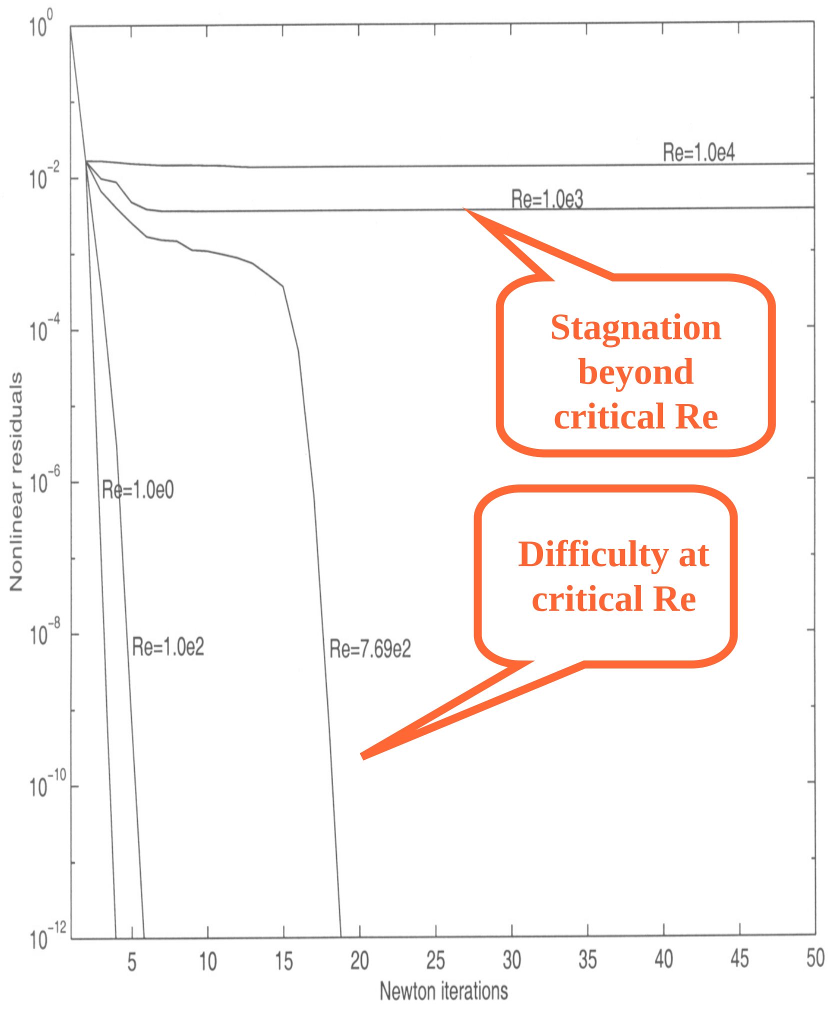
- It is simple to prove that if the Jacobian of $F(u)$ is nonsingular in a neighborhood of the desired root then $\Phi(u) = 0$ and $F(u) = 0$ have the same unique root
- To lead to a Jacobian-free Newton-Krylov algorithm we need to be able to evaluate for any $u, v \in \Re^n$:
  - The residual $\Phi(u) = \sum_i \delta_i(u)$
  - The Jacobian-vector product $\Phi(u)' v$
- Remarkably, (Cai-Keyes, 2000) it can be shown that

