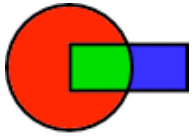
Domain Decomposition Methods and High Performance Computing



David Keyes
Department of Applied Physics and Applied Mathematics
Columbia University

Plan of presentation

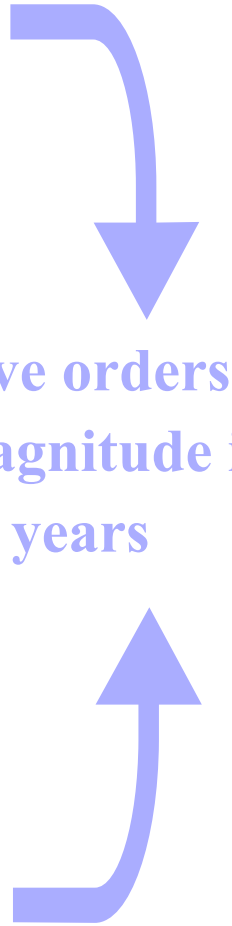
- ***Glimpse*** **at intertwined history of high performance computing and domain decomposition algorithms**
- ***Mention*** **three programming paradigms for domain decomposition methods**
- ***Peek*** **at the algorithmic motivation for "standard" bulk synchronous implicit parallel implementations based on additive subspace corrections**
  - **definitions of scalability**
  - **a family of techniques that *weak-scales indefinitely* (for multirate problems with exploitable scale separation) in the Bulk Synchronous Programming (BSP) paradigm**

Domain decomposition and parallel computing

1985		Intel iPSC/1, Transputer, Alliant FX/8, nCUBE, ICL DAP Linda
1986		CM-1, Multimax, Cray X-MP, FPS Hypercube
1987	Paris	ETA-10, CM-2 Multiflow, Ametek isoefficiency, first Gordon Bell Prize, LAPACK
1988	Los Angeles	Convex C2, Cray Y-MP, iPSC/2, SGI Power, Tera inspector/executor model
1989	Moscow	nCUBE/2 Bulk Synchronous Programming, CRPC
1990	Houston	NEC SX-3, iPSC/860, MasPar MP-1 PVM
1991	Norfolk	Cray C90, CM-200, CM-5, Intel Delta FORTRAN-D, World Wide Web, CORBA
1992	Como	KSR-1 MasPar MP-2 MPI
1993	University Park	Intel Paragon, Numerical Wind Tunnel, Cray T3D Mosaic, Legion, first Top500 list

Gordon Bell Prize "peak performance"

Year	*Type*	*Application*	*No. Procs*	*System*	*Gflop/s*
1988	PDE	Structures	8	Cray Y-MP	1.0
1989	PDE	Seismic	2,048	CM-2	5.6
1990	PDE	Seismic	2,048	CM-2	14
1992	NB	Gravitation	512	Delta	5.4
1993	MC	Boltzmann	1,024	CM-5	60
1994	IE	Structures	1,904	Paragon	143
1995	MC	QCD	128	NWT	179
1996	PDE	CFD	160	NWT	111
1997	NB	Gravitation	4,096	ASCI Red	170
1998	MD	Magnetism	1,536	T3E-1200	1,020
1999	PDE	CFD	5,832	ASCI BluePac	627
2000	NB	Gravitation	96	GRAPE-6	1,349
2001	NB	Gravitation	1,024	GRAPE-6	11,550
2002	PDE	Climate	5,120	Earth Sim	26,500
2003	PDE	Seismic	1,944	Earth Sim	5,000
2004	PDE	CFD	4,096	Earth Sim	15,200
2005	MD	Solidification	131,072	BG/L	101,700
2006	MD	Elec. Struct.	131,072	BG/L	207,000



Five orders of magnitude in 17 years

Parallelism in Domain Decomposition proceedings

- Paris, France, 1987
- Los Angeles, USA, 1988
- Houston, USA, 1989
- Moscow, USSR, 1990
- Norfolk, USA, 1991
- Como, Italy, 1992
- University Park, USA, 1993
- Beijing, China, 1995
- Ullensvang, Norway, 1996
- Boulder, USA, 1997
- Greenwich, UK, 1998
- Chiba, Japan, 1999
- Lyon, France, 2000
- Cocoyoc, Mexico, 2002
- Berlin, Germany, 2003
- New York, USA 2005
- Strobl, Austria, 2006
- Jerusalem, Israel, 2008
- Zhangjiajie, China, 2009

No parallel results in DD1 proceedings though the motivation is obvious

DD2-DD9 were generally organized into four sections:

I. Theory

II. Algorithms

III. Parallel Implementations

IV. Applications

From DD10 onwards, parallelism is more or less taken for granted and plays no organizing role in proceedings

Yale's Intel iPSC-1: 128 80286 processors connected through ethernet controllers = 1.536 Gflop/s

On the floor in the Yale CS machine room in Fall 2005 – the world's largest Intel iPSC configuration at that time.

Had to be manually rebooted hourly or more often

Thread-concurrency: 128 nodes programmed using Intel's "NX"

My first DD paper (with W. D. Gropp)

- The Bramble-Pasciak-Schatz preconditioner

$$M_2^{-1} = \sum_{i=1}^{m} R_{E_i}^T S_{E_i E_i}^{-1} R_{E_i} + R_H^T A_H^{-1} R_H$$

Theorem 6 There exists a constant C independent of H, h such that

$$\text{cond}\,(M_2^{-1} S) \leq C \left(1 + \log^2(H/h)\right).$$

In case the coefficients a are constant in each subdomain Ω_i, then C is also independent of a.

Proof. See Bramble *et al.* (1986a), Widlund (1988) and Dryja *et al.* (1993).

My first DD paper (with W. D. Gropp)

- Parallel numerical tests: c.c. Poisson on unit square (many subdomains)
- Dryja preconditioner on each interface
- With vertex coupling

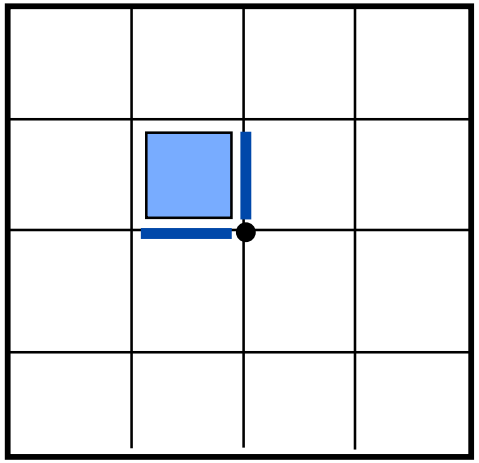


TABLE 13

$\nabla^2 u = f$ in the unit square, divided into equal boxes. Results for PMM with M_D on the interfaces and with vertex coupling after Bramble et al., as a function of problem size and number of processors.

