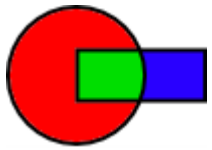
Fourth International Workshop on
Numerical Analysis and Lattice QCD

Yale University and NSF

3 May 2007

Scaling of Domain Decomposition and Multigrid Methods



David E. Keyes

Department of Applied Physics & Applied Mathematics

Columbia University

Happy Birthday, Vito

- **Born 3 May 1860 (Ancona)**
- **PhD Physics, 1882 (Pisa)**
- **Invented the idea of a "functional" in 1883**
- **Wrote on "Volterra" integral equations, 1884-1892**
- **Worked on population biology, predator-prey systems, cf. 1918**
- **Abandoned Italy under fascism for Paris in 1931**
- **Died 11 Oct 1940 (Rome)**

Vito Volterra

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"**

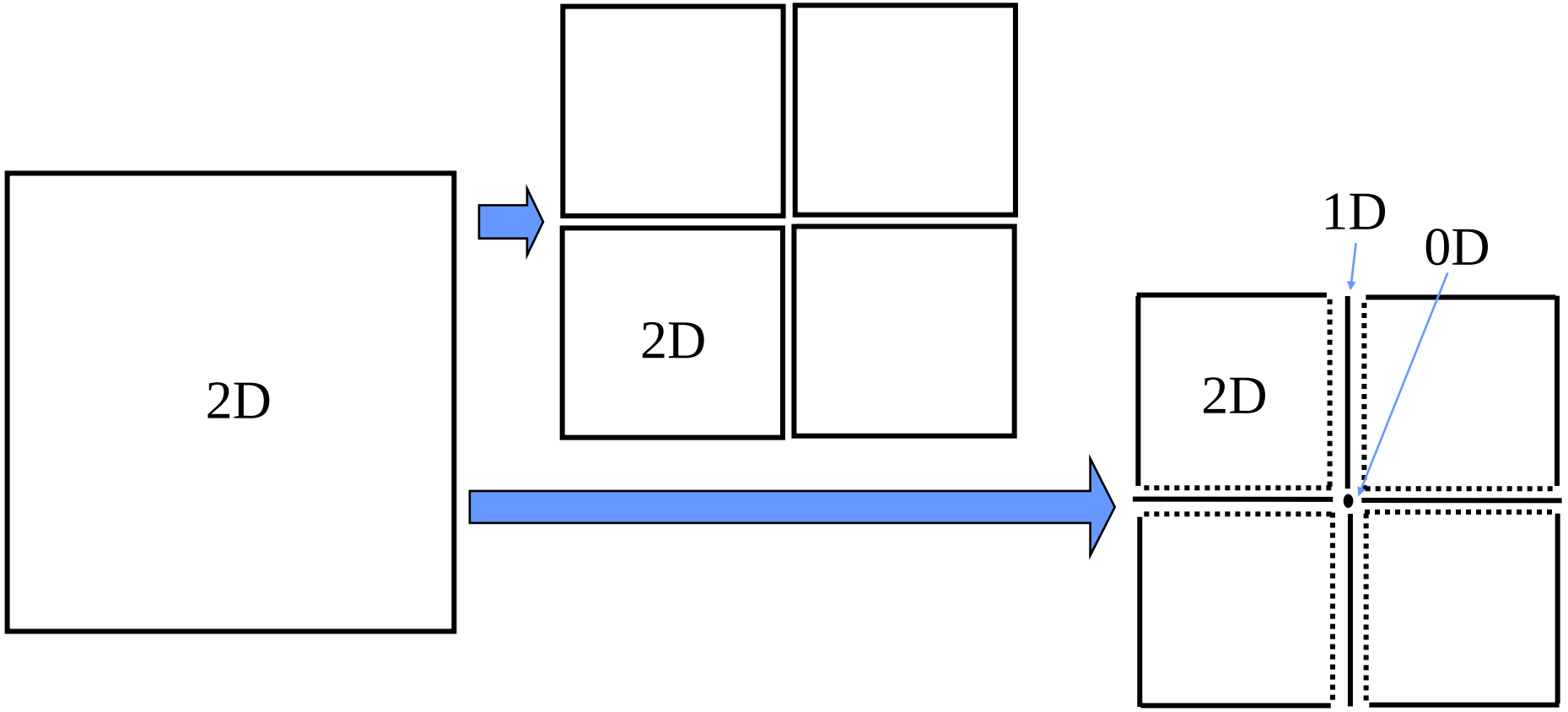
> We often think we know all about "two" because two is "one and one". We forget that we have to make a study of "and."
>
> A. S. Eddington (1882-1944)

Remarks on definition

- **Domain decomposition has generic and specific senses within the universe of parallel algorithms**
  - **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

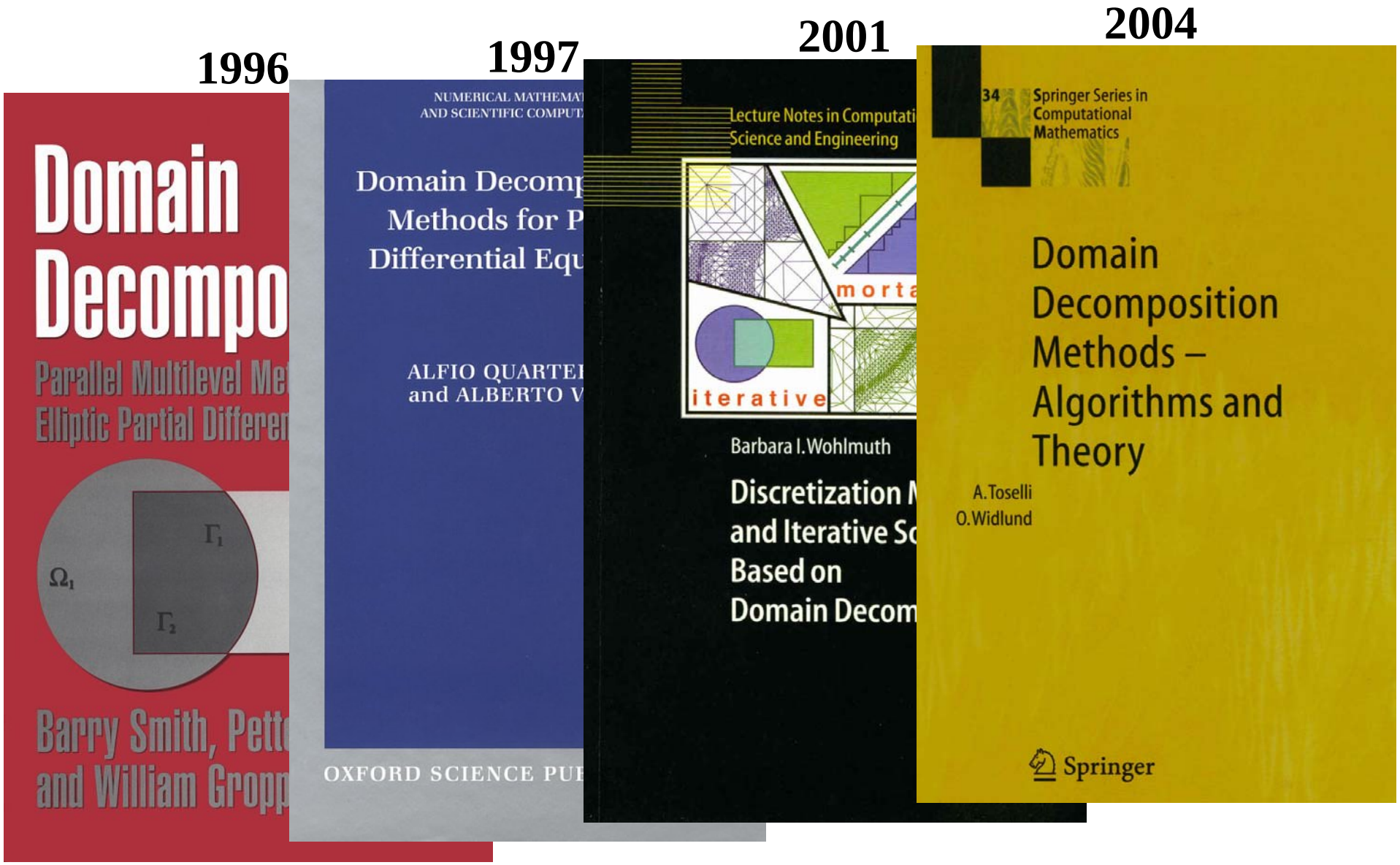
- **The subdomains may be of the same or different dimensionality as the original**



Plan of presentation

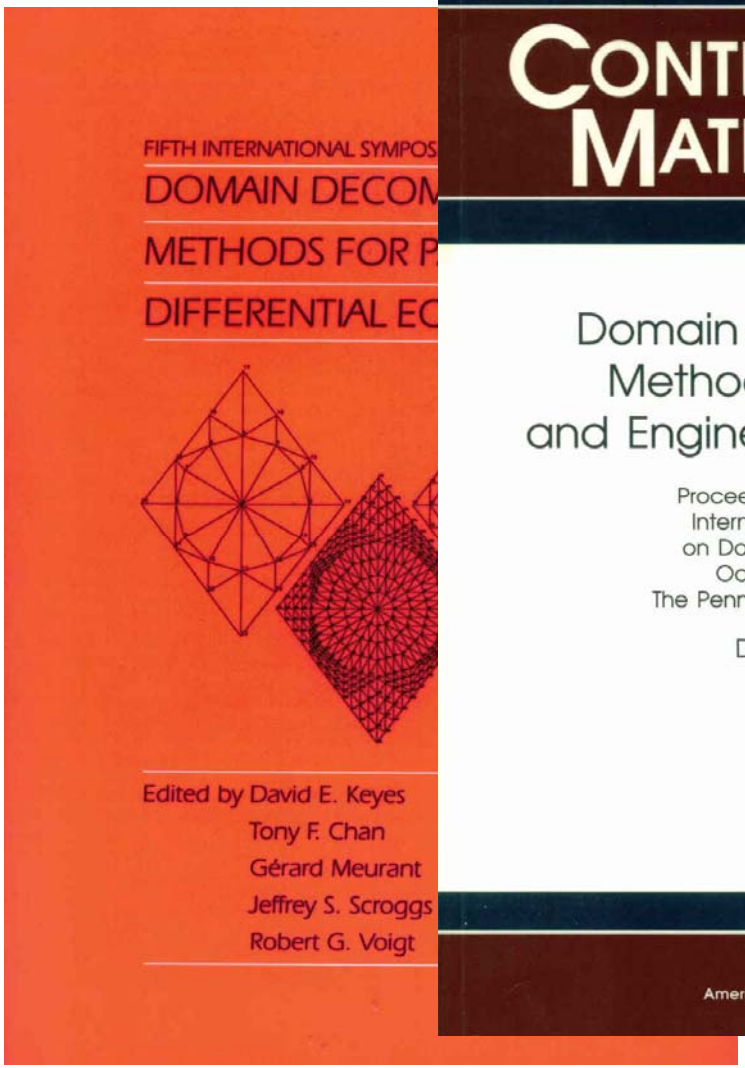
- **Imperative of domain decomposition (DD) for terascale computing**
- **Basic DD algorithmic concepts**
  - **Schwarz**
  - **Schur**
  - **Schwarz-Schur combinations**
- **Basic DD convergence and scaling properties**
- **Comparison of DD and multilevel preconditioners in parallel (c/o P. Fischer)**