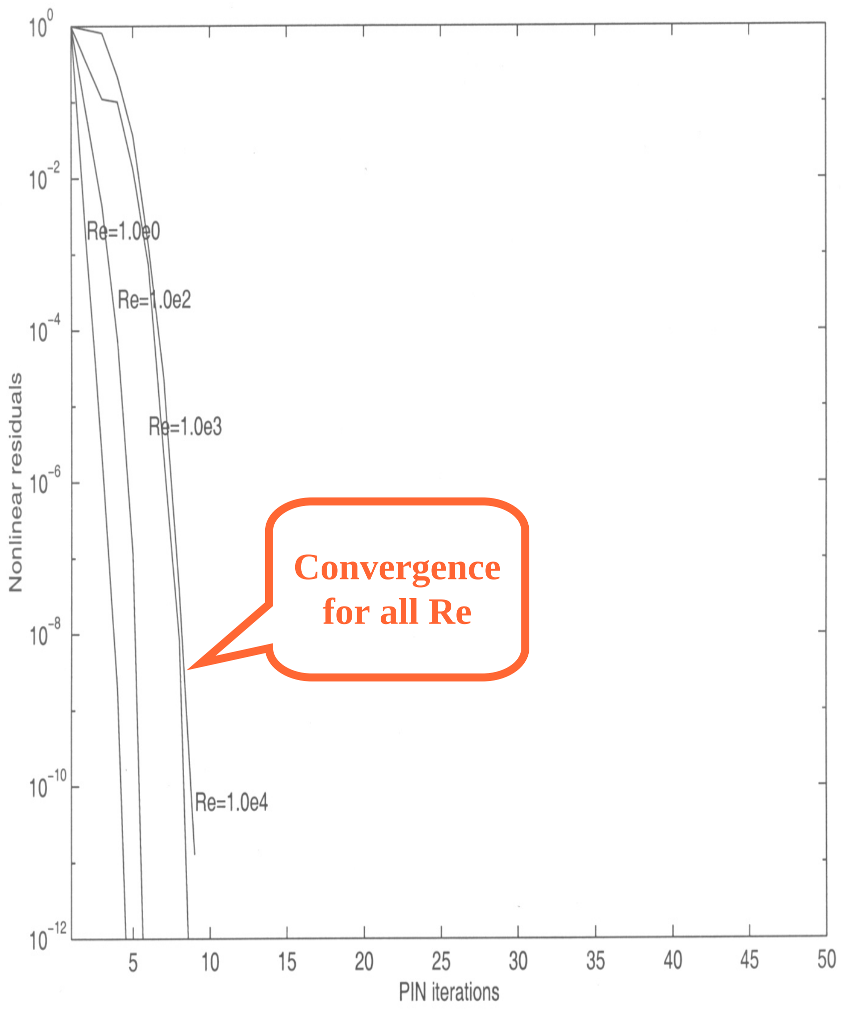
$$\Phi'(u)v \approx \sum_i (R_i^T J_i^{-1} R_i) Jv$$

where $J = F'(u)$ and $J_i = R_i J R_i^T$

- All required actions are available in terms of $F(u)$!

Experimental example of nonlinear Schwarz





Vanilla Newton's method

Nonlinear Schwarz

Multiphysics coupling: nonlinear Schwarz

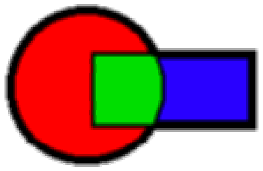
- **Given initial iterate** $\{u_1^0, u_2^0\}$
- **For** *k=1, 2, …,* **until convergence, do**
  - Define $G_1(u_1,u_2) \equiv \delta u_1$ by $F_1(u_1^{k-1}+\delta u_1, u_2^{k-1}) = 0$
  - Define $G_2(u_1,u_2) \equiv \delta u_2$ by $F_2(u_1^{k-1}, u_2^{k-1}+\delta u_2) = 0$
- **Then solve** $\begin{cases} G_1(u,v)=0 \\ G_2(u,v)=0 \end{cases}$ **in matrix-free manner**
- **Jacobian:**

$$\begin{bmatrix} \frac{\partial G_1}{\partial u} & \frac{\partial G_1}{\partial v} \\ \frac{\partial G_2}{\partial u} & \frac{\partial G_2}{\partial v} \end{bmatrix} \approx \begin{bmatrix} I & \left(\frac{\partial F_1}{\partial u}\right)^{-1} \frac{\partial F_1}{\partial v} \\ \left(\frac{\partial F_2}{\partial v}\right)^{-1} \frac{\partial F_2}{\partial u} & I \end{bmatrix}$$

- **Finally** $\{u_1^k, u_2^k\} = \{v, w\}$

State of the art

- Domain decomposition is the dominant paradigm in contemporary terascale PDE simulation
- Several freely available software toolkits exist, and successfully scale to thousands of tightly coupled processors for problems on quasi-static meshes
- Concerted efforts underway to make elements of these toolkits interoperate, and to allow expression of the best methods, which tend to be modular, hierarchical, recursive, and above all — *adaptive*!
- Many challenges loom at the “next scale” of computation
- Implementation of domain decomposition methods on parallel computers has inspired many useful variants of domain decomposition methods
- The past few years have produced an incredible variety of interesting results (in both the continuous and the discrete senses) in domain decomposition methods, with no slackening in sight



Closing inspiration

"… at this very moment the search is on – every numerical analyst has a favorite preconditioner, and you have a perfect chance to find a better one."

- Gil Strang (1986)