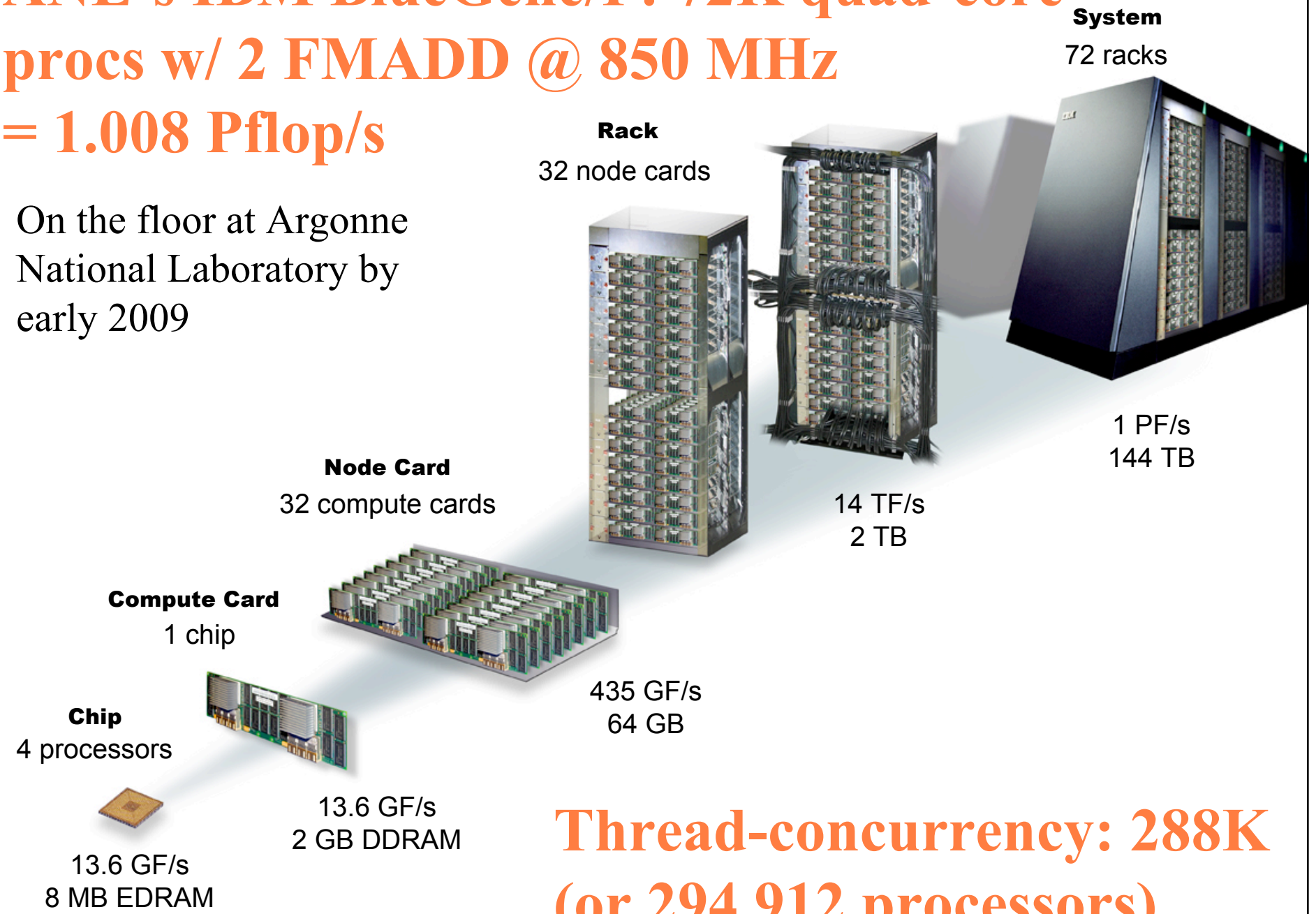
h^{-1}		$p=1$	4	16	64
16	I	1	6		
	κ	1.00	5.74		
	ρ	5.0 (−6)	1.3 (−1)		
	T	4.85	4.28		
	s	—	*1.13*		
32	I	1	5	7	
	κ	1.00	7.88	7.01	
	ρ	1.7 (−5)	1.5 (−1)	2.6 (−1)	
	T	25.0	16.6	7.39	
	s	—	*1.51*	*2.25*	
64	I		6	7	6
	κ		10.7	10.2	7.16
	ρ		1.5 (−1)	2.5 (−1)	2.1 (−1)
	T		95.1	24.7	16.7
	s		—	*3.85*	*1.48*
128	I			7	7
	κ			14.1	10.5
	ρ			2.5 (−1)	2.4 (−1)
	T			111	36.3
	s			—	*3.06*
256	I				8
	κ				14.5
	ρ				2.7 (−1)
	T				139
	s				—

4×4 proc mesh

64×64 mesh

ANL's IBM BlueGene/P: 72K quad-core procs w/ 2 FMADD @ 850 MHz = 1.008 Pflop/s

On the floor at Argonne National Laboratory by early 2009



Thread-concurrency: 288K (or 294,912 processors)

