Unit 1, Part 1

Introduction to High Performance Computing

A Blue Waters Online Course
Fall 2016

David Keyes, Instructor

Professor of Applied Mathematics and Computational Science
Director, Extreme Computing Research Center

King Abdullah University of Science and Technology

How many computers can dance on the head of a pin?

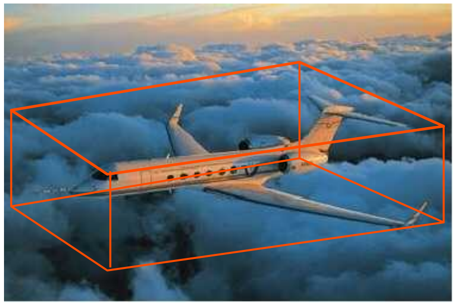
(an introductory lecture on the interaction of application, algorithms, and architecture)

David E. Keyes
King Abdullah University of Science and Technology

with acknowledgments to
William D. Gropp
University of Illinois, Urbana-Champaign

A representative simulation

- Suppose we wish to model the flow about a business jet in *real time*, e.g.,
  - real-time control of flaps, slats, rudder
  - aero-elastic response
  - self-healing adaptations



- Circumscribing box is about 30×20×10 m^3
- Want velocity, density, pressure in every centimeter-sized cell ⇒ 6,000,000,000 points
- Velocity has 3 components, so there are 5 unknowns per point ⇒ 30,000,000,000 or

3×10^{10} unknowns

What do we compute?

- Balance fluxes of mass, momentum, energy
  - conservation of mass (1 equation per cell)
  - Newton's second law (3 equations per cell)
  - first law of thermodynamics (1 equation per cell)
- Take conservation of mass as an example
  - "The time rate of change of mass in the cell balances the flux of mass convected into or out of the cell."
  - As a partial differential equation, we write for mass density ρ and velocity v:

$$\dot{\rho} = -\nabla \bullet (\rho \mathrm{v})$$

Discrete conservation laws

- For a computer, we need to *discretize*
  - convert continuously posed partial differential equation into algebraic form
  - then convert the algebra into an arithmetic algorithm
  - we code the algorithm in floating point computer arithmetic
- Flux balances can be drawn up for mass, momentum, and energy in each cell
- Simplest first attempt at data structure: center the cells on an integer lattice
  - index i runs in the x direction , j in y , and k in z
  - time level l runs over different copies of the spatial lattice
  - store a value in each cell and alternatively update copies

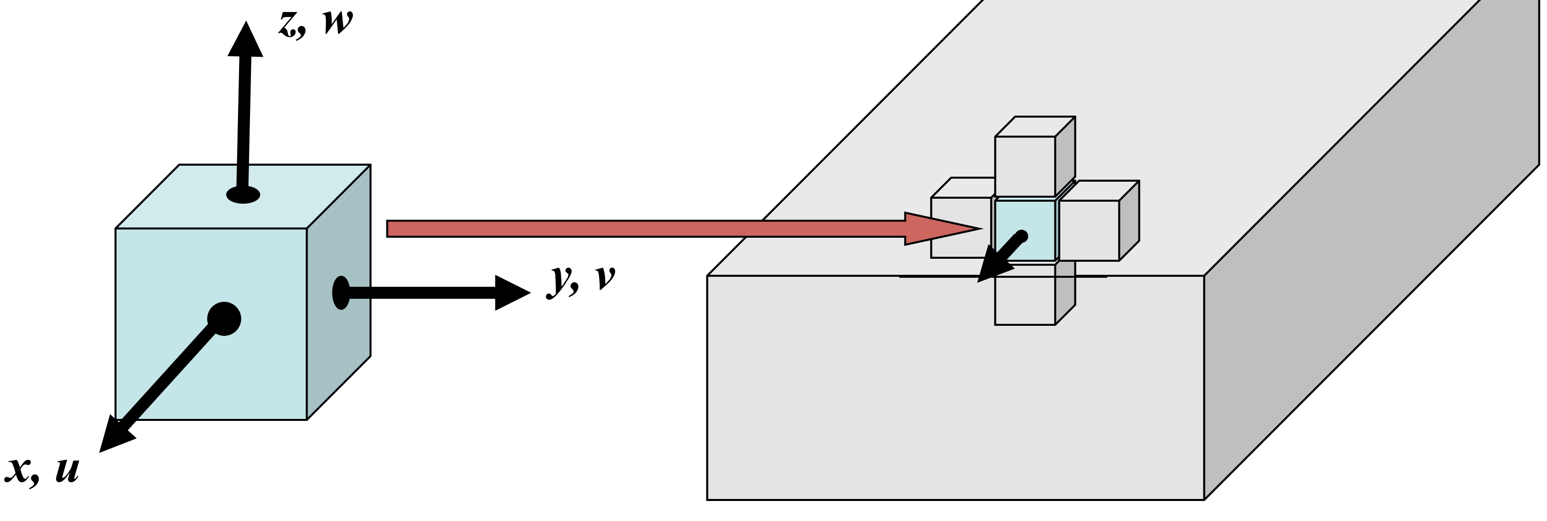
Conservation of mass

- In three dimensions and Cartesian coordinates

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \qquad \mathrm{v} = (u, v, w)$$