Prime sources for domain decomposition



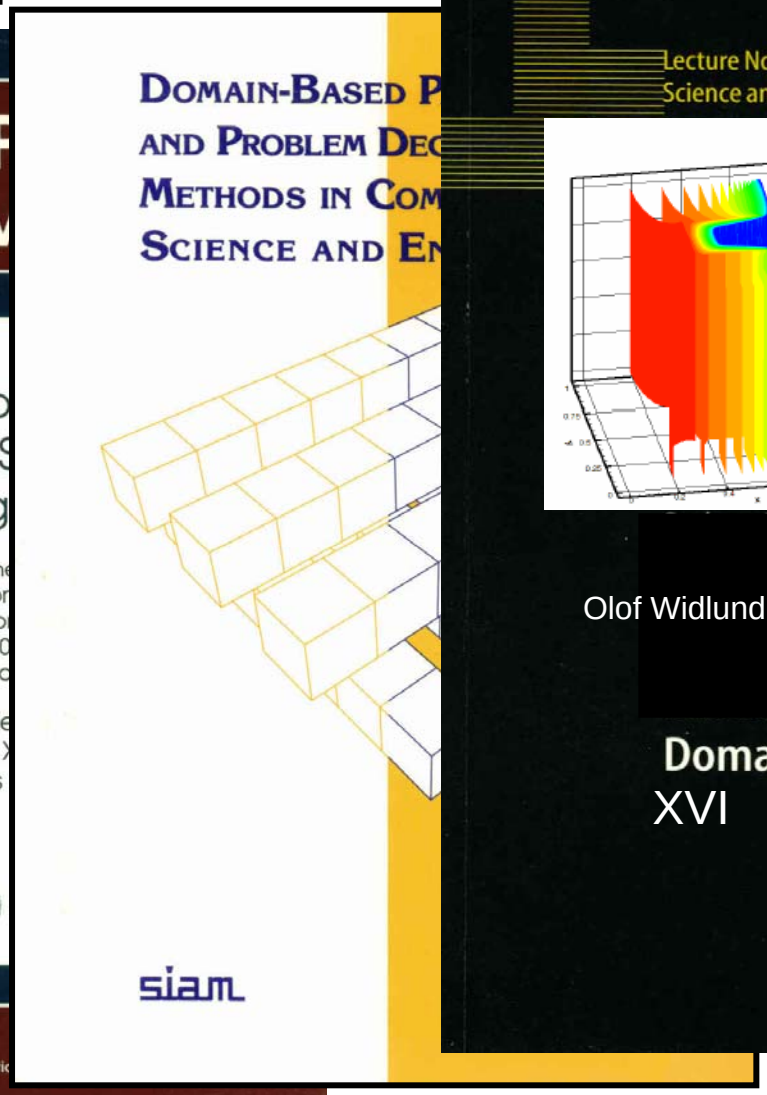
Other sources for domain decomposition

1992

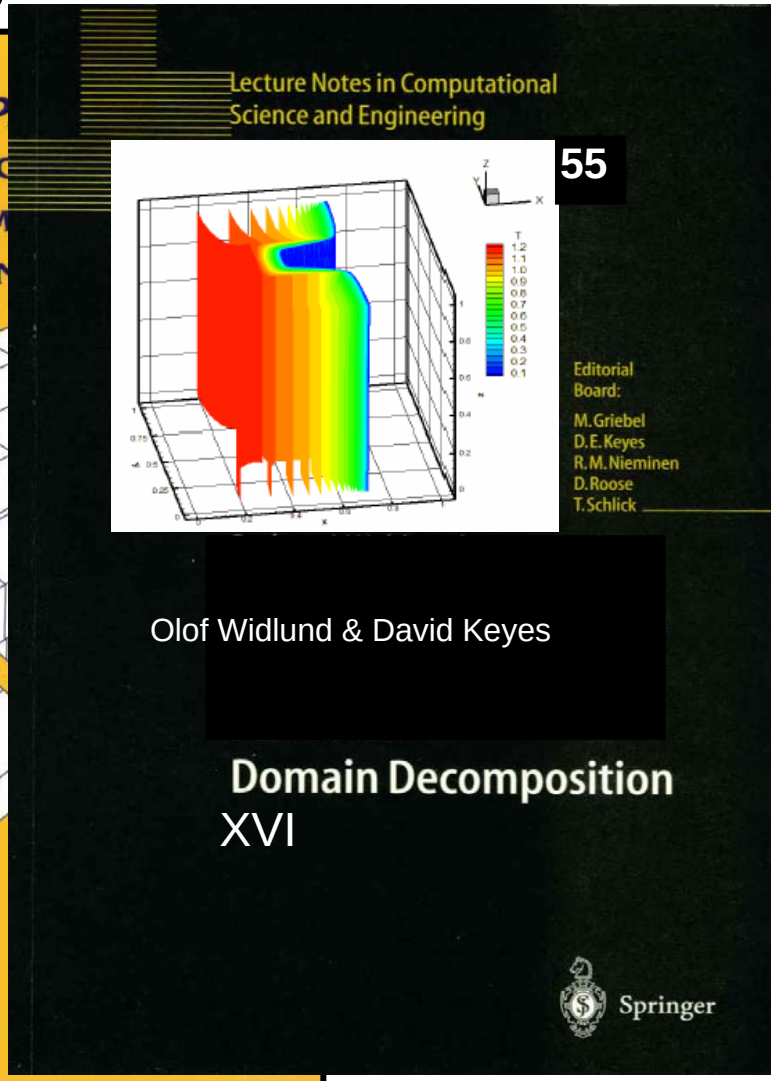


1994

1995



2001

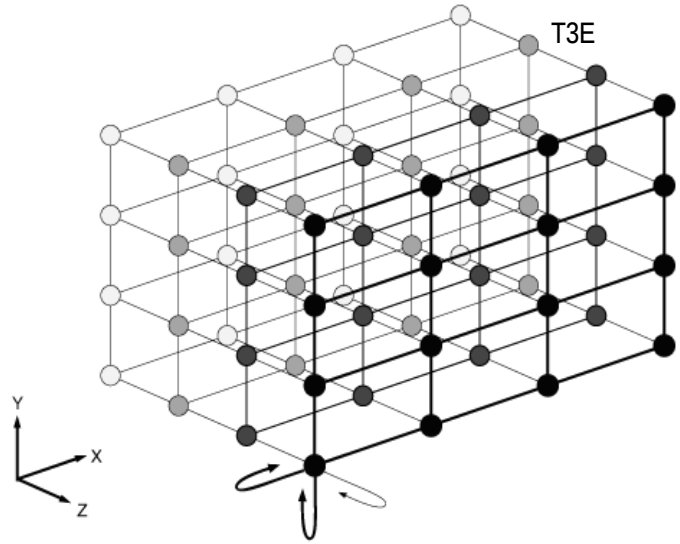


+ DDM.ORG and other proceedings volumes, 1988-2006

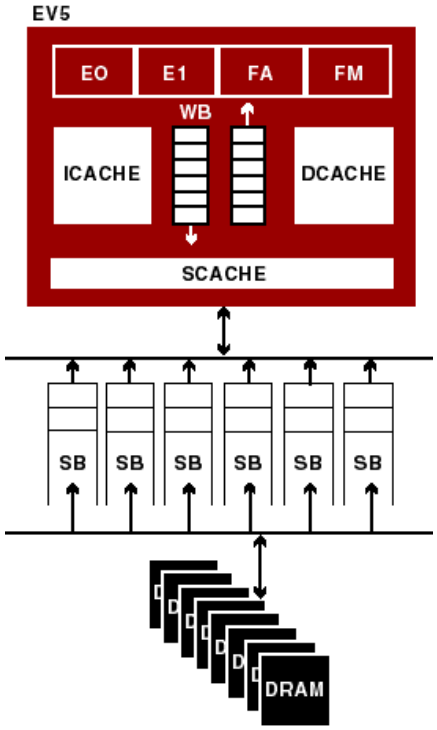
Algorithmic requirements from architecture

- **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



Platforms capable of peak petaflop/s by 2009

	Scale Demonstrated Factor to PF/s	Failures per Month Per TF/s	Power Consumption @ 1 PF/s	Estimated System Cost
Cray XT3/XT4	~~10880 cpus~~ 10x to PF ~100,000 cpus	~.1 - ~1	~8MW XT4	>$150M XT4 +memory
IBM Power5/6	~~10240 cpus~~ 7x to PF ~72,000 cpus	1.3	~9.4MW P6	>$170M P6 +memory
Clusters x86-64/AMD64	~~8000 cpus~~ 12x to PF ~100,000 cpus	2.6-8.0	~6MW x86QC	> $150M x86 +memory
Blue Gene L/P	~~131,072 cpus~~ 2.2x to PF 294,912 cpus	.01-.03	~2.3MW P	< $100M BG Including 288TB

Overview of laboratory plans

- **Cray XT**
  - **Lawrence Berkeley**
  - **Oak Ridge**
  - **Sandia**
- **IBM BlueGene**
  - **Argonne**
  - **Lawrence Livermore**
- **IBM Cell**
  - **Los Alamos**
- ***Caveat: not all of the plans are approved for funding***