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

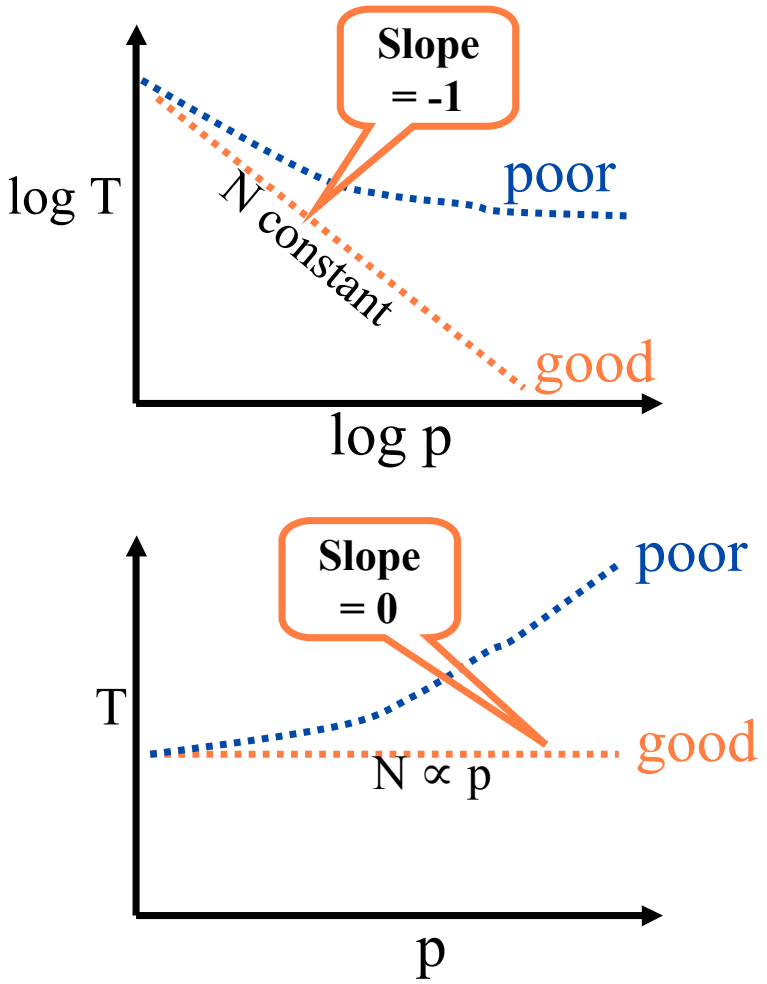
- **Goal is algorithmic scalability:**

  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

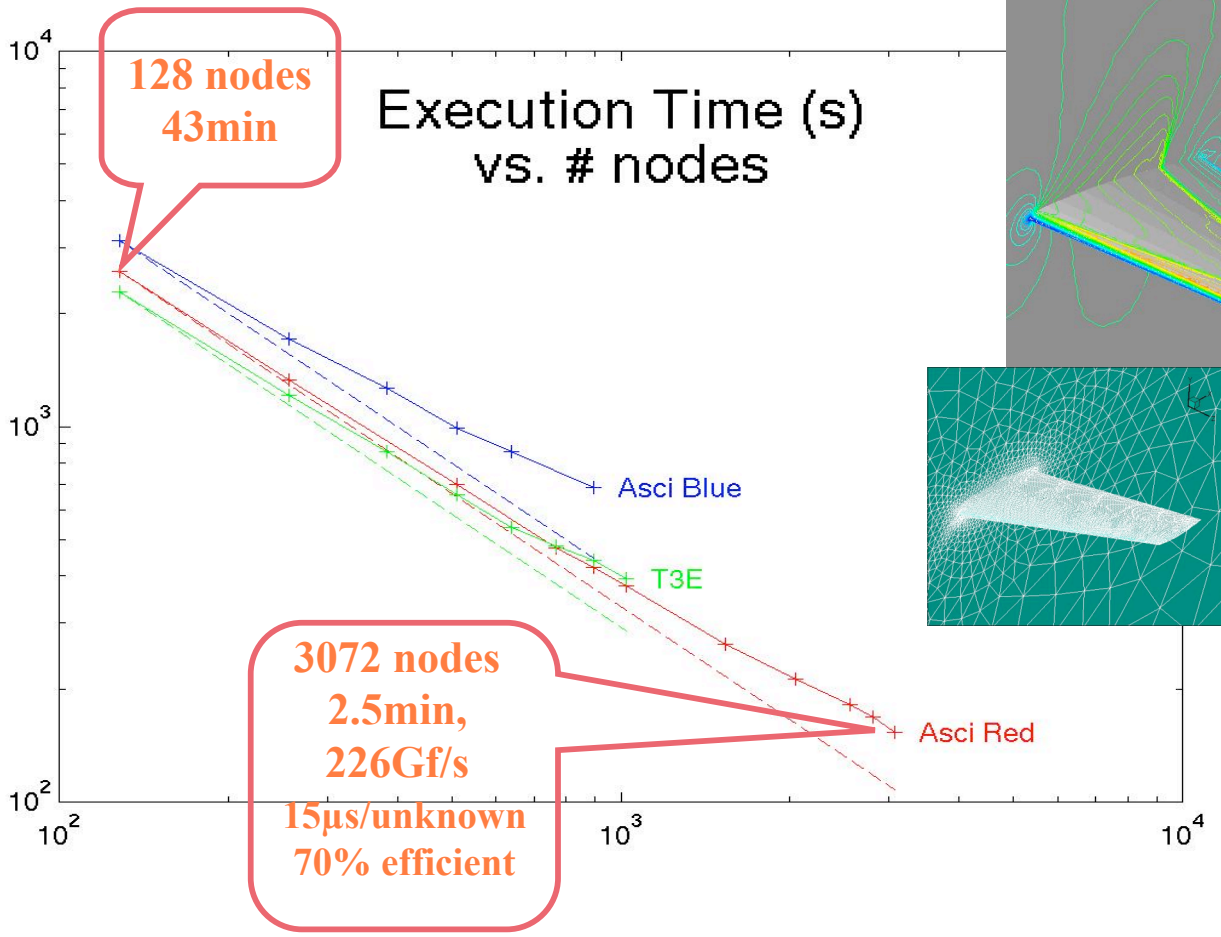
Review: two definitions of scalability

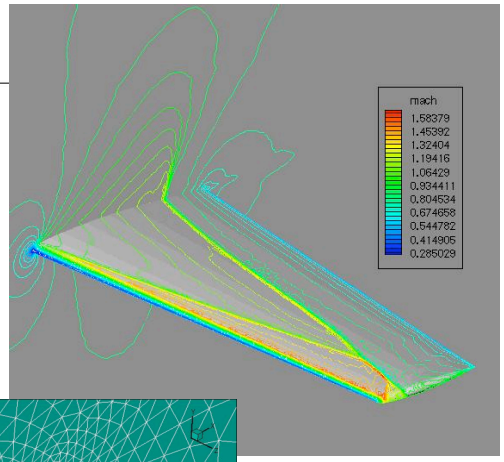
- "Strong scaling"
  - execution time decreases in inverse proportion to the number of processors
  - *fixed size problem overall*
  - often instead graphed as reciprocal, "speedup"
- "Weak scaling"
  - execution time remains constant, as problem size and processor number are increased in proportion
  - *fixed size problem per processor*
  - also known as "Gustafson scaling"



Strong scaling illus. (1999 Bell Prize)