- Differential equation expands in 3D to

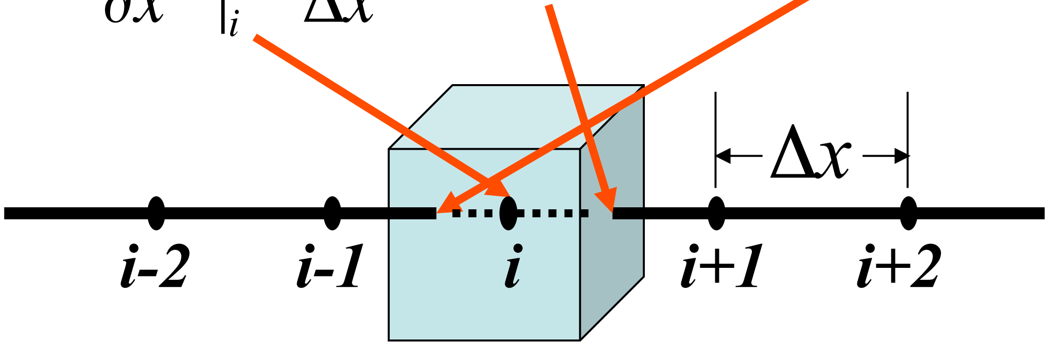
$$\frac{\partial \rho}{\partial t} = -\left[\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z}\right]$$



Discretize the derivatives

- Let $\rho(x_i, y_j, z_k, t_l)$ be approximated as ρ^{l}_{ijk}, with shorthand for lower dimensions
- We can estimate the gradient of the mass flux over the $+x$ face of the cell as follows

$$\left.\frac{\partial(\rho u)}{\partial x}\right|_i \approx \frac{1}{\Delta x}[(\rho u)_{i+1/2} - (\rho u)_{i-1/2}]$$



- Similar expressions may be developed for the y and z derivatives, covering all six faces

Discretization, continued

- Note that each facial flux appears in the balance of the two cells on either side of the face
  - observations like this can allow re-use of intermediates
  - for this introduction, we ignore such details of spatial and temporal locality, which are important in practice

- Estimate the time derivative similarly, using a promotion of the superscript (“forward difference”)

$$\left.\frac{\partial \rho}{\partial t}\right|_{ijk}^{l} \approx \frac{1}{\Delta t}[\rho_{ijk}^{l+1} - \rho_{ijk}^{l}]$$

l is “current”
$l+1$ is “next”

Discretization, continued

- These derivative approximations are *not* the most accurate possible, nor are they universally chosen
  - sometimes, unknowns are staggered at different locations per cell
  - accuracy improves here as first power of Δx and Δt
  - often, faster rates of convergence are sought
  - for a fixed order, accuracy increases as mesh is refined
  - for a fixed mesh, accuracy increases as the order is raised
  - at least second-order is frequently achieved in practice
  - in recent research targeting many-core machines, we have used up to 32nd order (!)

Discrete mass conservation

$$\left\{ \begin{array}{c} rate\ of \\ mass \\ accumulation \end{array} \right\} = \left\{ \begin{array}{c} rate\ of \\ mass \\ in \end{array} \right\} - \left\{ \begin{array}{c} rate\ of \\ mass \\ out \end{array} \right\}$$

$$\frac{\Delta x\,\Delta y\,\Delta z}{\Delta t}\left(\rho\Big|_{ijk}^{l+1} - \rho\Big|_{ijk}^{l}\right) = -\Delta y\,\Delta z\left[\left(\rho u\right)\Big|_{i+1/2\,,jk}^{l} - \left(\rho u\right)\Big|_{i-1/2\,,jk}^{l}\right]$$

$$-\Delta x\,\Delta z\left[\left(\rho v\right)\Big|_{i\,,j+1/2\,,k}^{l} - \left(\rho v\right)\Big|_{i\,,j-1/2\,,k}^{l}\right]$$

$$-\Delta x\,\Delta y\left[\left(\rho w\right)\Big|_{ij\,,k+1/2}^{l} - \left(\rho w\right)\Big|_{ij\,,k-1/2}^{l}\right]$$

