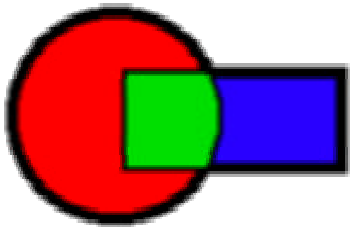
Postscript:
"The Unreasonable Effectiveness of Domain Decomposition"



(with apologies to Eugene P. Wigner)

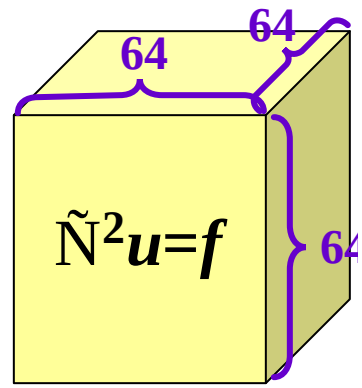
Why optimal algorithms?

- The more powerful the computer, the *greater* the premium on optimality
- Example:
  - Suppose *Alg1* solves a problem in time CN^2, where N is the input size
  - Suppose *Alg2* solves the same problem in time CN
  - Suppose that the machine on which *Alg1* and *Alg2* have been parallelized to run has 10,000 processors
- In constant time (compared to serial), *Alg1* can run a problem 100X larger, whereas *Alg2* can run a problem fully 10,000X larger
- Or, filling up the machine, *Alg1* takes 100X longer

The power of optimal algorithms

- **Advances in algorithmic efficiency rival advances in hardware architecture**
- **Consider Poisson's equation on a cube of size $N=n^3$**

Year	*Method*	*Reference*	*Storage*	*Flops*
1947	GE (banded)	Von Neumann & Goldstine	n^5	n^7
1950	Optimal SOR	Young	n^3	$n^4 \log n$
1971	CG	Reid	n^3	$n^{3.5} \log n$
1984	Full MG	Brandt	n^3	n^3