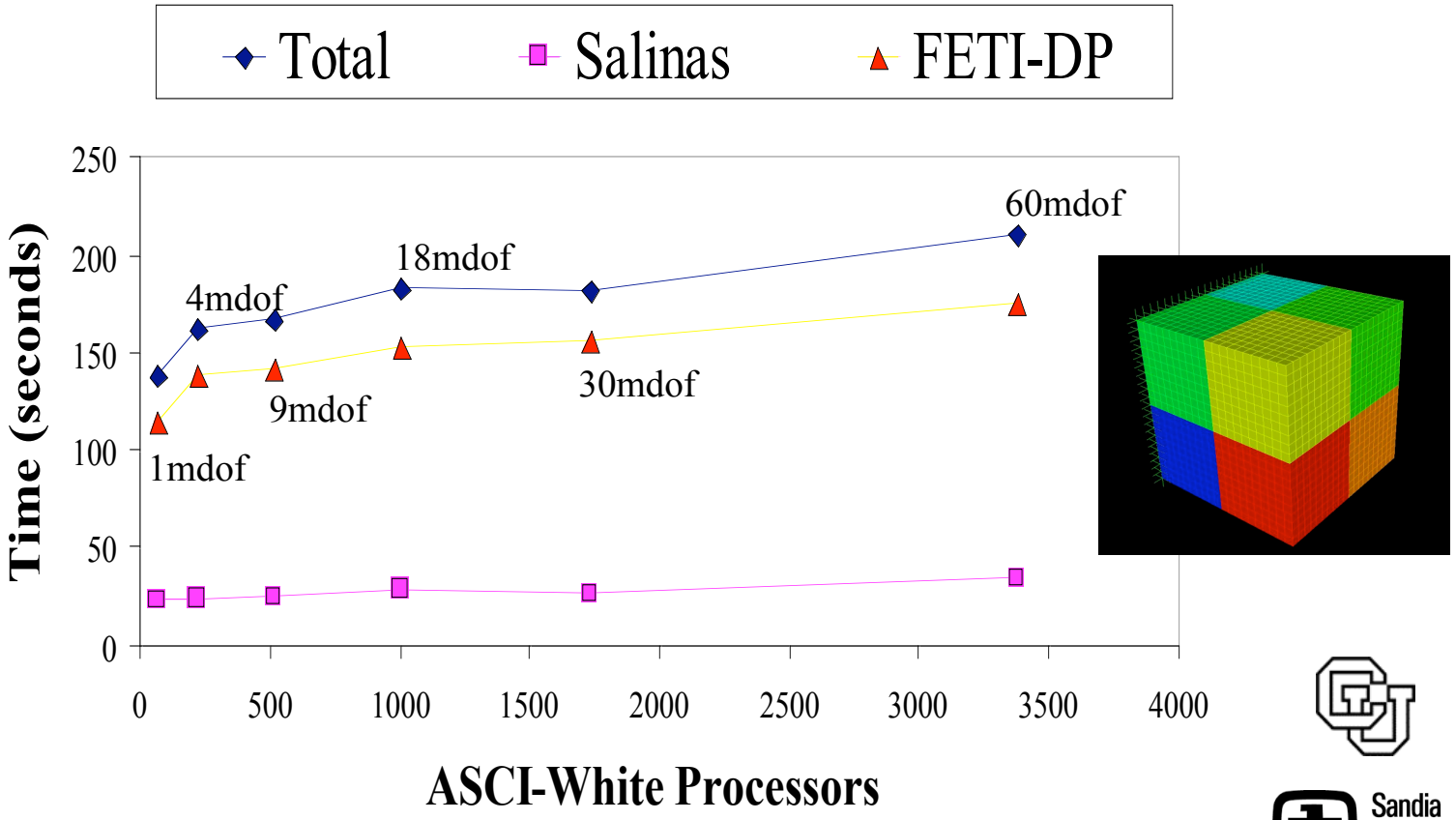
- Newton-Krylov-Schwarz (NKS) algorithm for compressible and incompressible Euler and Navier-Stokes flows
- Used in NASA application FUN3D (M6 wing results below with 11M dof)





Weak scaling illus. (2002 Bell Prize)

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



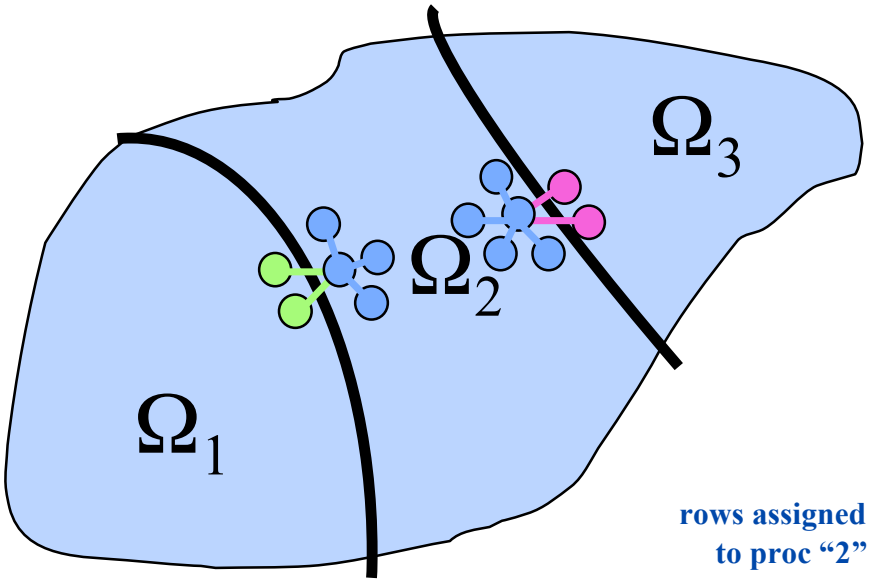
Sandia National Laboratories

c/o C. Farhat, Stanford

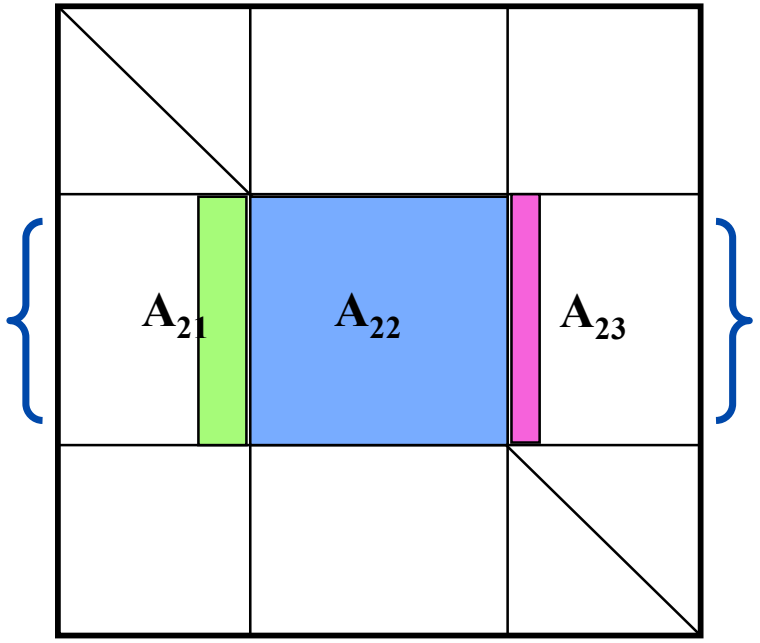
Parallel programming paradigms for domain decomposition

- **Multiple Instruction Multiple Data (MIMD)**
  - different processors specialize to different steps
  - example: Schur complement method in which subdomain condensation and interface systems are handled separately and Schur complements shipped between
- **Single Program Multiple Data (SPMD) Bulk Synchronous Programming**
  - all processors run the same code on a different subdomain and cycle between synchronized computation and communication phases
  - example: vast majority of all PDE-based parallel computing, including all codes run with TOPS software (PETSc, hypre, SUNDIALS, Trilinos, etc.), on dedicated, tightly-coupled resources
- **Semi-asynchronous programming**
  - groups of processes run mostly independently with infrequent synchronization
  - example: Grid-type computing, natural when jobs are hard to balance, e.g., nonlinear Schwarz (ASPIN) and cycles scavenged

SPMD parallelism w/domain decomposition puts off limitation of Amdahl in weak scaling