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!**

Contemporary interest

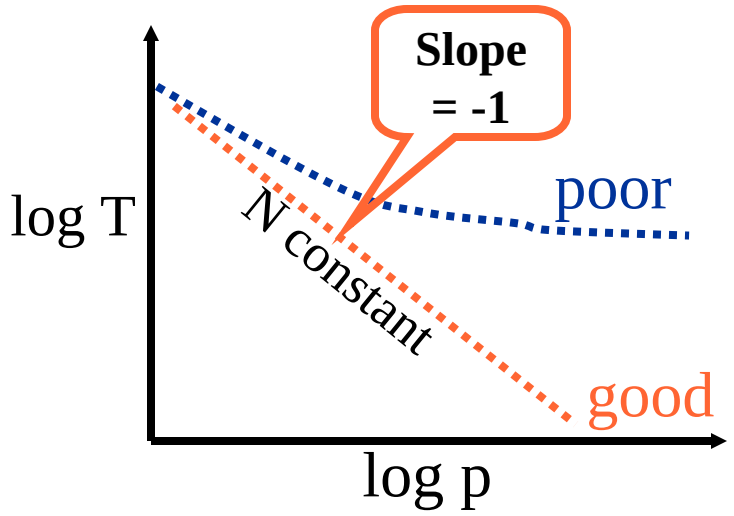
- **Goal is algorithmic scalability:**

  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

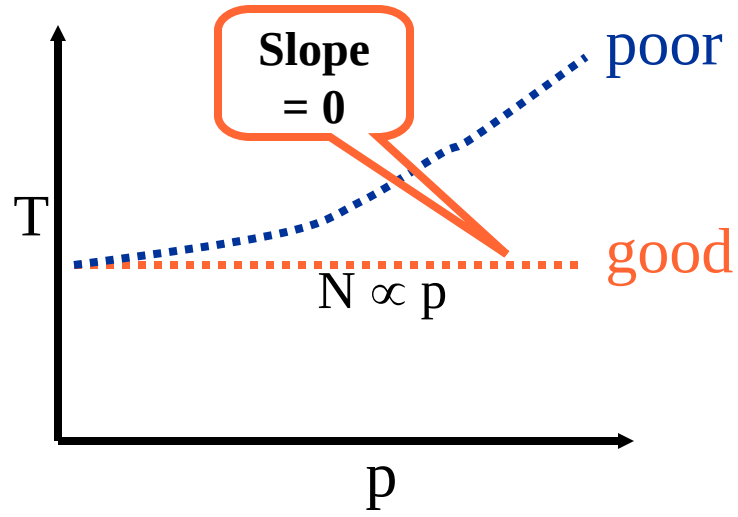
*at worst logarithmically growing

Two definitions of scalability

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

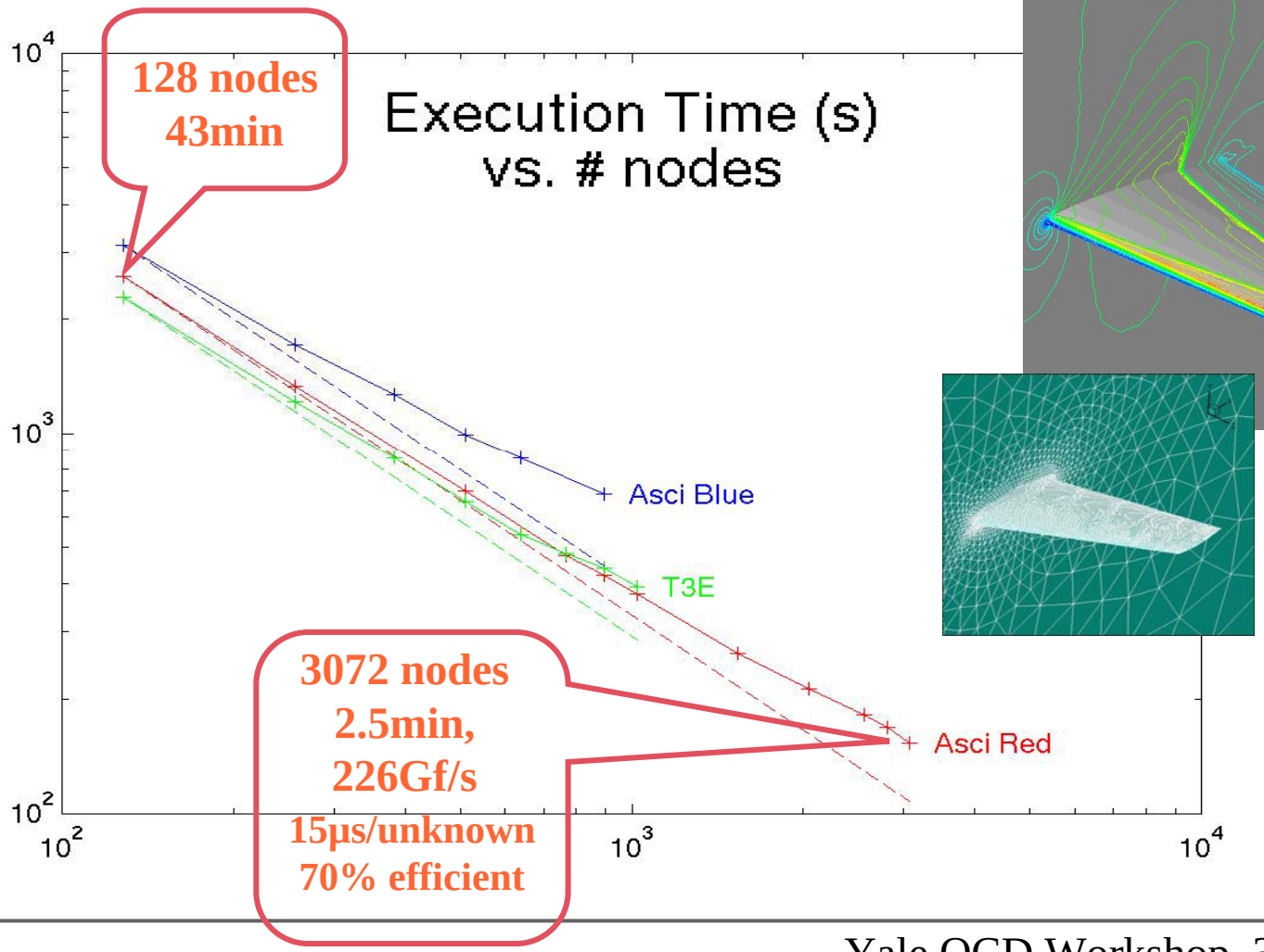


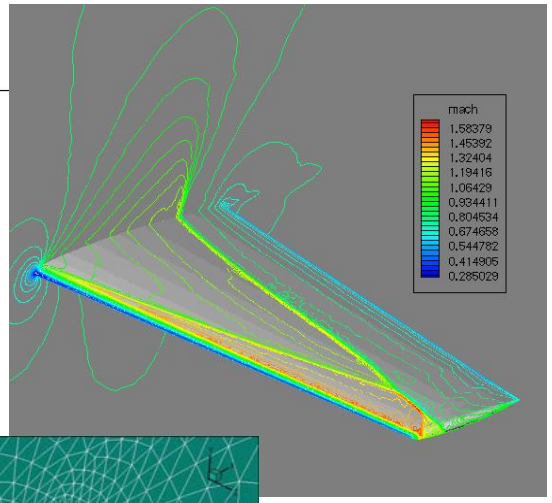
- **"Weak scaling"**
  - **execution time remains constant, as problem size and processor number are increased in proportion**
  - ***fixed size problem per processor***
  - **also known as "Gustafson scaling"**



Strong scaling illus. (1999 Bell Prize)

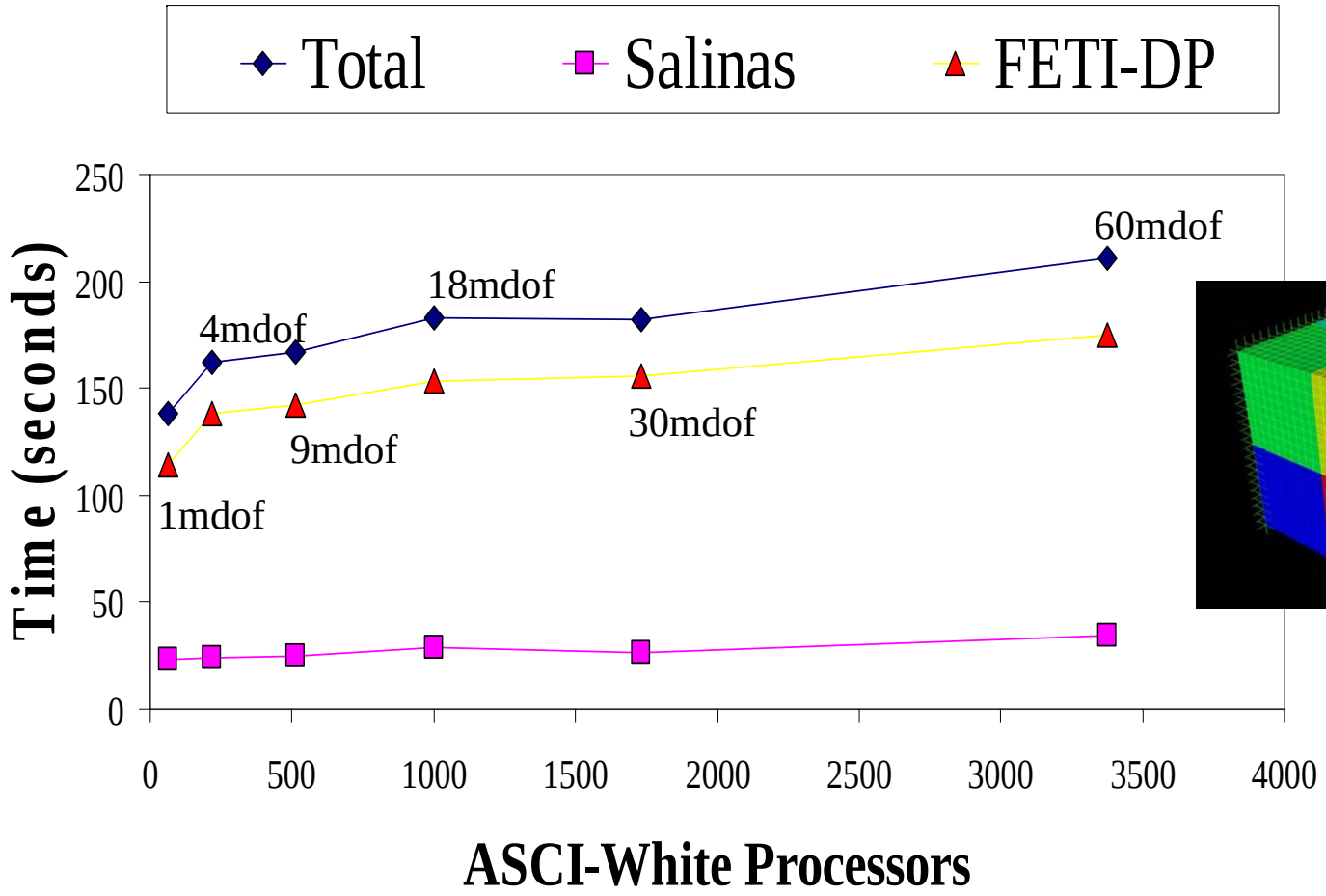
- Newton-Krylov-Schwarz (NKS) algorithm for compressible and incompressible Euler and Navier-Stokes flows
- Used in NASA application FUN3D (M6 wing results below with 11M dof)

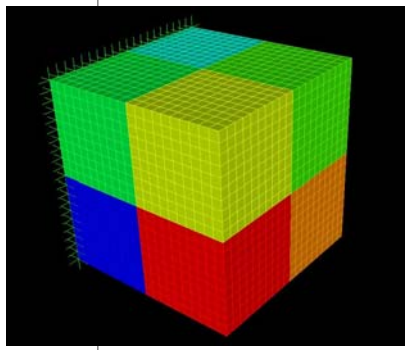




Weak scaling illus. (2002 Bell Prize)

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





Sandia National Laboratories

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**

$$\Omega = \bigcup_k \Omega_k$$

Parabolic PDE example

- **Continuous**

$$(\frac{\partial}{\partial t} - \nabla^2)\, u = f$$

- **Semi-discrete in time**

$$(\frac{I}{\tau} - \nabla^2)\, u^{(k+1)} = \frac{I}{\tau} u^{(k)} + f$$

- **Spatial discretization**

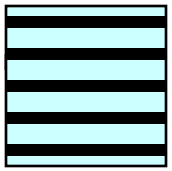
$$[\frac{I}{\tau} + \frac{1}{h^2}(\mathcal{L}_x + \mathcal{L}_y)]u^{(k+1)} = \frac{I}{\tau} u^{(k)} + f$$

$$\mathcal{L}_x = I \otimes \text{tridiag}\{-1,2,-1\}$$

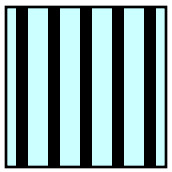
$$\mathcal{L}_y = \text{tridiag}\{-1,2,-1\} \otimes I$$

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**

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

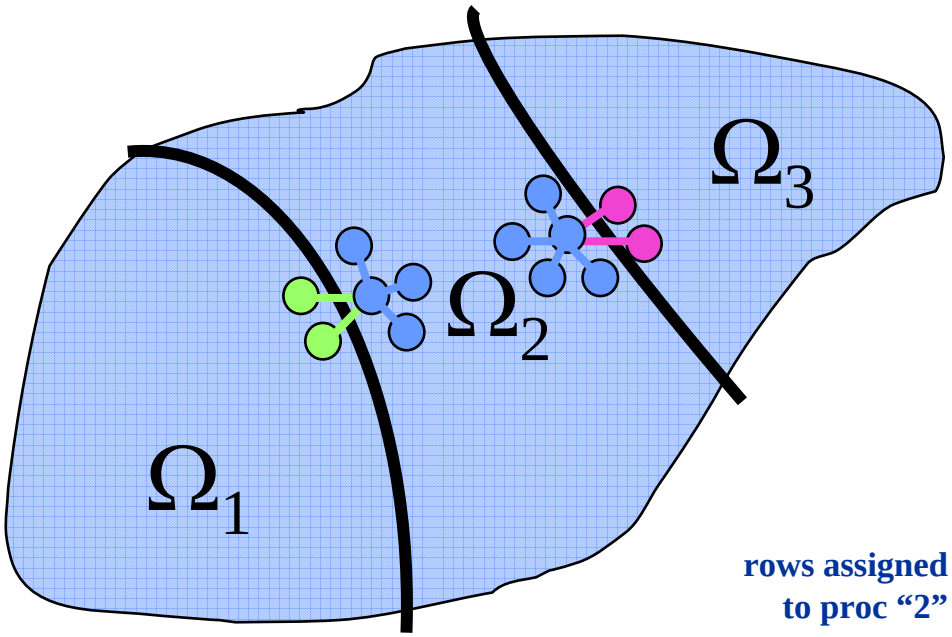
$$\frac{d}{dt}(\Phi_i, u) = (\Phi_i, \mathcal{L}u) + (\Phi_i, f), i = 1,\ldots,N$$

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

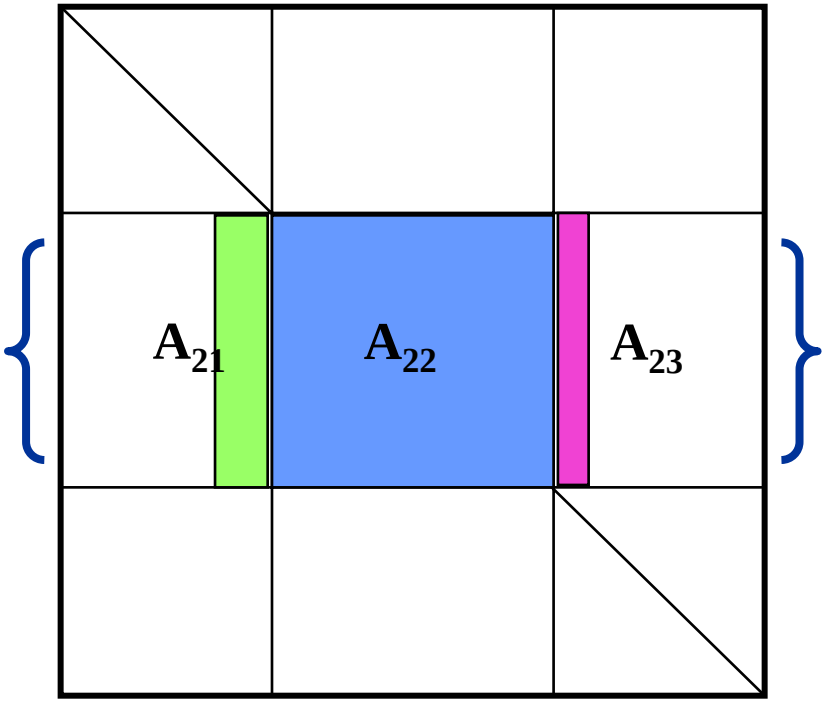
$$\frac{da}{dt} = M^{-1}Ka + M^{-1}f$$

- **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



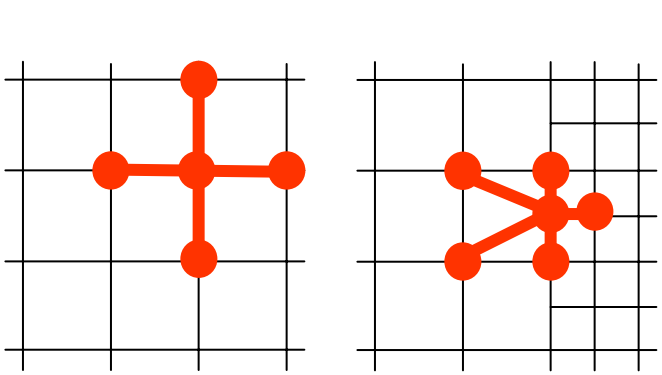
rows assigned
to proc "2"