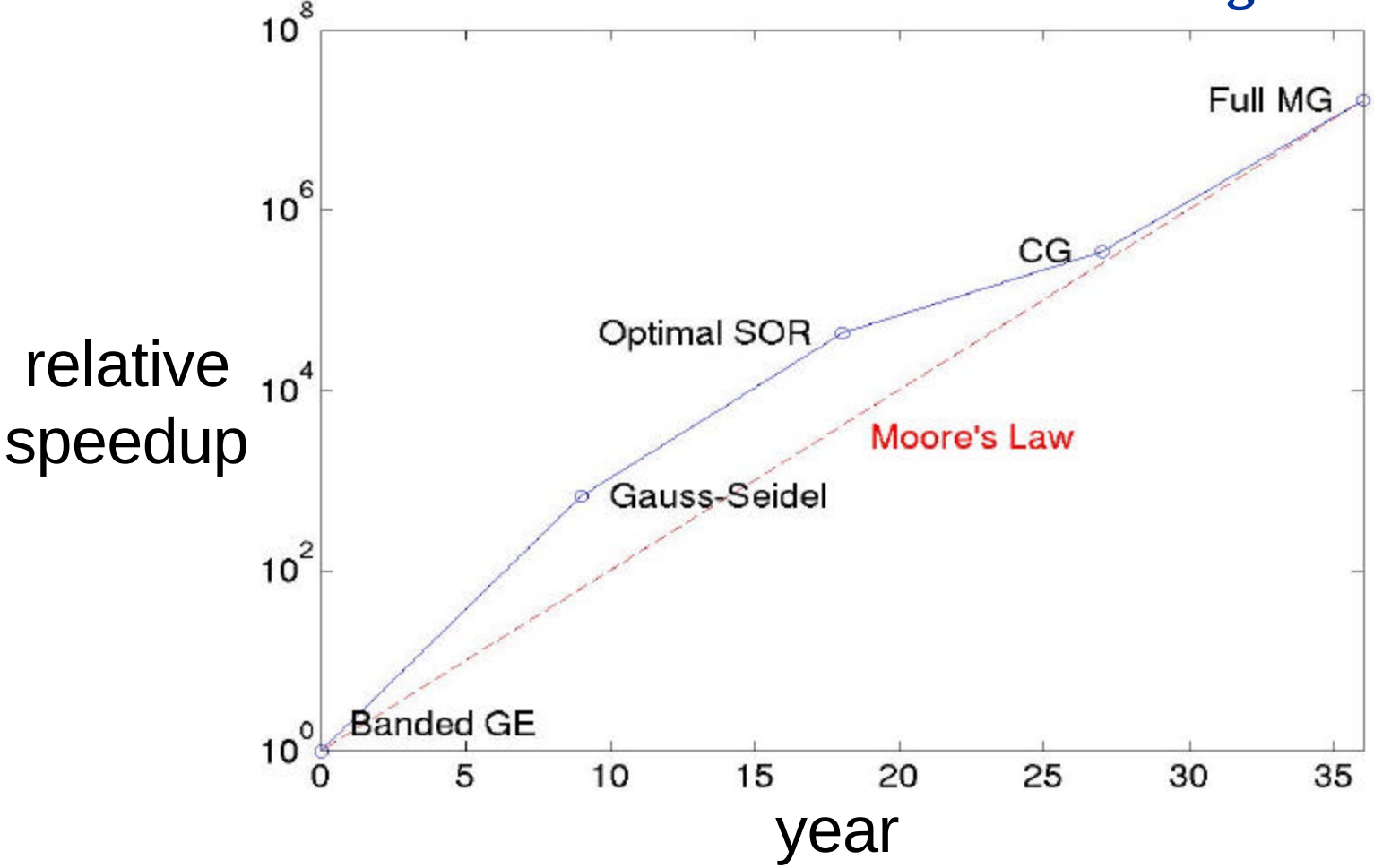


- **If n=64, this implies an overall reduction in flops of ~16 million ***

*On a 16 Mflop/s machine, six-months is reduced to 1 s

Algorithms and Moore's Law

- **This advance took place over a span of about 36 years, or 24 doubling times for Moore's Law**
- **2^{24}»16 million Þ the same as the factor from algorithms alone!**



Agenda for future DD research

- High concurrency (100,000 processors)
- Asynchrony
- Fault tolerance
- Automated tuning of algorithm (to application and to architecture)
- Integration of “forward” simulation with studies of sensitivity, stability, and optimization

High Concurrency

Today

- 10,000 processors in a single room with tightly coupled network
- DD computations scale well, when provided with
  - network rich enough for parallel near neighbor communication
  - fast global reductions (complexity sublinear in processor count)

Future

- 100,000 processors, in a room or as part of a grid
- Most phases of DD computations scale well
  - favorable surface-to-volume comm-to-comp ratio
- However, latencies will nix frequent exact reductions
- Paradigm: *extrapolate* data in retarded messages; *correct (if necessary)* when message arrives, such as in *C(p,q,j)* schemes by Garbey and Tromeur-Dervout

Asynchrony

Today

- *A priori* partitionings for quasi-static meshes provide load-balanced computational tasks between frequent synchronization points
- Good load balance is critical to parallel scalability on 1,000 processors and more

Future

- Adaptivity requirements and far-flung, nondedicated networks will lead to idleness and imbalance at synchronization points
- Need algorithms with looser outer loops than global Newton-Krylov
- Can we design algorithms that are robust with respect to incomplete convergence of inner tasks, like inexact Newton?
- Paradigm: *nonlinear Schwarz* with regional (not global) nonlinear solvers where most execution time is spent