Estimating the half-cell quantities

- Relating the staggered quantities on the right-hand side to the original grid functions, we get

$$\left(\rho u\right)\Big|_{i+1/2,jk}^{l} \approx \frac{1}{2}\left[\left(\rho u\right)\Big|_{ijk}^{l} + \left(\rho u\right)\Big|_{i+1,jk}^{l}\right] \approx \frac{1}{2}[\rho_{ijk}^{l} u_{ijk}^{l} + \rho_{i+1,jk}^{l} u_{i+1,jk}^{l}]$$

  and five similar expressions

- This completes the discretization of the law of conservation of mass

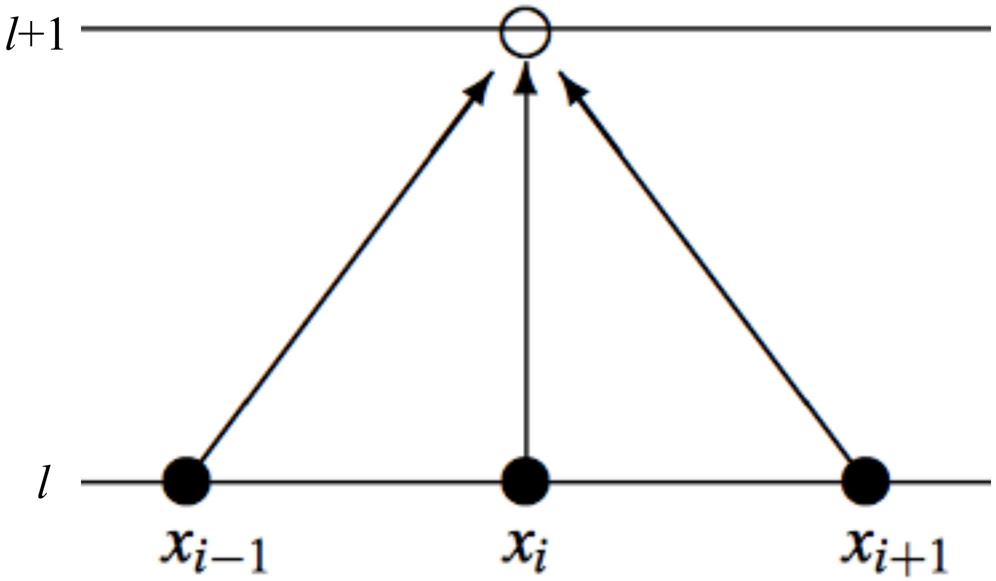
Additional conservation laws

- Similar flux balances can be drawn up for momentum and energy in each cell
- These fluxes are slightly more complex and are modeled in terms of second derivatives (or higher)
- See a *Mathematical Modeling* course in AeroE, MechE, ChemE, ElecE, etc., to understand how the terms to be discretized for computation arise physically
- See a *Numerical Analysis of PDEs* course to see how higher-order discretizations can be developed

Aside

- Often in this course, we need to appeal to results from disciplines that we cannot presume all students will have seen
  - High performance computing has been motivated, historically, by problems in science and engineering and our examples are largely drawn from these
- To succeed in the course, concentrate on what is in common and developed internally to our lecture series
- To succeed in life, aggressively expand your knowledge into adjacent areas

Stencil for mass conservation in one space dimension



$$\begin{aligned}\rho_i^{l+1} &= \rho_i^l + \frac{\Delta t}{2\Delta x}\left[\rho_{i+1}^l u_{i+1}^l - \rho_{i-1}^l u_{i-1}^l\right] \\ &= a_{i-1}\cdot\rho_{i-1}^l + a_i\cdot\rho_i^l + a_{i+1}\cdot\rho_{i+1}^l\end{aligned}$$

How much computation?

- For our 7-point spatial discretization in 3D, and first-order temporal discretization, each equation at each point (a “stencil operation”) at each computational time step requires roughly 8 arithmetic operations
  - more efficient stencil evaluations on emerging manycore architecture hardware, memory-bandwidth limited, is a hot topic
- How many computational time steps?
  - how much real time must be simulated?
  - how big can the time step Δt be?

Computational stability

- It turns out (beyond the scope of this introductory lecture) that the computational simulation will “blow up” if the algorithm tries to outrun “causality” in nature with too large a time step
  - waves propagate with finite speed
  - wave speed defines for each point (x,y,z,t) a spatial “domain of dependence” on data from earlier times
  - if this domain of dependence is not respected, small errors can be amplified (and floating point computations always create small rounding errors)
  - this is known as *numerical instability*

Computational stability, cont.