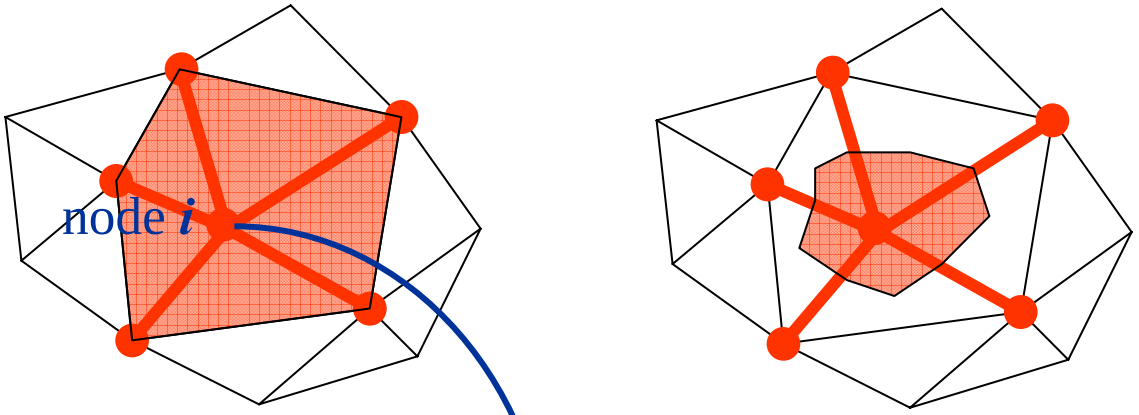


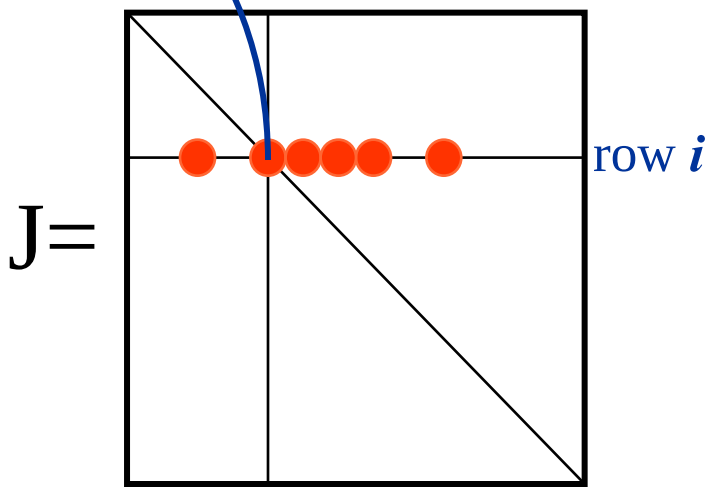
Partitioning of the grid induces block structure on the system matrix (Jacobian)

DD relevant to any local stencil formulation

finite differences finite elements finite volumes



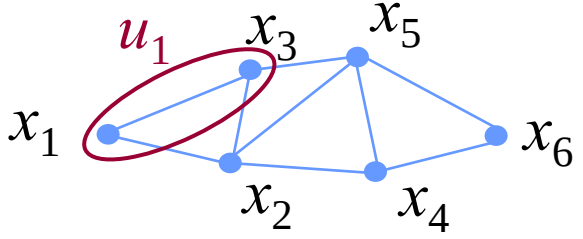




- **All lead to sparse Jacobian matrices**
- **However, the inverses are generally dense; even the factors suffer unacceptable fill-in in 3D**
- **Want to solve in subdomains only, and use to precondition full sparse problem**

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**



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

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

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

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

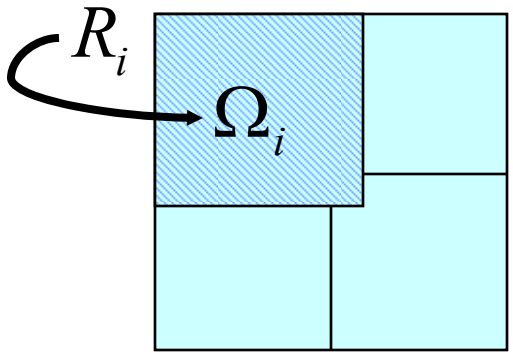
$$R_1^T = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad R_1^T u_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ 0 \\ x_3 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

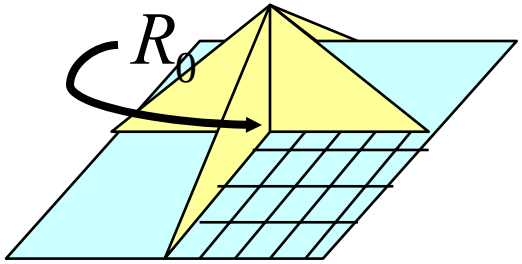
Schwarz domain decomposition method

- **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**

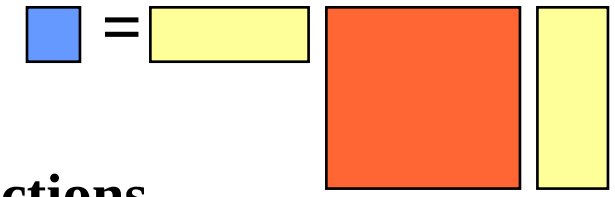
$$B^{-1}Au = B^{-1}f$$

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





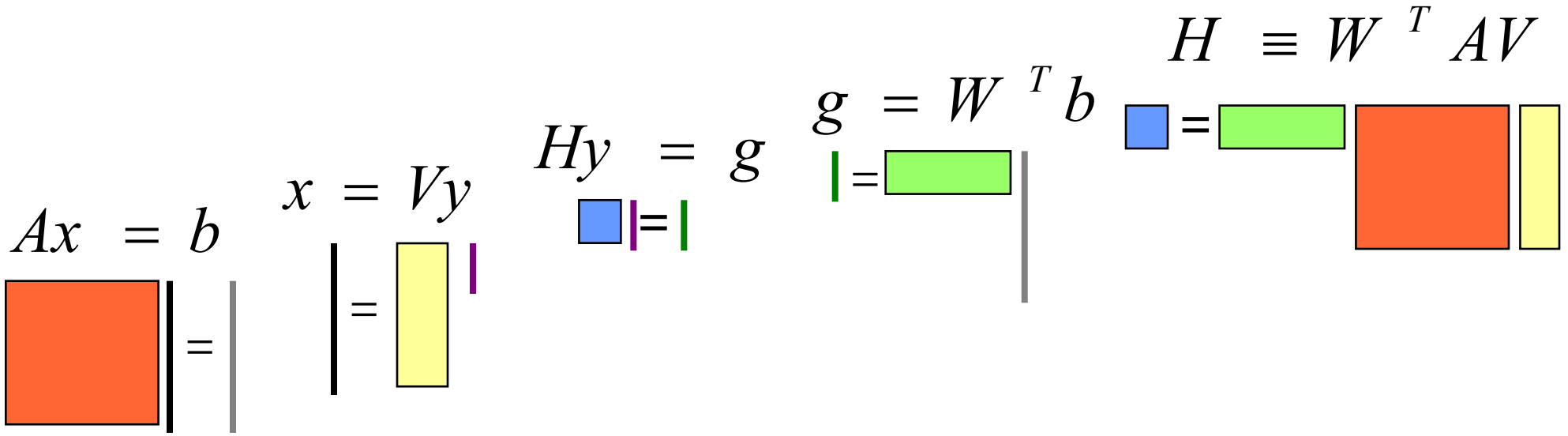
$$A_i = R_i A R_i^T$$



- **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**



Remember this formula of Schwarz …

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

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

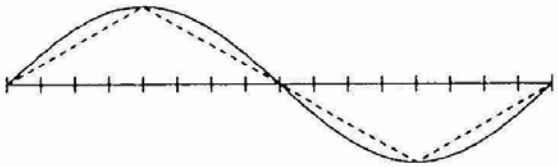
Now, let's compare!

- **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**

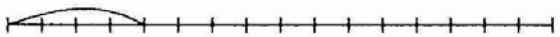
(Of course, domain decomposition can be interpreted as a *special* operator or function space decomposition)

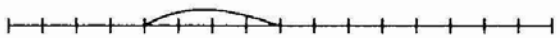
Schwarz subspace decomposition

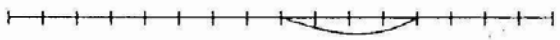
Consider a one-dimensional example. The function $u(x)$ sketched below

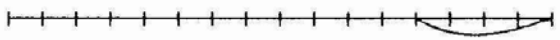


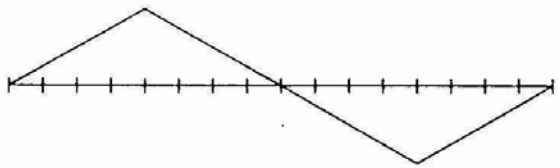
can be decomposed into the sum of the following five functions:





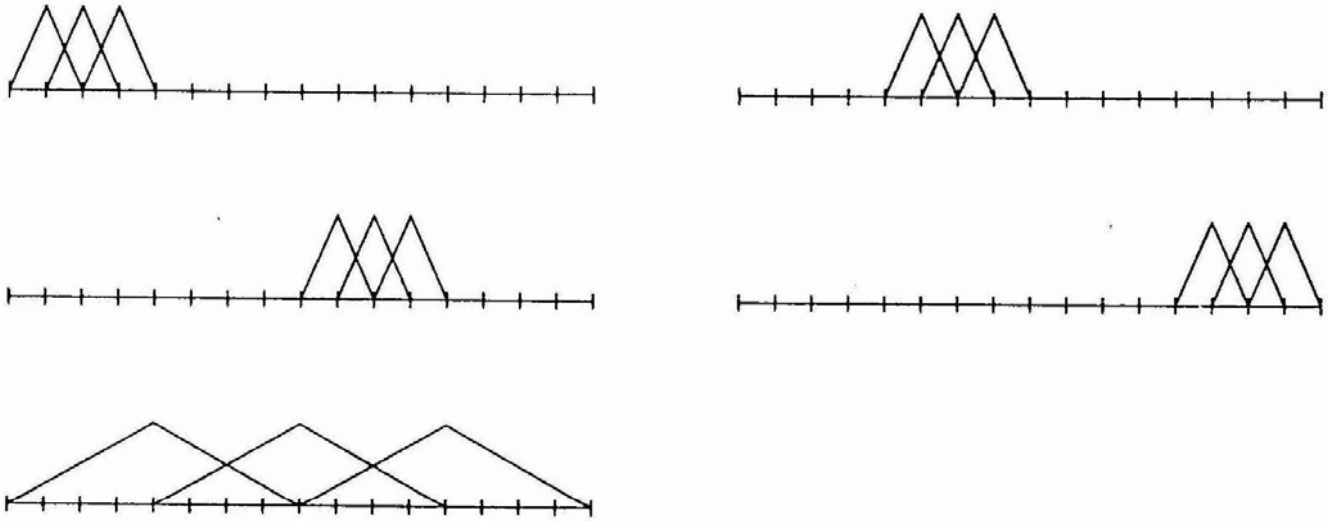






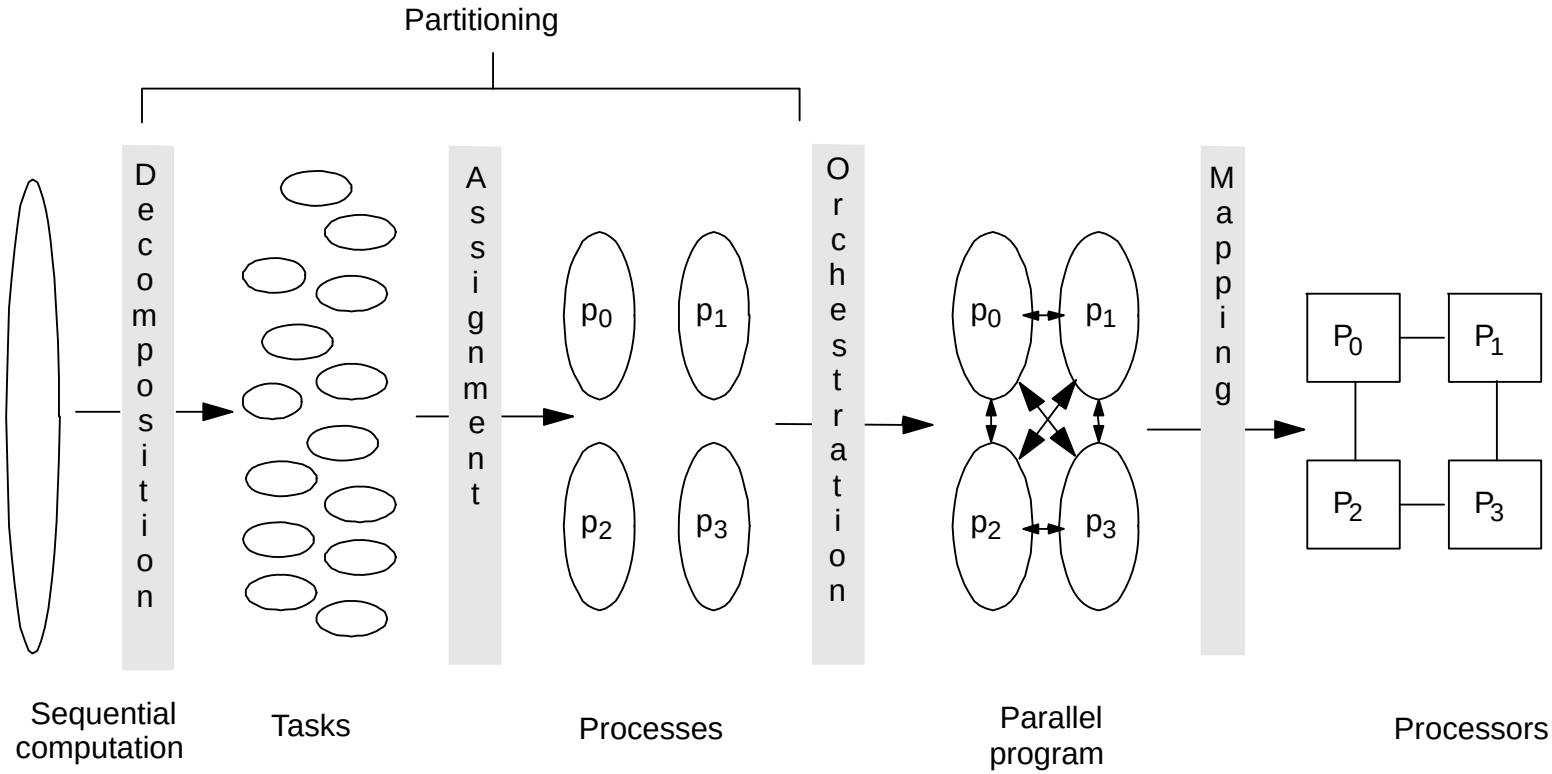
Schwarz subspace decomposition

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

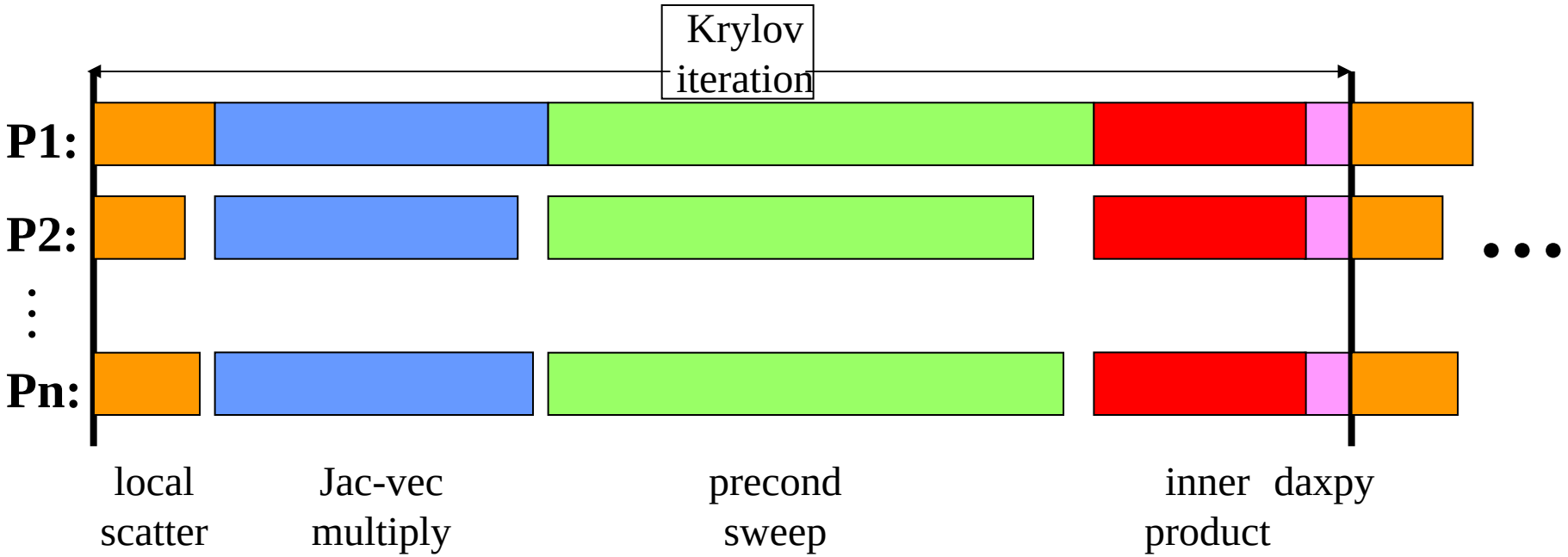


- **Decomposition of computation in tasks**
- **Assignment of tasks to processes**
- **Orchestration of data access, communication, synchronization**
- **Mapping processes to processors**

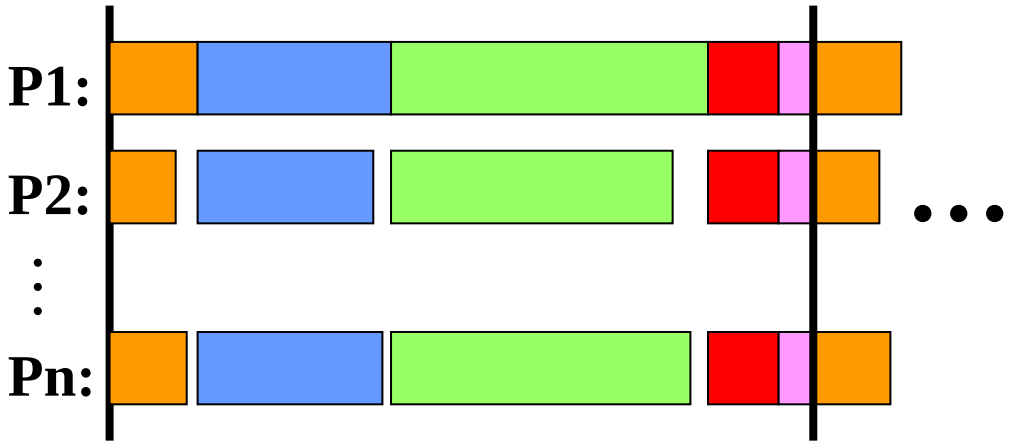
Krylov-Schwarz parallelization summary

- **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

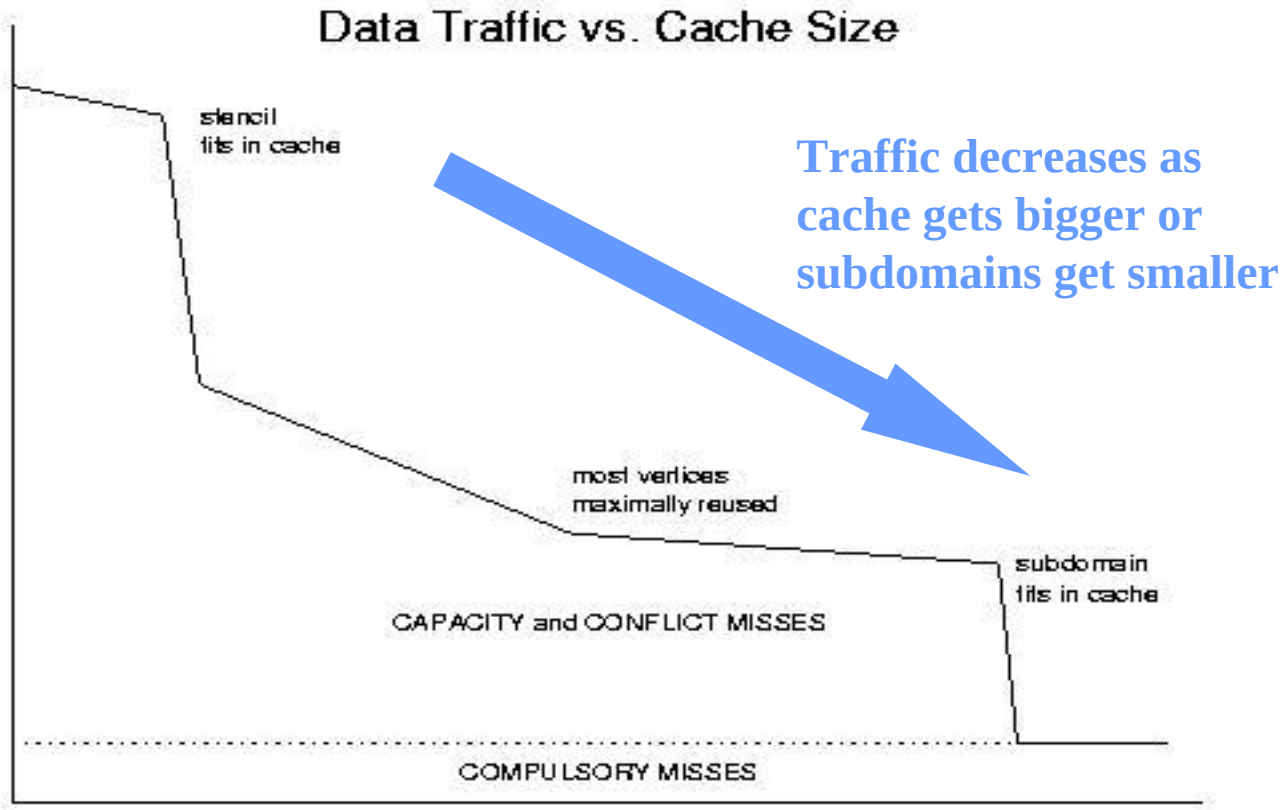


What happens if, for instance, in this (schematicized) iteration, arithmetic speed is ***doubled***, scalar all-gather is ***quartered***, and local scatter is ***cut by one-third***? Each phase is considered separately. Answer is to the right.