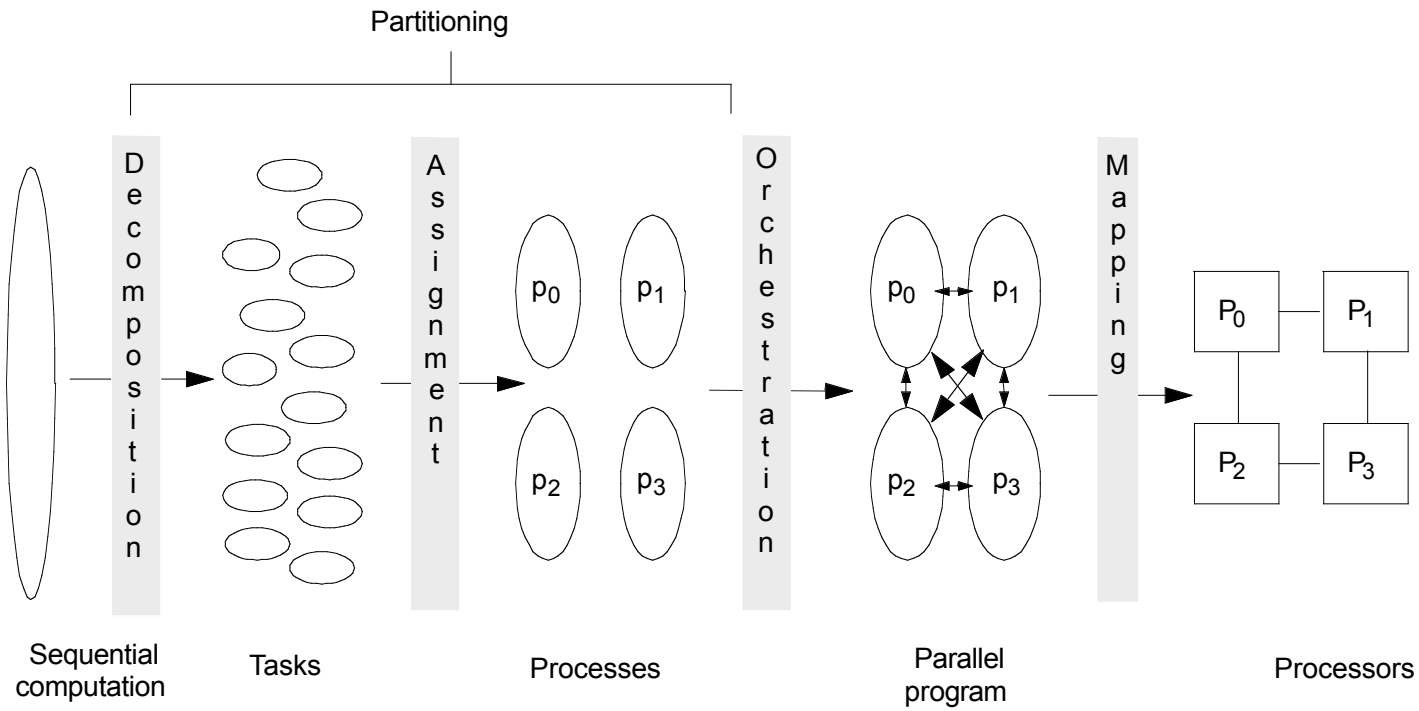


rows assigned
to proc "2"



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

Four steps in creating a parallel program



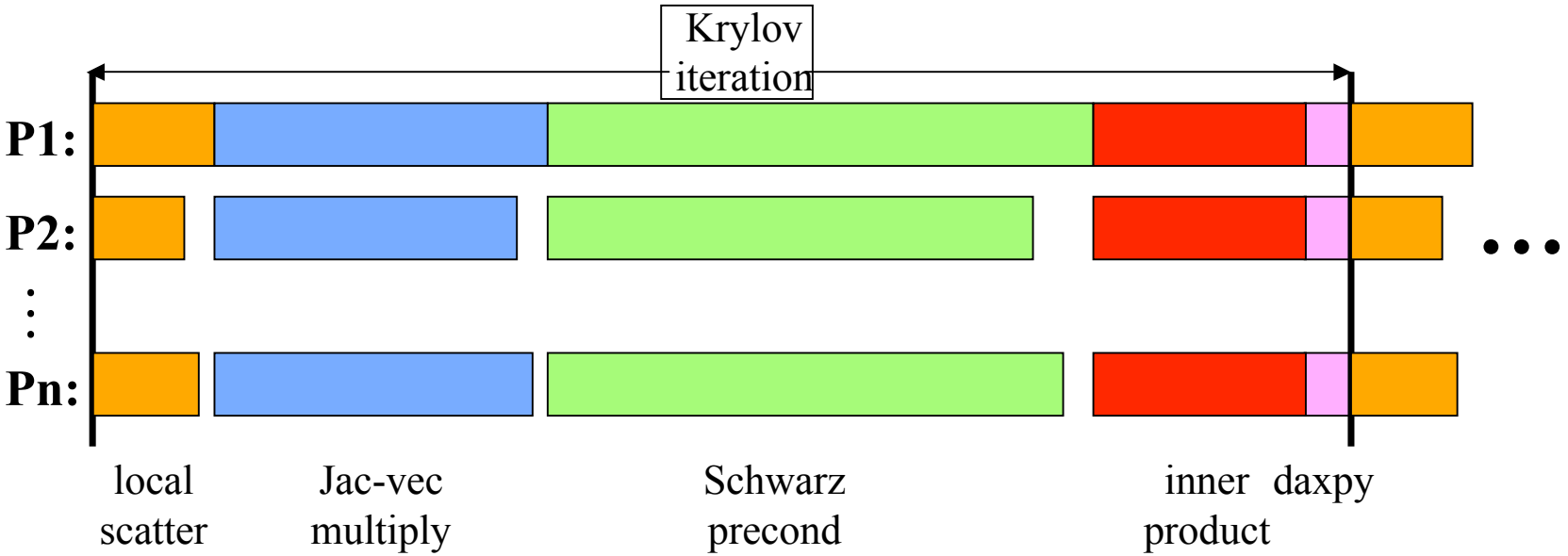
- **Decomposition of computation in tasks**
- **Assignment of tasks to processes**
- **Orchestration of data access, communication, synchronization**
- **Mapping processes to processors**

c/o D. E. Culler, Berkeley

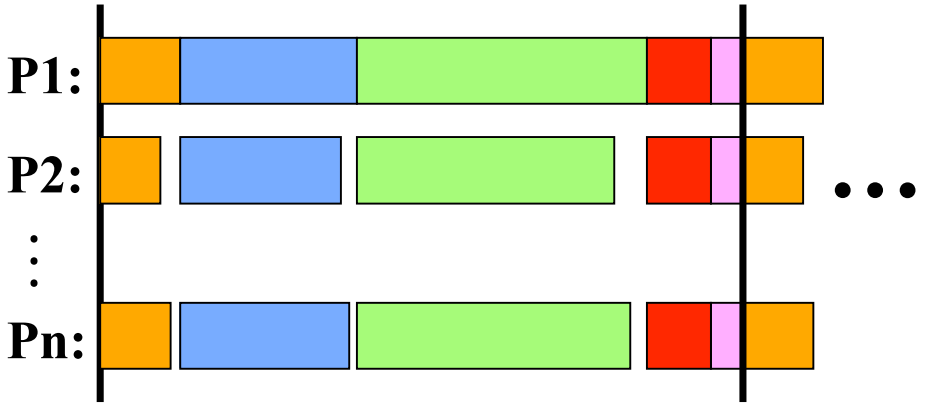
Krylov-Schwarz parallelization summary

- **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.