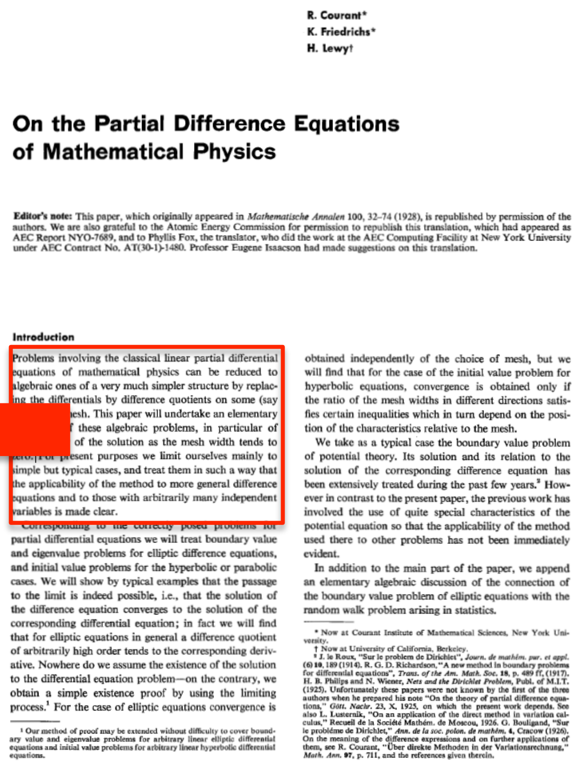
- The computational time step must be small enough that the fastest wave admitted by the governing equations does not cross an entire cell in a single time step
  - the computational domain of dependence must include the entire physical domain of dependence
- For our compressible flow, information propagates at the speed of sound
- For speed c , Courant, Friedrichs, and Levy in 1928 told us:

$$\Delta t < \Delta x / c$$

- This is the “CFL stability criterion” for a wave equation

The original CFL paper (1928)

“Problems involving the classical linear partial differential equations of mathematical physics can be reduced to algebraic ones of a very much simpler structure by replacing the differentials by difference quotients on some (say rectilinear) mesh. This paper will undertake an elementary discussion of these algebraic problems, in particular of the behavior of the solution as the mesh width tends to zero.”



R. Courant*
K. Friedrichs*
H. Lewy†

On the Partial Difference Equations of Mathematical Physics

Editor's note: This paper, which originally appeared in *Mathematische Annalen* 100, 32–74 (1928), is republished by permission of the authors. We are also grateful to the Atomic Energy Commission for permission to republish this translation, which had appeared as AEC Report NYO-7689, and to Phyllis Fox, the translator, who did the work at the AEC Computing Facility at New York University under AEC Contract No. AT(30-1)-1480. Professor Eugene Isaacson had made suggestions on this translation.

Introduction

Problems involving the classical linear partial differential equations of mathematical physics can be reduced to algebraic ones of a very much simpler structure by replacing the differentials by difference quotients on some (say rectilinear) mesh. This paper will undertake an elementary discussion of these algebraic problems, in particular of the behavior of the solution as the mesh width tends to zero. For present purposes we limit ourselves mainly to simple but typical cases, and treat them in such a way that the applicability of the method to more general difference equations and to those with arbitrarily many independent variables is made clear.

Corresponding to the correctly posed problems for partial differential equations we will treat boundary value and eigenvalue problems for elliptic difference equations, and initial value problems for the hyperbolic or parabolic cases. We will show by typical examples that the passage to the limit is indeed possible, i.e., that the solution of the difference equation converges to the solution of the corresponding differential equation; in fact we will find that for elliptic equations in general a difference quotient of arbitrarily high order tends to the corresponding derivative. Nowhere do we assume the existence of the solution to the differential equation problem—on the contrary, we obtain a simple existence proof by using the limiting process.[1] For the case of elliptic equations convergence is obtained independently of the choice of mesh, but we will find that for the case of the initial value problem for hyperbolic equations, convergence is obtained only if the ratio of the mesh widths in different directions satisfies certain inequalities which in turn depend on the position of the characteristics relative to the mesh.

We take as a typical case the boundary value problem of potential theory. Its solution and its relation to the solution of the corresponding difference equation has been extensively treated during the past few years.[2] However in contrast to the present paper, the previous work has involved the use of quite special characteristics of the potential equation so that the applicability of the method used there to other problems has not been immediately evident.

In addition to the main part of the paper, we append an elementary algebraic discussion of the connection of the boundary value problem of elliptic equations with the random walk problem arising in statistics.