Krylov-Schwarz compelling in serial, too

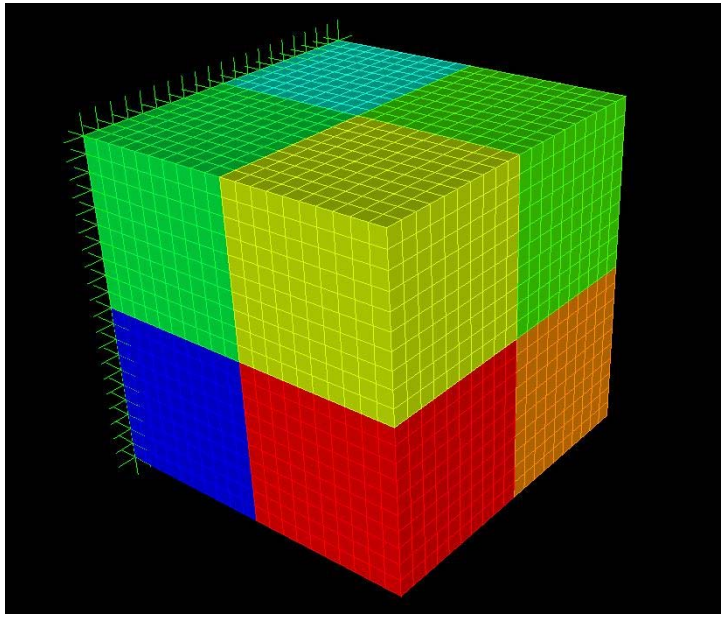
- As successive workingsets “drop” into a level of memory, capacity (and with effort conflict) misses disappear, leaving only compulsory misses, reducing demand on main memory bandwidth
- Cache size is not easily manipulated, but domain size *is*



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 $N=O(n^3)$
- processors in each direction p, total procs $P=O(p^3)$
- 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

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

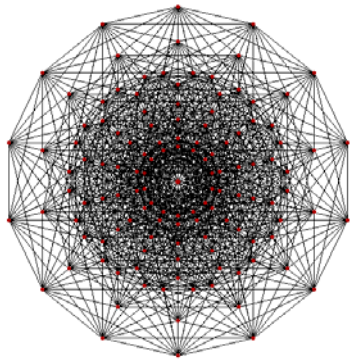
- **limit of infinite neighbor bandwidth, zero neighbor latency ($B \to 0$)**

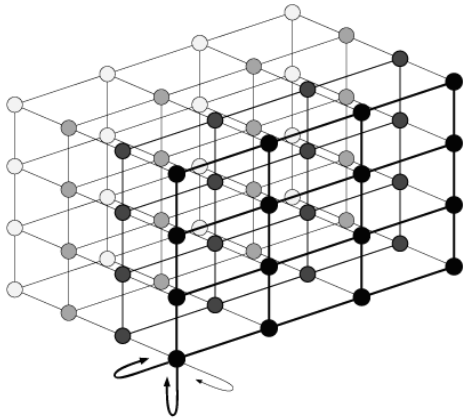
$$T(n, p_{opt}(n)) = C\left[\log n + \frac{1}{3}\log\frac{A}{C} + const.\right]$$

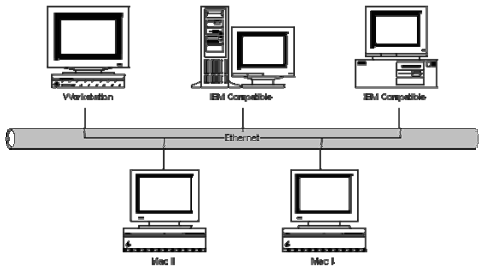
(This analysis is on a per iteration basis; complete analysis multiplies this cost by an iteration count estimate that generally depends on *n* and *p*.)

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")**







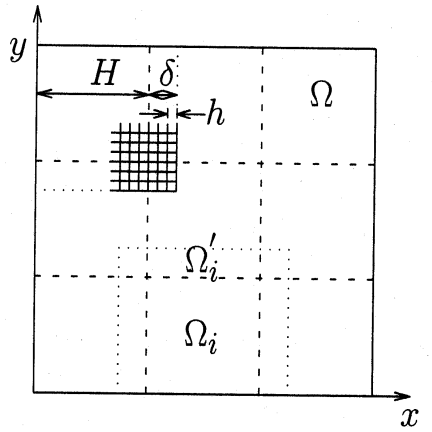
Resource scaling for PDEs

- **For 3D problems, work is proportional to four-thirds power of memory, because**
  - **for equilibrium problems, work scales with problem size times number of iteration steps -- proportional to resolution in single spatial dimension**
  - **for evolutionary problems, work scales with problems size times number of time steps -- CFL arguments place latter on order of spatial resolution, as well**
- **Proportionality constant can be adjusted over a very wide range by both discretization (high-order implies more work per point and per memory transfer) and by algorithmic tuning**
- **Machines designed for PDEs can be "memory-thin"**
- **If frequent time frames are to be captured, other resources -- disk capacity and I/O rates -- must both scale linearly with work, more stringently than for memory.**

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.

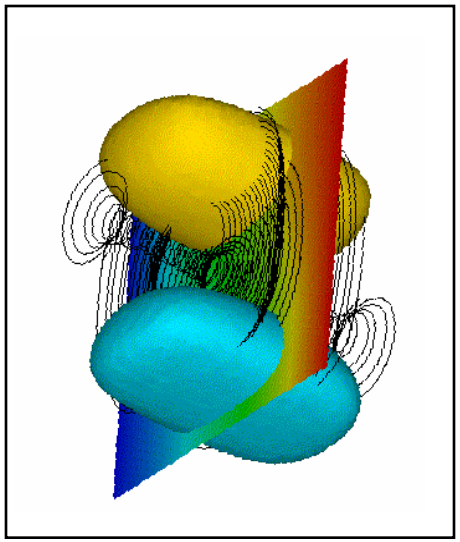
- **Theory still requires "sufficiently fine" coarse mesh**
  - **However, coarse space need *not* be nested in the fine space or in the decomposition into subdomains**
- **Practice is better than one has any right to expect**

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

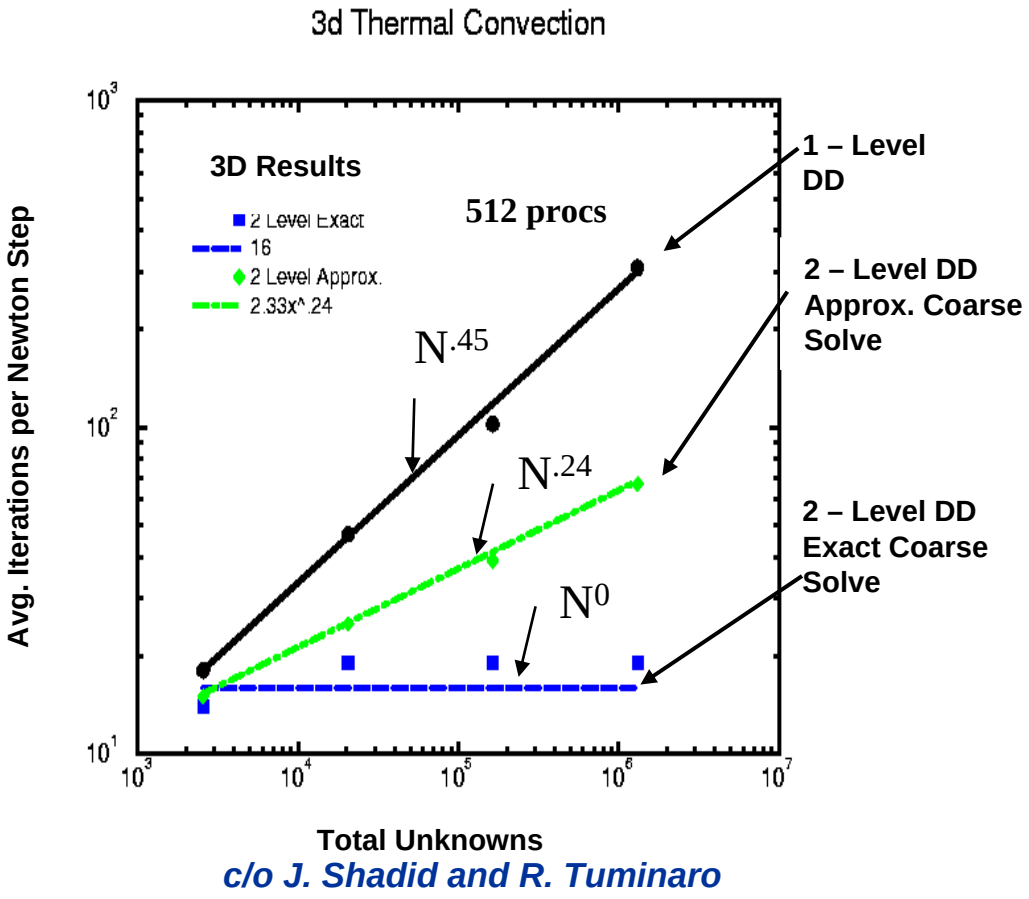
- **Wave Helmholtz (e.g., acoustics) is delicate at high frequency:**
  - **standard Schwarz Dirichlet boundary conditions can lead to undamped resonances within subdomains,** $u_\Gamma = 0$
  - **remedy involves Robin-type transmission boundary conditions on subdomain boundaries,** $(u + \alpha\, \partial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

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)

Sandia National Laboratories

"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 $a_i = r_i A r_i^T$

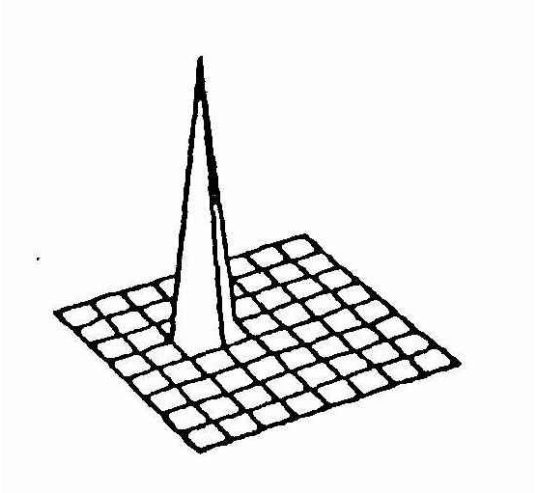
  Then
  $$A = \Sigma_i r_i^T a_i r_i$$

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

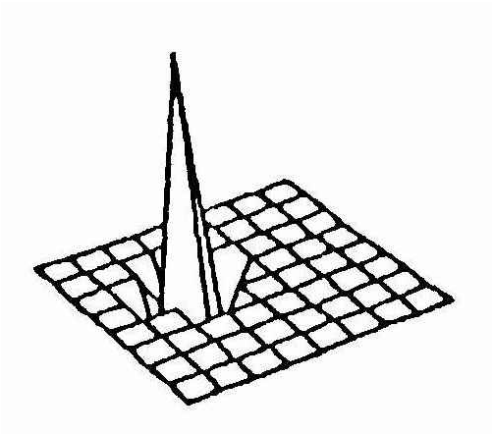
  — the Schwarz formula!

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

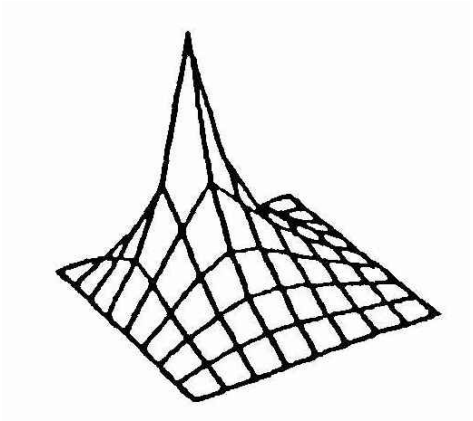
"Unreasonable effectiveness" of Schwarz, cont.



Delta function, $\delta(x)$



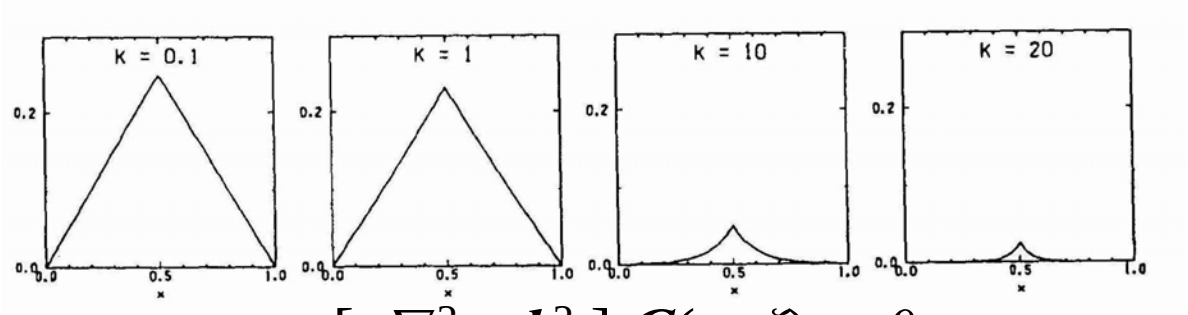
$A\ \delta(x)$



$A^{-1}\ \delta(x)$

- **Forward Poisson operator is localized and sparse**
- **Inverse operator is *locally concentrated*, but dense**
- **A coarse grid is necessary (and sufficient, for good conditioning) to represent the coupling between a field point and its forcing coming from nonlocal regions**

"Unreasonable effectiveness" of Schwarz, cont.