"Scalable" includes "optimal"

- "Optimal" for a theoretical numerical analyst means a method whose floating point complexity grows at most linearly in the data of the problem, N, or (more practically and almost as good) linearly times a polylog term
- For iterative methods, this means that the *cost per iteration* must be at most $O(N \log^p N)$ and the *number of iterations* must be at most $O(\log^p N)$
- Cost per iteration must include communication cost as processor count increases in weak scaling, $P \propto N$
  - BlueGene permits this with its log-diameter global reduction
- Number of iterations comes from condition number for linear iterative methods; Newton's superlinear convergence is important for nonlinear iterations

Why optimal algorithms?

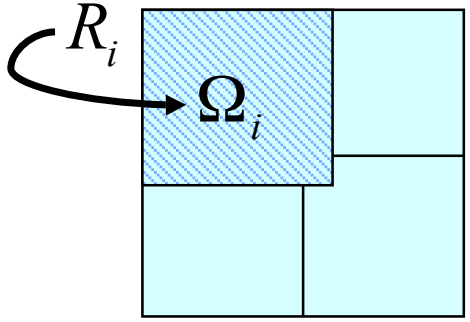
- **The more powerful the computer, the *greater* the importance of optimality**
  - **though the counter argument is often employed ☹**
- **Example:**
  - **Suppose *Alg1* solves a problem in time $C\,N^2$, where N is the input size**
  - **Suppose *Alg2* solves the same problem in time $C\,N\log_2 N$**
  - **Suppose *Alg1* and *Alg2* parallelize *perfectly* on a machine of 1,000,000 processors**
- **In constant time (compared to serial), *Alg1* can run a problem 1,000 X larger, whereas *Alg2* can run a problem about 50,000 X larger**

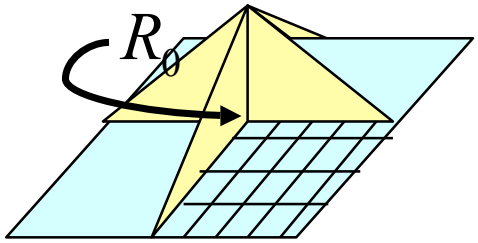
Schwarz domain decomposition (DD) method

- 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

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

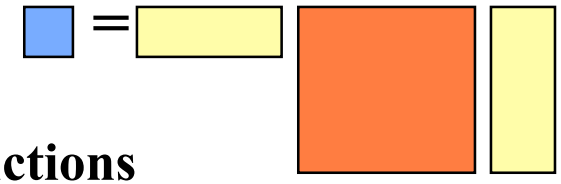
$$B^{-1} = R_0^T A_0^{-1} R_0 + \sum_i R_i^T A_i^{-1} R_i$$





- 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)

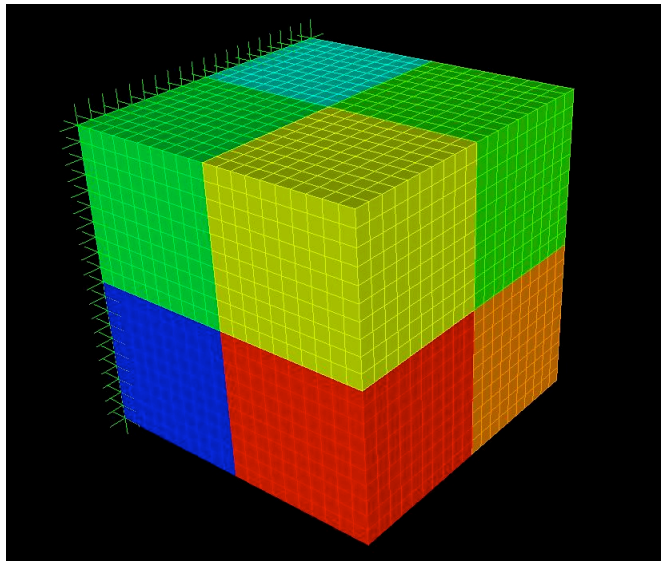
$$A_i = R_i A R_i^T$$



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

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

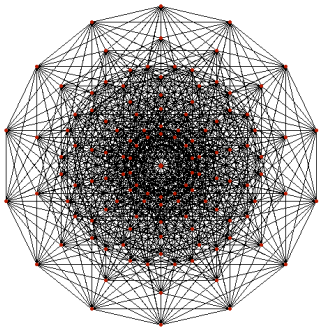
- **limit of infinite neighbor bandwidth, zero neighbor latency ($B \to 0$)**

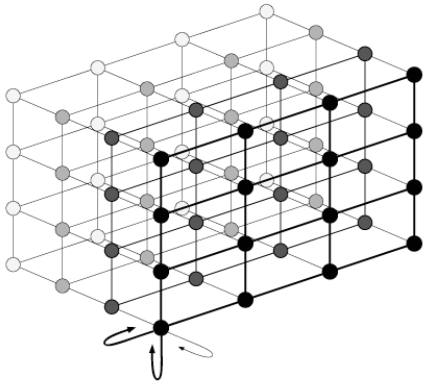
$$T(n, p_{opt}(n)) = C\left[\log n + \frac{1}{3}\log\frac{A}{C} + const.\right]$$

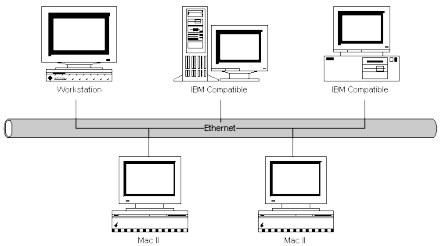
(This analysis is on a per iteration basis; complete analysis multiplies this cost by an iteration count estimate that generally depends on *n* and *p*.)