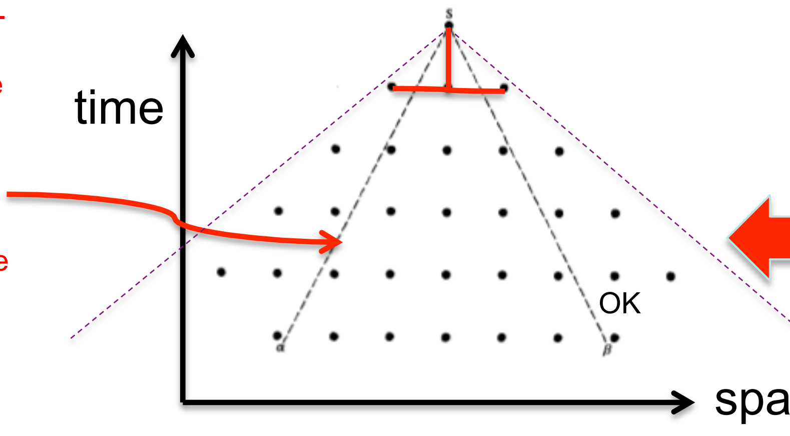
[1] Our method of proof may be extended without difficulty to cover boundary value and eigenvalue problems for arbitrary linear elliptic differential equations and initial value problems for arbitrary linear hyperbolic differential equations.

* Now at Courant Institute of Mathematical Sciences, New York University.

† Now at University of California, Berkeley.

[2] J. le Roux, "Sur le problem de Dirichlet", *Journ. de mathém. pur. et appl.* (6) **10**, 189 (1914). R. G. D. Richardson, "A new method in boundary problems for differential equations", *Trans. of the Am. Math. Soc.* **18**, p. 489 ff, (1917). H. B. Philips and N. Wiener, *Nets and the Dirichlet Problem*, Publ. of M.I.T. (1925). Unfortunately these papers were not known by the first of the three authors when he prepared his note "On the theory of partial difference equations," *Gött. Nachr.* 23, X, 1925, on which the present work depends. See also L. Lusternik, "On an application of the direct method in variation calculus," Recueil de la Société Mathém. de Moscou, 1926. G. Bouligand, "Sur le probléme de Dirichlet," *Ann. de la soc. polon. de mathém.* **4**, Cracow (1926). On the meaning of the difference expressions and on further applications of them, see R. Courant, "Über direkte Methoden in der Variationsrechnung," *Math. Ann.* **97**, p. 711, and the references given therein.

Figure 9



not
OK

A finite sloped line in space-time represents a signal velocity. Points later in time depend upon information from earlier time; spatial spread grows with signal velocity. Numerical domain of dependence must include the physical domain of dependence, by taking timesteps sufficiently small.

Aside

- We will look experimentally at the consequences of violating the Courant-Friedrichs-Levy stability limit later in the course
  - the theory is not difficult, but better developed in a numerical analysis course
- Throughout this course, including three times today, we will direct the curious student to original sources for foundational ideas in high performance computing
  - the 1928 CFL paper laid foundations for today's HPC
  - along with L. F. Richardson's 1922 book, which was written in ignorance of the stability limit (more later)

Let’s plug in and see…

- Sound travels approximately 700 mi/hr in the standard atmosphere, or about 3 × 10^4 cm/s
- Therefore, for a 1 cm distance between cell centers

$$\Delta t \leq \Delta x / c \approx 3 \times 10^{-5}\,\mathrm{sec}$$

How many operations per second?

- Suppose we want to simulate 1 sec of real time
- Total operations required are

$$\frac{1\ \mathrm{sec}}{3\times 10^{-5}\ \mathrm{sec/step}} \times \frac{8\ \mathrm{operations/step}}{\mathrm{unknown}} \times 3\times 10^{10}\ \mathrm{unknowns}$$

or 8×10^{15} operations

- To perform the simulation in real time, we need 8 × 10^{15} operations per second, or 8 Pflop/s, or, equivalently, one operation every 1.25 × 10^{-16} sec

Prefix review

- "flop/s" means "floating point operations per sec"

1,000	Kiloflop/s	Kf
1,000,000	Megaflop/s	Mf
1,000,000,000	Gigaflop/s	Gf
1,000,000,000,000	Teraflop/s	Tf
1,000,000,000,000,000	Petaflop/s	Pf

Your laptop

University lab

Best in class

As of July 2015, approximately 95 computers in the world were known to exceed 1 PF/s on the dense linear algebraic benchmark, ranging up to 93.01 PF/s.

How big can the computer be?