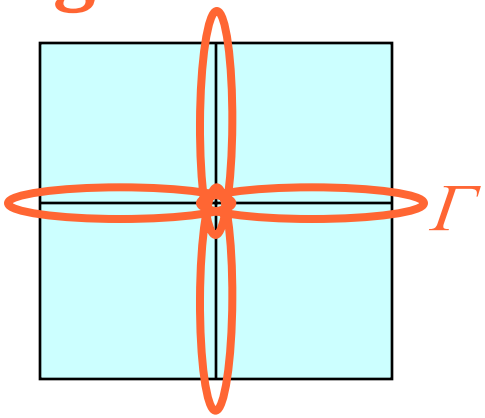
$$[\, -\nabla^2 + \boldsymbol{k}^2 \,]\, \boldsymbol{G}(\boldsymbol{x}, \xi) = 0$$

- **Green's functions for the "good Helmholtz" operator on the unit interval, shown with four increasing diagonal shifts, for ξ = 0.5**
- **It is intuitively clear why the diagonally dominant case is easy to precondition without a coarse grid**
- **This corresponds to the implicitly differenced parabolic system, and arises commonly 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:**

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

- **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**

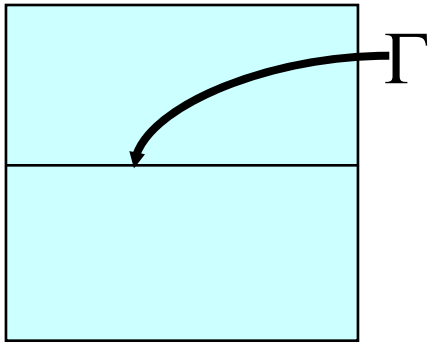
$$A^{-1} = \begin{bmatrix} I & A_{ii}^{-1} A_{i\Gamma} \\ 0 & S \end{bmatrix}^{-1} \begin{bmatrix} A_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix}^{-1}$$

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

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**

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

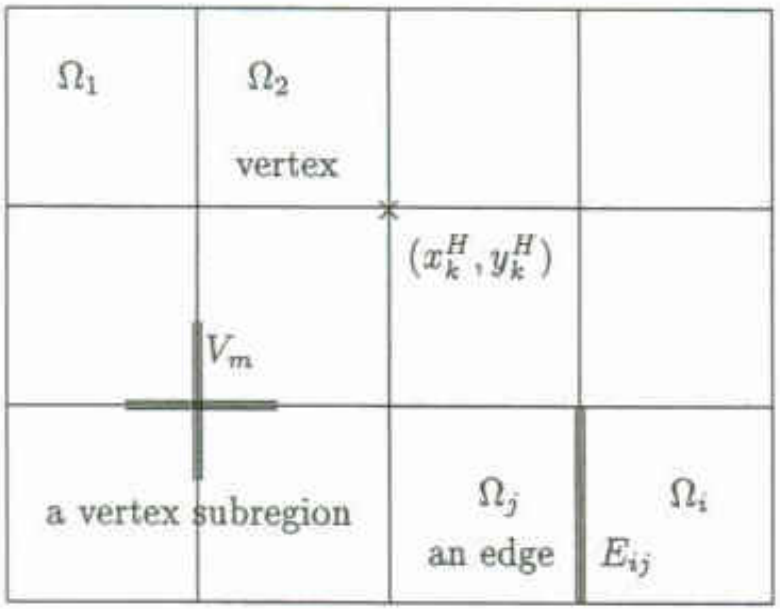
- **Balancing Neumann-Neumann**

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

- ***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)**

Overarching scaling question:

- **Can we scale to $P = 10^5$ and beyond, as required by the DOE petascale roadmaps?? It depends….**
- **The answer is strongly tied to the number of gridpoints per processor (surface-to-volume ratio)** [Fox et al., 1988, Gustafson et al. 1988 (1st Gordon Bell Pr.)]
- **For the mesh-based PDE/lattice solves under consideration,**

$$N/P \sim 1000\text{—}10000 \text{ points per processor}$$

is sufficient, given current day parameters.

- ➔ $10^8 - 10^9$, or $512^3 - 1024^3$
  is the minimum number of points to scale to $P = 10^5$

Scaling meshed-based codes to petaflop/s

- **Three issues —**
  - **Parallelism — scaling to $P = 10^5$**
  - **Algorithmic scaling —**
    - **Linear system solves**
    - **Discretizations**
  - **Single node performance**

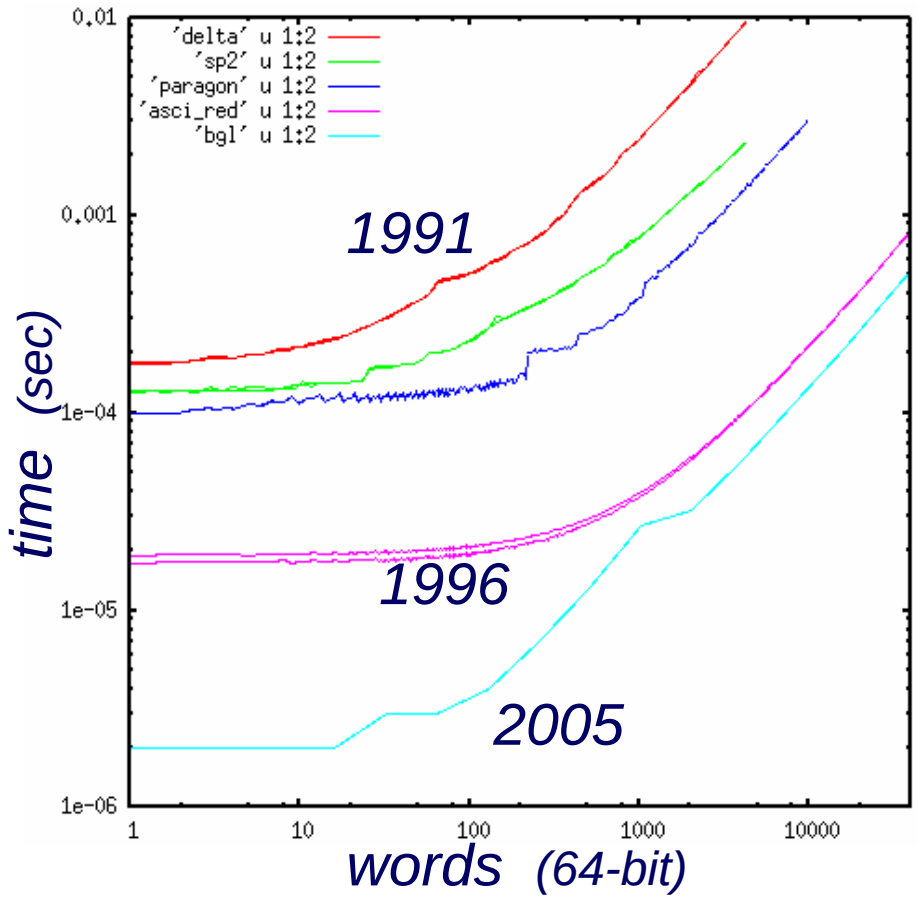
Model assumptions

- *P-processor solution time:*
  - *Non-overlapping comp./comm:* $T_P = T_A(P) + T_C(P)$
    - *maximum time savings with overlap is 2x*
    - *pales in comparison to 100,000x from P-fold parallelism.*
- $T_A(P) = T_A(1)/P$ *— total arithmetic cost*
  - *assumes perfect load balance, etc. for local work.*
- $T_C(P)$ *— total communication cost:*

  Assume linear message cost: $t_c(m) = (\alpha + \beta m) * t_a$
  - *m = number of 64-bit words*
  - t_a *= representative (observable) time for c=a*b*
  - α *= nondimensional latency* ($\alpha := \alpha^* / t_a$)
  - β *= nondimensional inverse-bandwidth* ($\beta := \beta^* / t_a$)

Communication performance

- **Communication performance has improved by orders of magnitude over the years…**



Linear communication model :

$t_c(m) = \alpha^* + \beta^* m$, *m: 64-bit words*

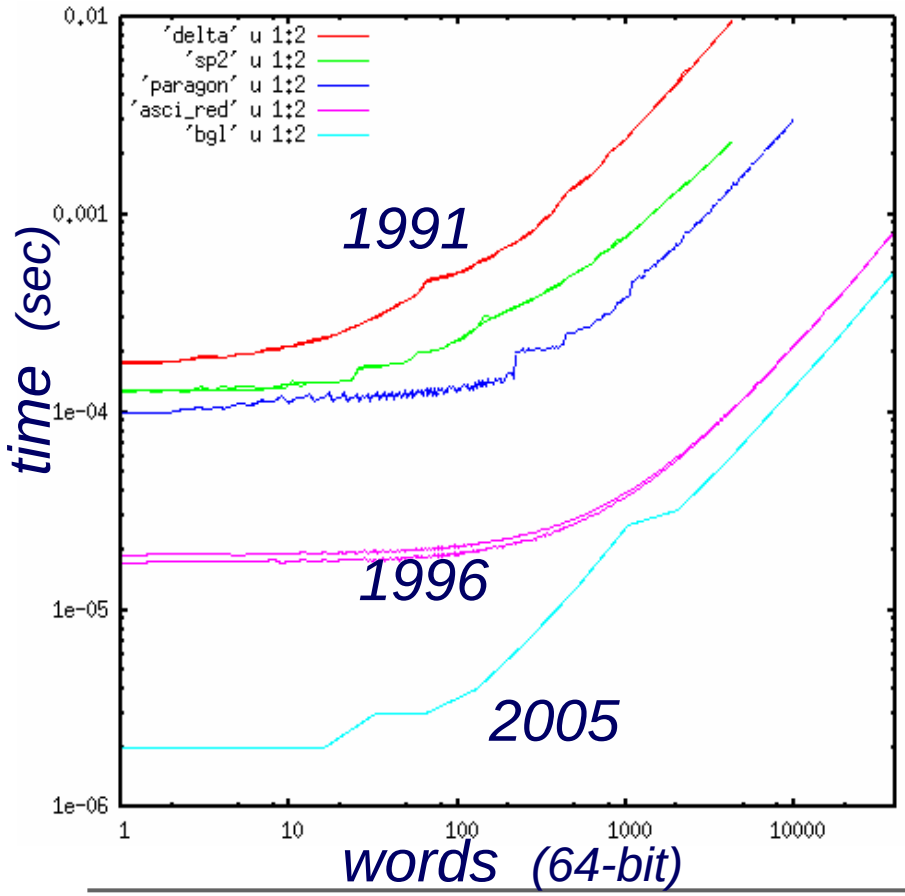
Nondimensionalize by t_a [$c = a*b$] :

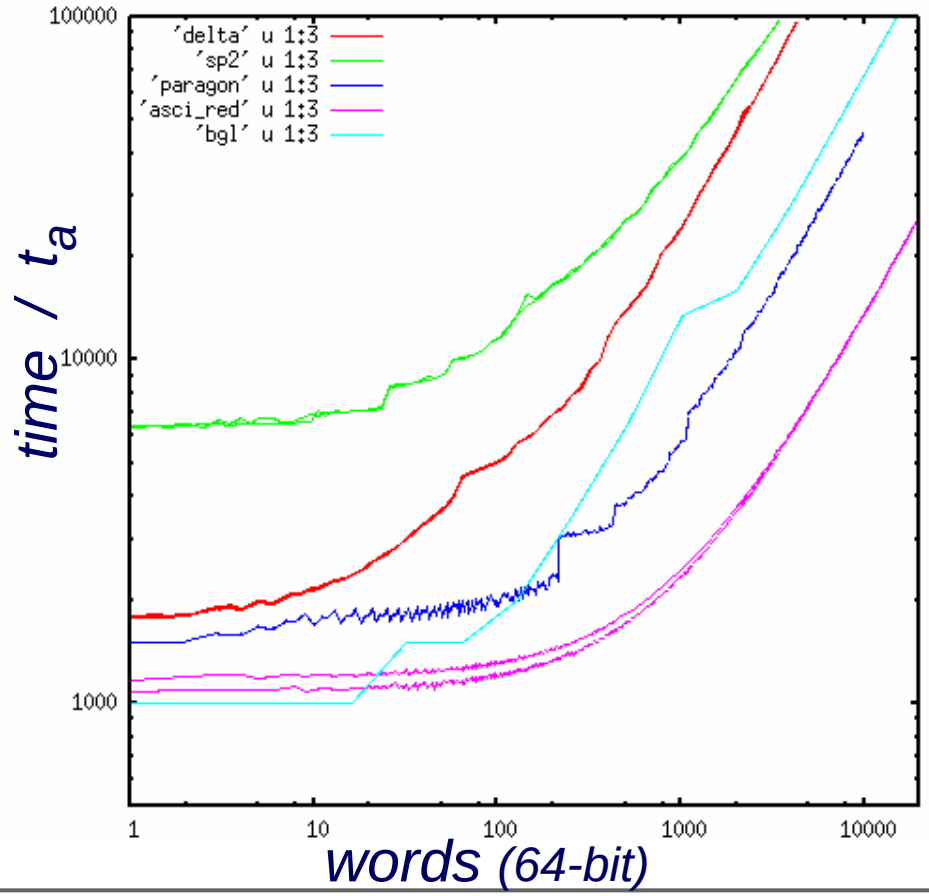
$$t_c(m) = (\alpha + \beta m)\, t_a$$

$$\alpha = \alpha^* / t_a\,, \quad \beta = \beta^* / t_a$$

Communication performance

- **Communication performance has improved by orders of magnitude over the years**
- **Computational rates have also improved… (*reduced* t_a)**
- **Net result is that programming models that were valid 20 years ago are still valid today, and will likely remain so (for many reasons….)**