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")

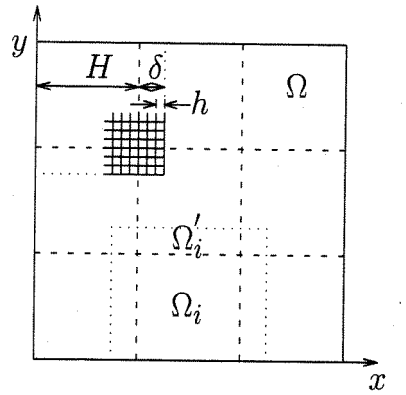






Factoring convergence rate into estimates

- 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, $\kappa \leq \omega\,[1+\rho(\varepsilon)]\,C_0^2$**
  - **C_0^2 is a splitting constant for the subspaces of the decomposition**
  - **$\rho(\varepsilon)$ 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**

Onward to nonlinearity

- **Linear versus nonlinear problems**
  - **Solving linear systems often constitutes 90% of the running time of a large PDE simulation**
  - **The nonlinearity is often a fairly straightforward outer loop, in that it introduces no new types of messages or synchronizations not present in Krylov-Schwarz, 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 leave Newton-outside (NKS) to Xiao-Chuan and comment on Newton-inside methods in the parallel context**

Nonlinear Schwarz

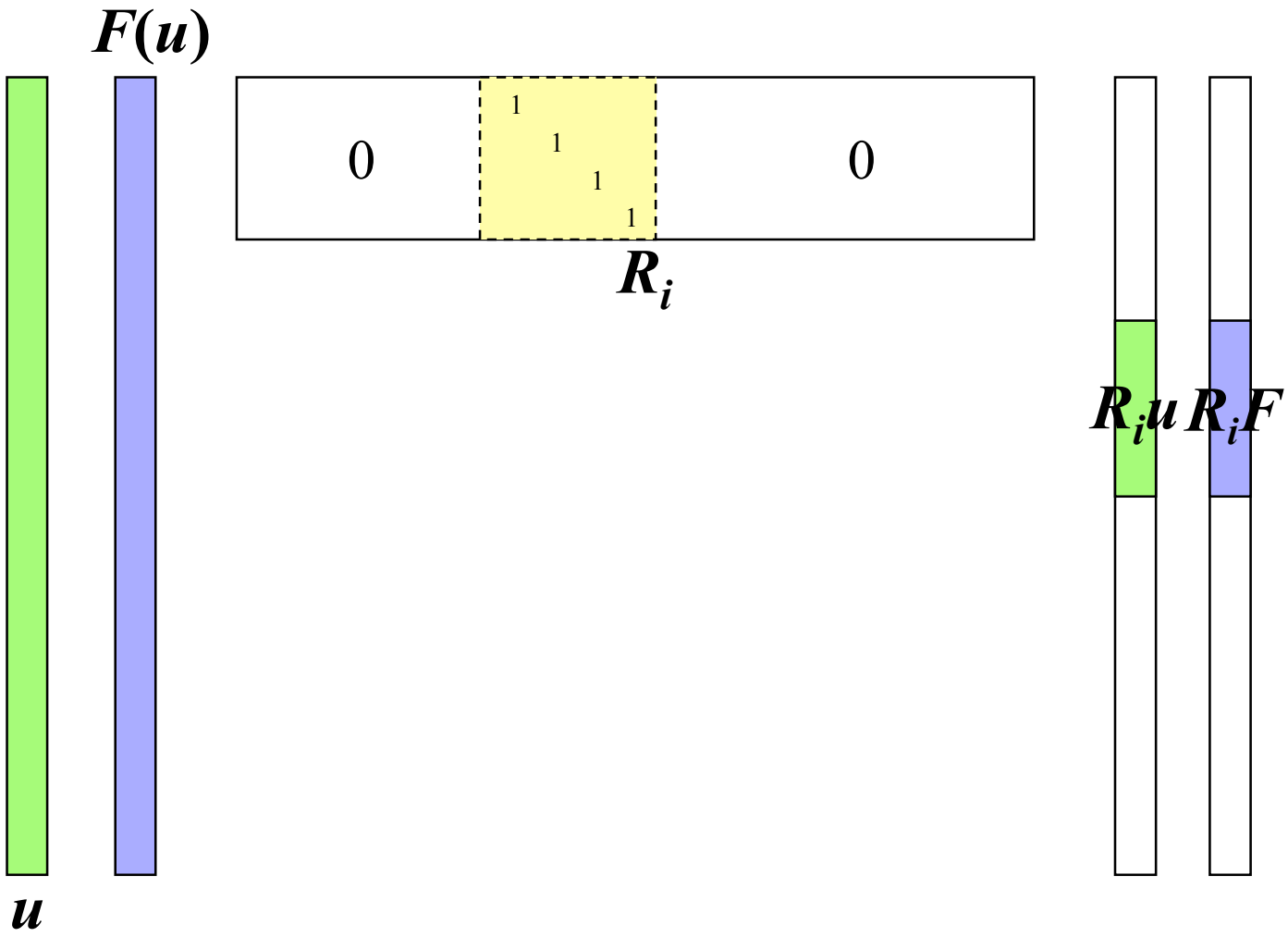
- **Nonlinear Schwarz has Newton both *inside* and *outside* and is fundamentally Jacobian-free**
- **It replaces $F(u) = 0$ with a new nonlinear system possessing the same root, $\Phi(u) = 0$**
- **Define a correction $\delta_i(u)$ to the i^{th} partition (e.g., subdomain) of the solution vector by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

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

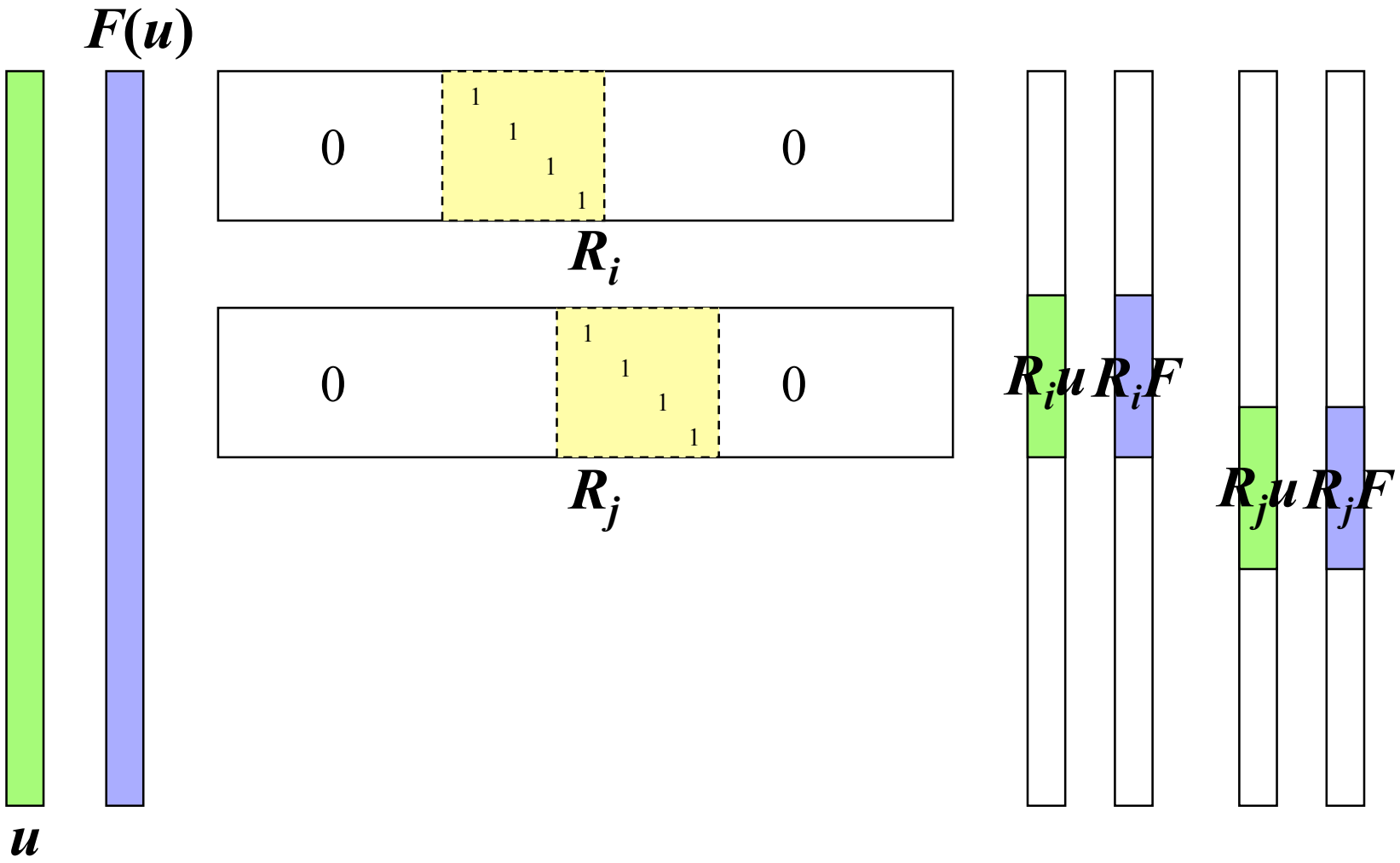
where $\delta_i(u) \in \Re^n$ is nonzero only in the components of the i^{th} partition