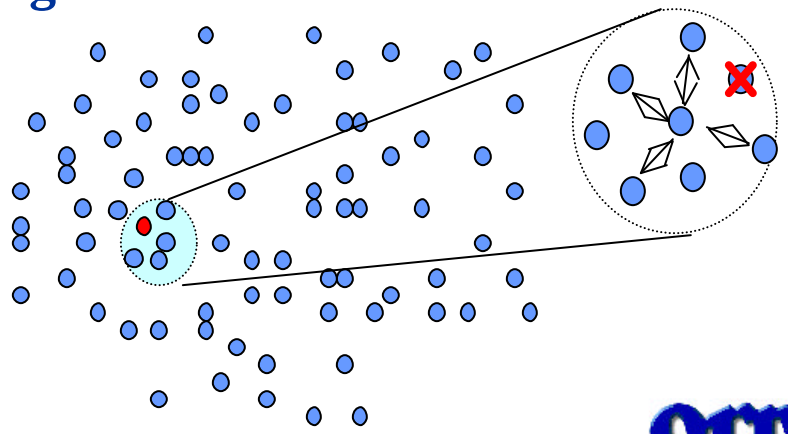
Fault Tolerance

Today

- Fault tolerance is not a driver in most scientific application code projects
- FT handled as follows:
  - Detection of wrong
    - System – in hardware
    - Framework – by runtime env
    - Library – in math or comm lib
  - Notification of application
    - Interrupt – signal sent to job
    - Error code returned by app process
  - Recovery
    - Restart from checkpoint
    - Migration of task to new hardware
    - Reassignment of work to remaining tasks

Future

- With 100,000 processors or worldwide networks, MTBF will be in minutes
- Checkpoint-restart could take longer than the time to next failure
- Paradigm: *naturally fault tolerant algorithms*, robust with respect to failure, such as a new FD algorithm at ORNL



c/o A. Geist

Automated Tuning

Today

- Knowledgeable user-developers parameterize their solvers with experience and theoretically informed intuition for:
  - problem size/processor ratio
  - outer solver type
  - Krylov solver type
  - DD preconditioner type
  - maximum subspace dimensions
  - overlaps
  - fill levels
  - inner tolerances
  - potentially many others

Future

- Less knowledgeable users required to employ parallel iterative solvers in taxing applications
- Need safe defaults and automated tuning strategies
- Paradigm: *parallel direct search (PDS) derivative-free optimization methods*, or other *machine learning* (ML), using overall parallel computational complexity as objective function and algorithm tuning parameters as design variables, to tune solver in preproduction trial executions

Integrated Software

Today

- Each analysis is a “special effort”
- Optimization, sensitivity analysis (e.g., for uncertainty quantification), and stability analysis to fully exploit and contextualize scientific results are *rare*

Future

- Analysis increasingly an “inner loop” around which more sophisticated science-driven tasks are wrapped
- Need PDE task functionality (e.g., residual evaluation, Jacobian evaluation, Jacobian inverse) exposed to optimization/sensitivity/stability algorithms
- Paradigm: integrated software based on common distributed data structures

Architecturally driven ideas in DD

- **Chaotic Relaxation (1969)**
- **Schwarz Waveform Relaxation (1997)**
- **Restricted Additive Schwarz (1997)**
- ***C(p,q,j)* schemes (2000)**
- **Hybrid MPI/OpenMP-based domain decomposition (2000)**

Chaotic Relaxation

- By Chazan & Miranker (1969)
- Basic idea: assign subsets of interdependent equations to different processors and relax concurrently, importing refreshed data on which a given processor depends “as available”
- Convergence (for certain problem classes) as long as no subset goes infinitely long without being updated
- Weak results from theory, but occasional encouraging numerical experiments, including Giraud (2001), who showed that chaotic relaxation can be marginally faster, both in execution time (from relaxation of synchrony) and in terms of actual floating point work done!