History of nondimensional machine parameters

YEAR	t_a (μs)	α^*	β^*	α	β	m_2	MACHINE
1986	50.00	5960.	64	119.2	1.3	93	Intel iPSC-1 (286)
1987	.333	5960.	64	18060	192	93	Intel iPSC-1/VX
1988	10.00	938.	2.8	93.8	.28	335	Intel iPSC-2 (386)
1988	.250	938.	2.8	3752	11	335	Intel iPSC-2/VX
1990	.100	80.	2.8	800	28	29	Intel iPSC-i860
1991	.100	60.	.80	600	8	75	Intel Delta
1992	.066	50.	.15	758	2.3	330	Intel Paragon
1995	.020	60.	.27	3000	15	200	IBM SP2 (BU96)
1996	.016	30.	.02	1800	1.25	1500	ASCI Red 333
1998	.006	14.	.06	2300	10	230	SGI Origin 2000
1999	.005	20.	.04	4000	8	375	Cray T3E/450
2005	.002	2.	.013	1000	6.5	154	BGL/ANL

- $m_2 := \alpha / \beta$ ~ message size → twice cost of single-word message
- t_a based on matrix-matrix products of order 10—13

Three definitions of "efficiency"

- **For any iterative method,**

$$time = (\#\, of\ iters) \times (time\ per\ iter)$$

- **"Convergence efficiency"**

$$\eta_{\text{conv}} = \frac{(\#\, of\ iters\ for\ P_{\text{small}})}{(\#\, of\ iters\ for\ P_{\text{large}})}$$

- **"Implementation efficiency"**

$$\eta_{\text{impl}} = \frac{(time\ per\ iter\ for\ P_{\text{small}})}{(time\ per\ iter\ for\ P_{\text{large}})} \times \frac{P_{\text{small}}}{P_{\text{large}}}$$

- **Overall efficiency**

$$\eta = \eta_{\text{conv}} \times \eta_{\text{impl}} = \frac{(time\ for\ P_{\text{small}})}{(time\ for\ P_{\text{large}})} \times \frac{P_{\text{small}}}{P_{\text{large}}}$$

- **"Ideal" is unity for all three measures**

Mesh-based algorithms

- **Consider $\nabla^2 u = f$ with finite-volume, difference, or element schemes**
  - **Point Jacobi iteration (7-point stencil):** $u_i = a_{ii} f_i + \Sigma_j \; a_{ij} u_j$
    - **— Work:** $T_{aJ} \sim 14\, N/P\, t_a$
    - **— Communication:** $T_{cJ} \sim (6 + (N/P)^{2/3} (1/m_2)) \alpha t_a$
    - **— For fixed N/P, Jacobi implementation efficiency scales independent of P.**
    - **— However, algorithmic scaling is poor – a more communication intensive approach is required:**
      - **– conjugate gradient iteration, multigrid, etc.**

Mesh-based algorithms

- **Point Jacobi iteration (7-point stencil):** $u_i = a_{ii} f_i + \Sigma_j \; a_{ij} \, u_j$
  - **Work:** $T_{aJ} \sim \; 14 \; N/P \; t_a$
  - **Communication:** $T_{cJ} \sim \; (\; 6 + \; (N/P)^{\,2/3} \; (1/\, m_2) \;) \; \alpha \, t_a$

- **Conjugate gradient iteration (7-point stencil):**
  - **Work:** $T_{aCG} \sim \; 27 \; N/P \; t_a$
  - **Communication:** $T_{cCG} \sim T_{cJ} \; + 4 \; \log_2 P \; \alpha \, t_a$

- **Multigrid-preconditioned conjugate gradient iteration:**
  - **Work:** $T_{aMG} \sim \; 42 \; N/P \; t_a$
  - **Communication:** $T_{cMG} \sim T_{cCG} + \log_2 (N/P)^{1/3} \; \alpha \, t_a$

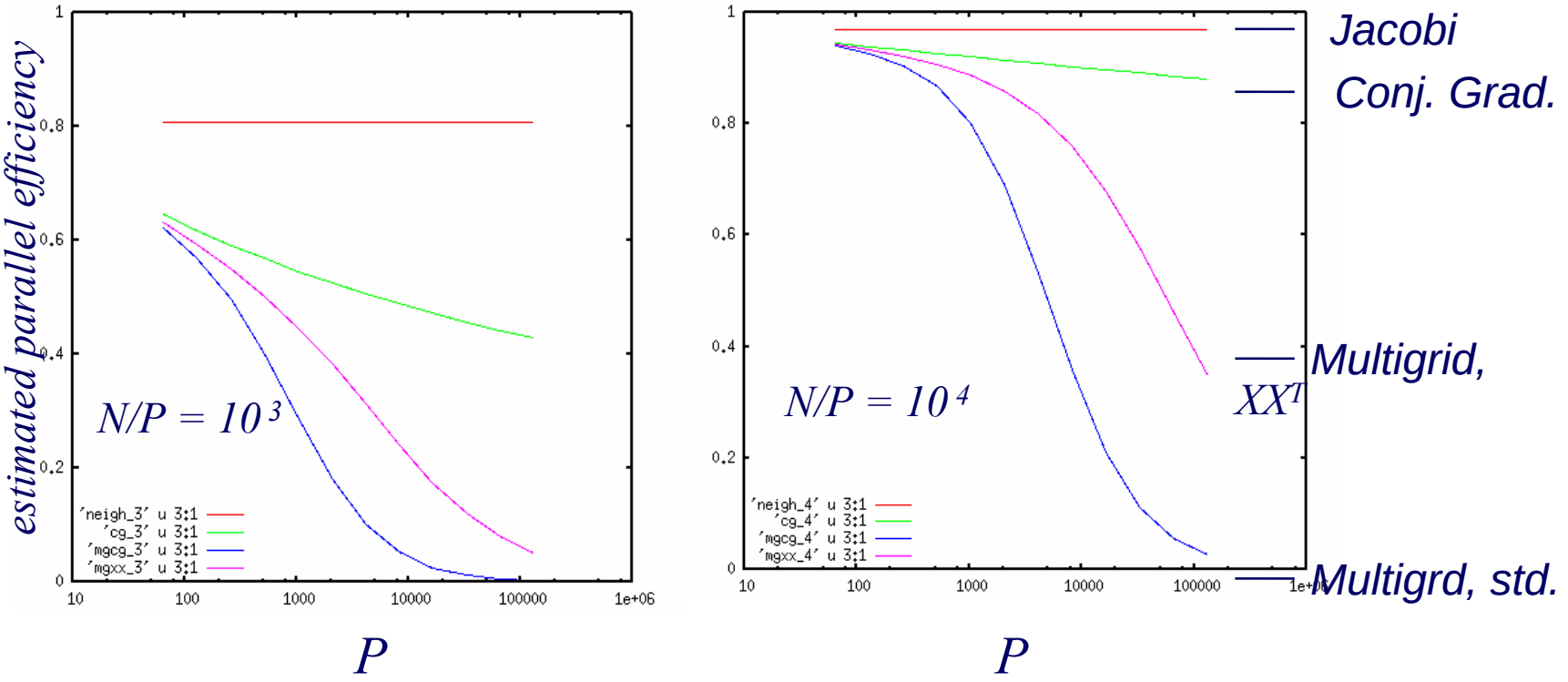
  Plus **coarse-grid solve:**

  - **Std. "fast" coarse-grid solve:** $T_{std} \; \sim 2 \; \log_2 P \; (\; 1 + P/m_2 \;) \; \alpha \, t_a$
  - $A_c^{-1} = XX^T$ **coarse-grid solve:** $T_{XXt} \; \sim 2 \; \log_2 P \; (\; 1 + 2.5/m_2 \;\; P^{2/3} \;) \; \alpha \, t_a$

(Tufo & F, JDPC 01)

Mesh-based scaled-speedup models

- **Jacobi iteration scales, *but will not scale algorithmically***
- **The inner-products for conjugate gradient do not pose a significant difficulty (on Blue Gene – a different story on seaborg…)**
- **The coarse-grid solve for multigrid can be a significant communication bottleneck, unless fast algorithms are used.**



Scaling conclusions

- **Scaling to 10^5 processors is possible:**
  - **— provided # of points/proc. ~ 10^3 – 10^4**
  - **— assuming current-day communication parameters**
  - **— applies to nearest-neighbor algorithms and, remarkably,**

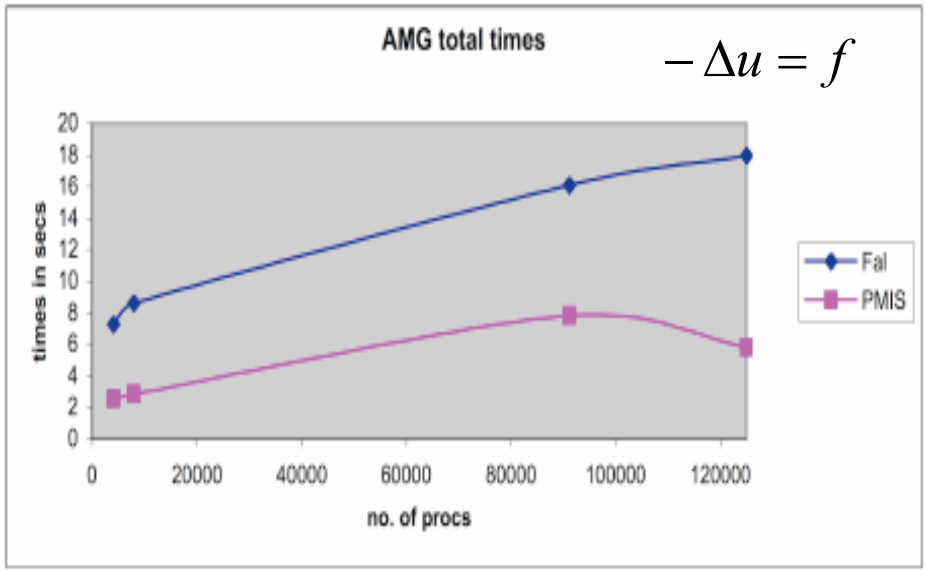
    to all-to-all algorithms, given sufficient minimum bisection bandwidth (3D interconnect suffices)
  - **— problem-dependent concerns:**

    mesh topology, locality of boundary conditions, I/O, etc.

Solvers *are* scaling: algebraic multigrid (AMG) on BG/L (hypre)

- Algebraic multigrid a key algorithmic technology
  - Discrete operator defined for finest grid by the application, itself, *and* for many recursively derived levels with successively fewer degrees of freedom, for solver purposes
  - Unlike geometric multigrid, AMG not restricted to problems with "natural" coarsenings derived from grid alone
- Optimality (cost per cycle) intimately tied to the ability to coarsen aggressively
- Convergence scalability (number of cycles) and parallel efficiency also sensitive to rate of coarsening
- While much research and development remains, multigrid will clearly be practical at BG/P-scale concurrency

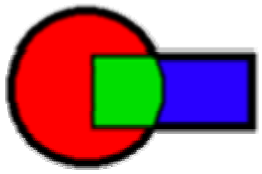
Figure shows weak scaling result for AMG out to 120K processors, with one 25×25×25block per processor (up to 1.875B dofs)



7-pt Laplacian, total execution time, AMG-CG, total problem size ~2 billion

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 (in SciDAC) 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)

More on domain decomposition

- **18th International Conference**
  - **January 12th to 17th 2008**
  - **Jerusalem (Hebrew University)**
- **Web home**
  - **ddm.org**
  - **Freely downloadable papers, bibtex resources, scientific contacts**