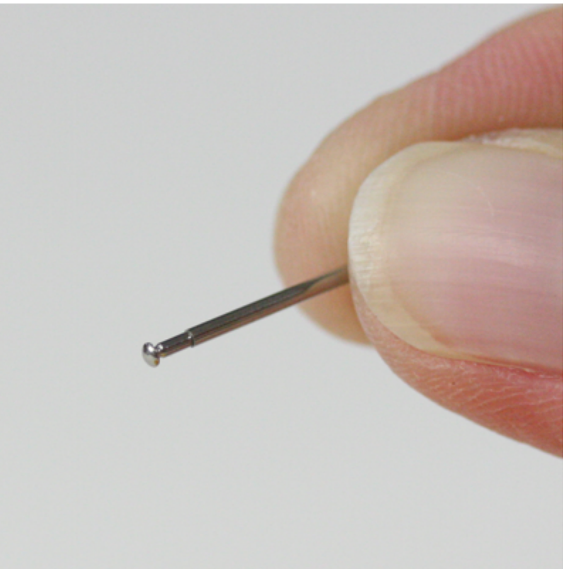
- Assume signal must travel from one end to the other in the time it takes to do one operation, 1.25×10^{-16} sec
- Light travels about a foot in 10^{-9} sec, or 1 cm in 3×10^{-11} sec
- Maximum size for the computer is therefore

$$\frac{1\,\mathrm{cm}}{3\times 10^{-11}\,\mathrm{sec}}\times 1.25\times 10^{-16}\,\mathrm{sec}$$

or about 4×10^{-6} cm

How many fit on the head of a pin?

- Pin head has area of about 10^{-2} cm^2
- For square computers with area (4 x 10^{-6} cm)^2, or 1.6×10^{-11} cm^2, there would be



$$\frac{10^{-2}}{1.6\times10^{-11}} = 6\times10^{8}$$

of our completely causally connected computers on the head of a pin

What is wrong with our assumptions?

- Signal must cross the computer every operation
- Perform one operation at a time
- Use simple monolithic algorithm on uniform grid

How to address these issues

- Signal must cross the computer every operation
  - *Pipelining* allows the computer to be “strung out” within a processor
- Perform one operation at a time
  - *Parallelism* allows many simultaneous operations on different processors
- Use simple monolithic algorithm on a uniform grid
  - *Adaptivity* reduces the number of operations required for a given accuracy

Pipelining

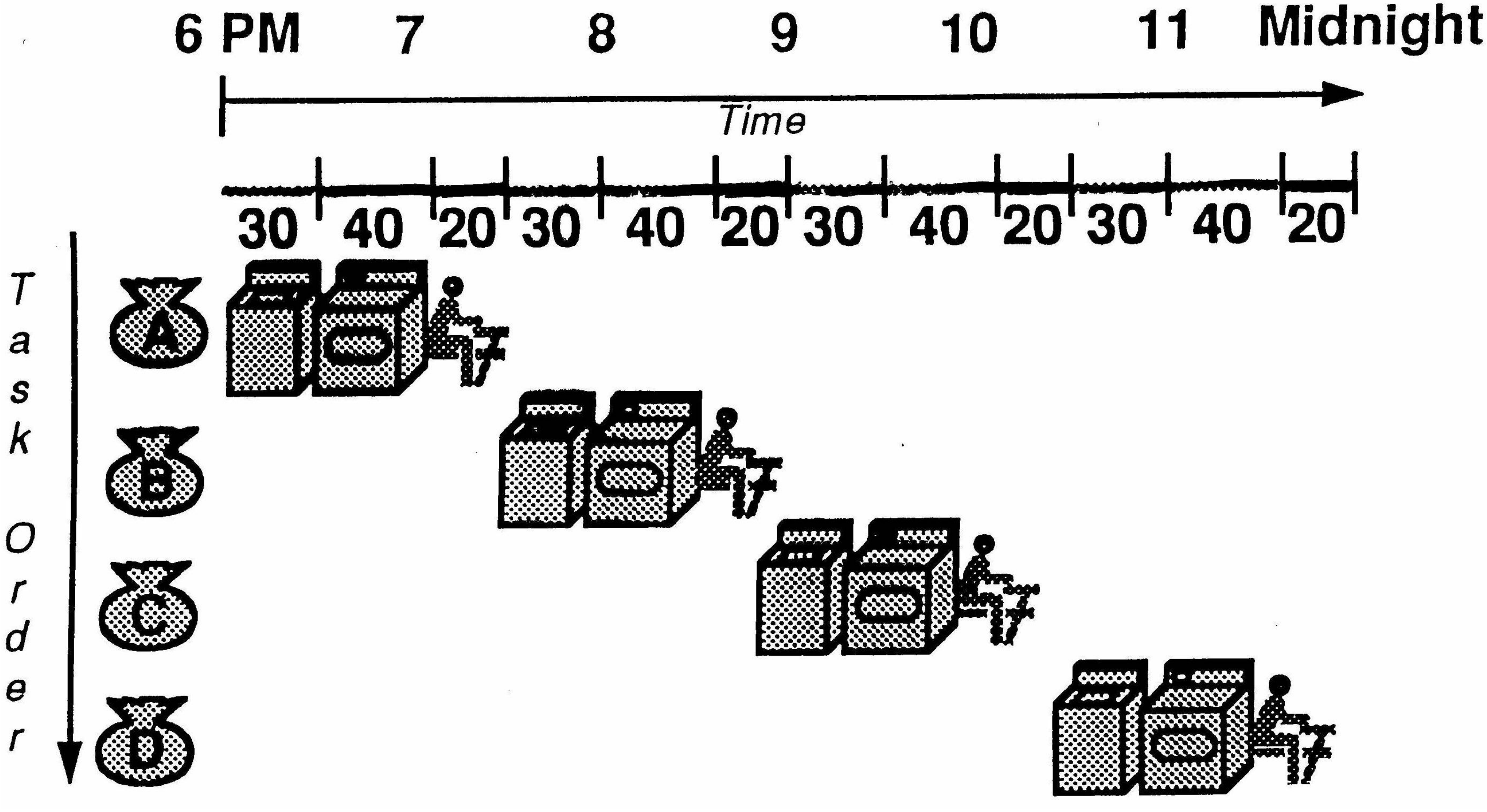
- Often, an operation (e.g., a multiplication of two floating point numbers) is done in several stages