- **Then sum the corrections: $\Phi(u) = \sum_i \delta_i(u)$ to get an implicit function of u**

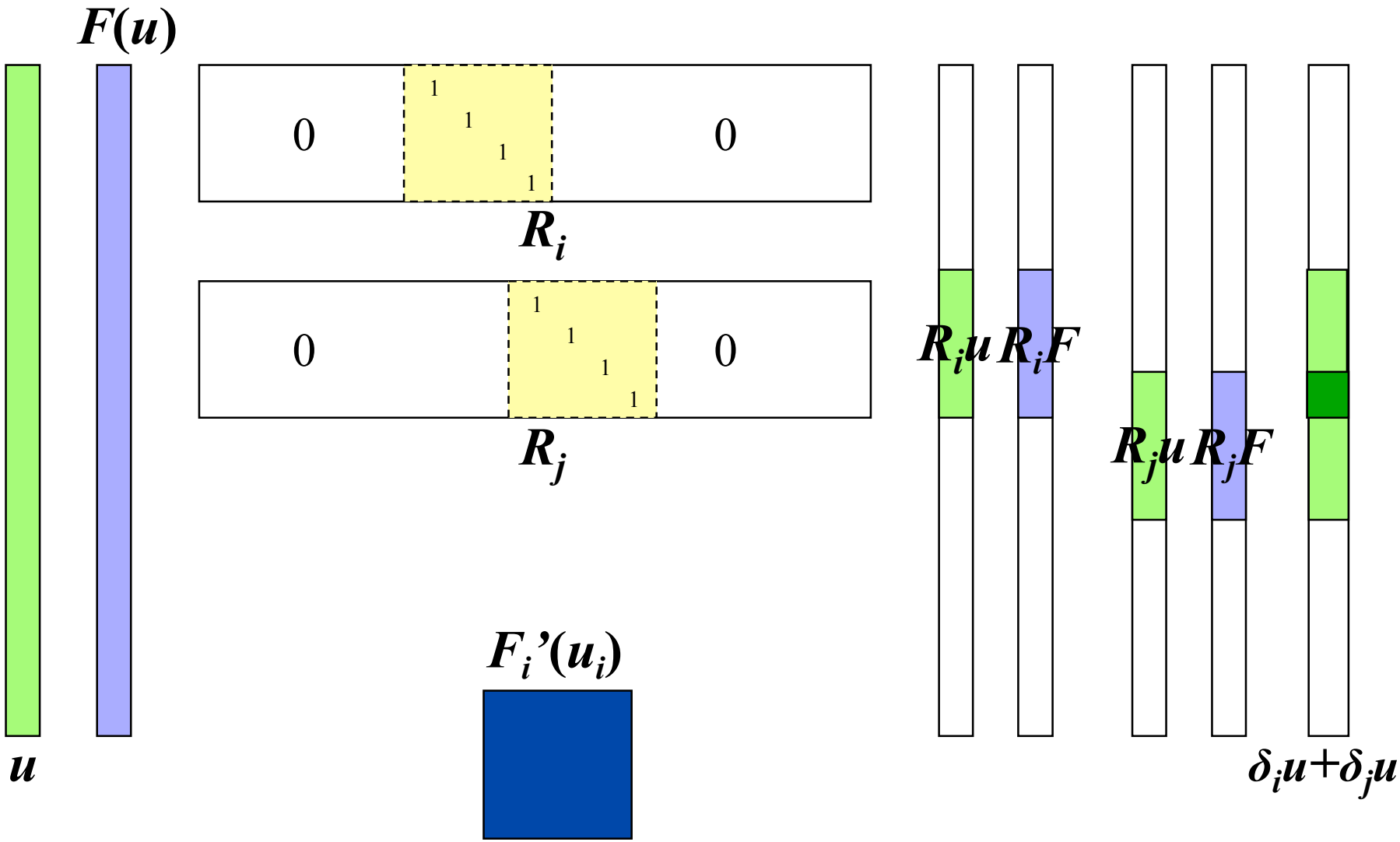
Nonlinear Schwarz – picture



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-K, 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 $F(u)$!
- Suitable for asynchronous parallelism, Newton inside

Some noteworthy algorithmic adaptations to distributed memory architecture

- **Nonlinear Schwarz (Cai & K)**
  - reduce global Krylov-Schwarz synchronizations by applying NKS within well-connected subdomains and performing *few* global outer Newton iterations
- **Restricted Schwarz (Cai & Sarkis)**
  - omit every other local communication (actually leads to *better* convergence, now proved)
- **Extrapolated Schwarz (Garbey & Tromeur-Dervout)**
  - hide interprocessor latency by extrapolating messages received in time integration, with rollback if actual messages have discrepancies in lower Fourier modes (higher mode discrepancies decay anyway)

State of the art in domain decomposition

- **Domain decomposition is the dominant paradigm in contemporary terascale PDE simulation and will be at peta-, exa-, *etc.*, scales**
- **Several freely available software toolkits exist, and successfully scale to thousands of tightly coupled processors for problems on quasi-static meshes**
- **Concerted efforts underway (e.g., 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**