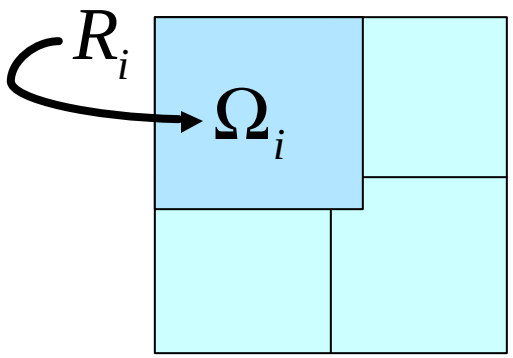
Schwarz Waveform Relaxation

- By Dauod & Gander (1997; see also DD-13 proceedings)
- Rather than exchanging messages at every time step in a space-time cylindrical domain, $(W,(0,T))$, over which a PDE is to be solved, solve in each domain over all time, and exchange interface data over $(0,T)$ at all overlapping Schwarz interfaces less frequently
- Nice convergence theory for parabolic problems using maximum principle
- Interesting for high-latency systems; also for multiphysics systems, since some subdomains can "step over" most restrictive time step arising in other domain
- Disadvantage: memory!

Schwarz Waveform Relaxation

- By Dauod & Gander (1997; see also DD-13 proceedings)
- Rather than exchanging messages at every time step in a space-time cylindrical domain, $(W,(0,T))$, over which a PDE is to be solved, solve in each domain over all time, and exchange interface data over $(0,T)$ at all overlapping Schwarz interfaces less frequently
- Nice convergence theory for parabolic problems using maximum principle
- Interesting for high-latency systems; also for multiphysics systems, since some subdomains can "step over" most restrictive time step arising in other domain
- Disadvantage: memory!

Restrictive Additive Schwarz



- By Cai & Sarkis (1997)
- Consider restriction and extension operators for subdomains, R_i, R_i^T
- Restrict either the restriction or the extension operator to ignore the overlap: $B^{-1} = \sum_i \overline{R}_i^T A_i^{-1} R_i$
- Solve as usual Krylov-Schwarz
- Saves 50% of communication, and actually converges faster in many cases; default in PETSc
- Active area in DD-13:
  - Cai, Dryja & Sarkis' RASHO shows that symmetry can be preserved if one projects to stay in a certain subspace
  - Frommer, Nabben & Szyld give an algebraic theory, including multiplicative RAS

C(p,q,j) schemes

- By Garbey & Tromeur-Dervout (2000)
- To conquer high-latency environments, extrapolate missing boundary data (treating higher and lower Fourier modes differently), and to accommodate low bandwidth environments, reuse extrapolations over several steps
- Employ a posteriori checks against real boundary data when it appears, and adjust as necessary
- Nice results for parabolic problems in the "computational grid" environment

OpenMP/MPI tradeoffs

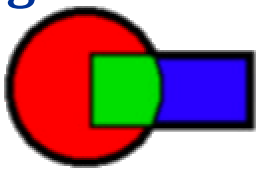
- By I. Charpentier & AHPIK software team (2000); explored for just two procs by Keyes *et al.* (1999)
- For p processors, rather than using p subdomains, use fewer, larger subdomains, and split a subdomain over several processors, using multithreaded subdomain solver, in a hybrid SPMD programming model
- Advantage: fewer subdomains, larger H, gives logarithmic or fractional power improvement in convergence for most DD methods (less information lost on subdomain cuts)

Comments on the DD-15 program

- ***Lots*** **of FETI and FETI-DP (3+ minisymposia and a couple of invited speakers), including FETI for boundary elements (not just finite elements)**
- **Mortar methods, nonconforming elements, discontinuous Galerkin methods**
- **Extensions of Schwarz optimal to "bad parameters" and indefinite problems**
- **Optimal derivations of Schur interface preconditioners**
- **Some CS support for domain decomposition**
- **DD for optimization**
- **Nonlinear Schwarz**
- **Lots of applications**

Conclusions/summary

- **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**
- **Undoubtedly, new theory/algorithms will be part of the solution!**



EOF