  input→stage1→stage2→output

- Each stage is hosted by a different piece of hardware and can be operating on a different multiplication
- The partially assembled “product” is passed from stage to stage
- Like assembly lines for airplanes, cars, and many other products

Consider laundry pipelining

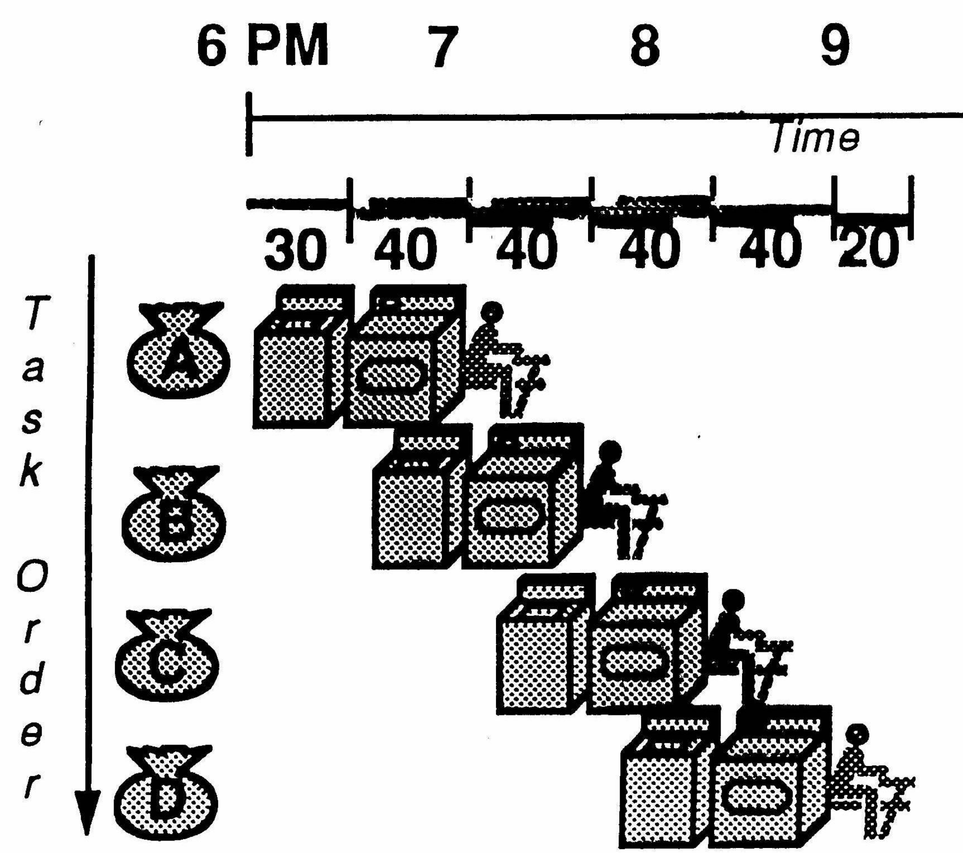
Alex, Briley, Chris, and Drew must each wash (30 min), dry (40 min), and fold (20 min) laundry. If each waits until the previous is finished, the four loads require 6 hours.



c/o D. Patterson

Laundry pipelining, cont.

If Briley starts the second wash as soon as Alex finishes, and then Chris starts the third wash as soon as Briley finishes, etc., the four loads require only 3.5 hours.



Note that in the middle of the task set, all three stations are in use simultaneously.

For long streams, ideal speed-up approaches three – the number of available stations.

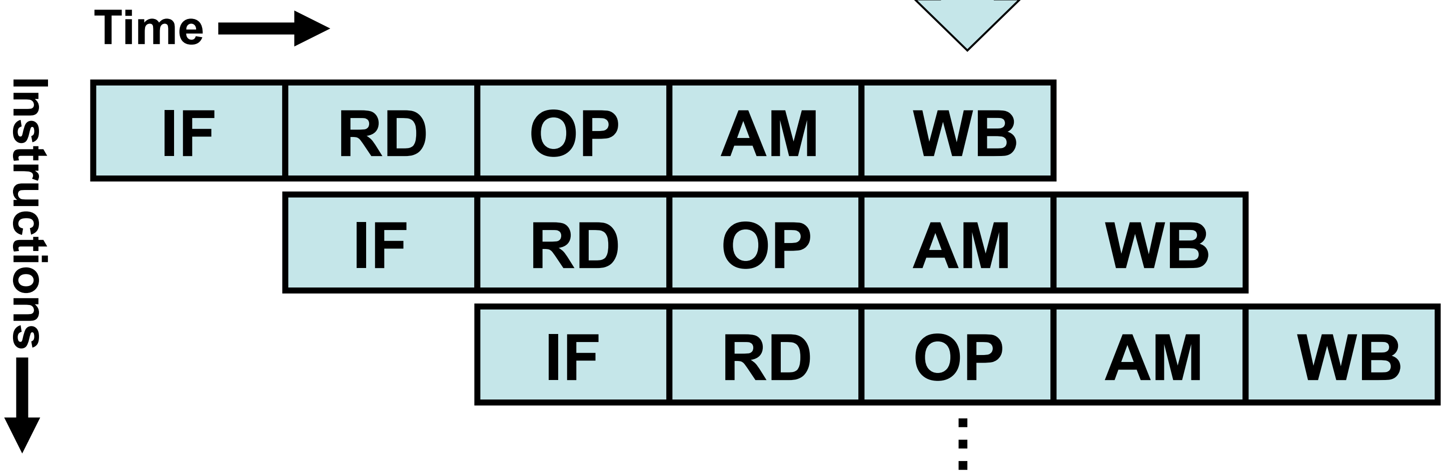
Imbalance between the stages, and pipe filling and draining effects make actual speedup less.

c/o D. Patterson

Arithmetic pipelining

- An arithmetic operation may have 5 stages
  - Instruction fetch (IF)
  - Read operands from registers (RD)
  - Execute operation (OP)
  - Access memory address (AM)
  - Write back to memory (WB)

Actually, each of these stages may be superpipelined further!

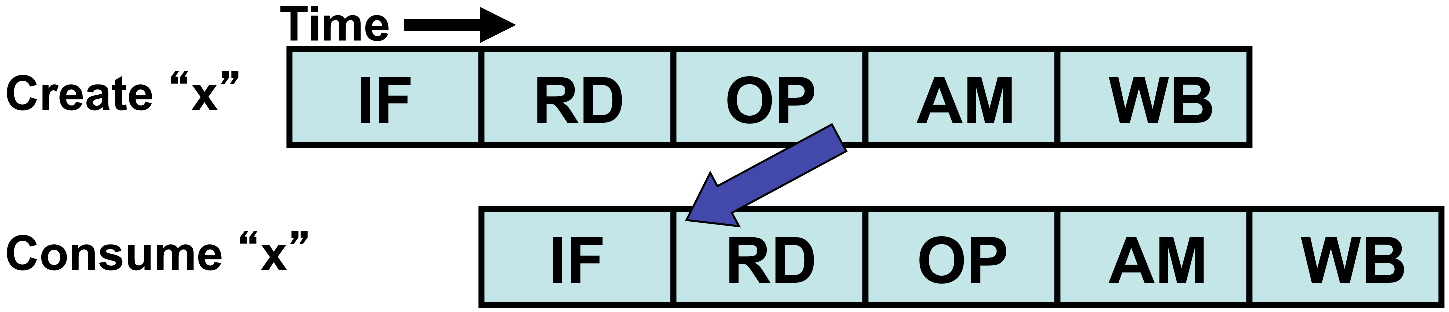


Benefits of pipelining

- Allows the computer to be physically larger
- Signals need travel only from one stage to the next per clock cycle, not over entire computer

Problems with pipelining

- Must find many operations to do independently, since results of earlier scheduled operations are not immediately available for the next; waiting may stall pipe



Time

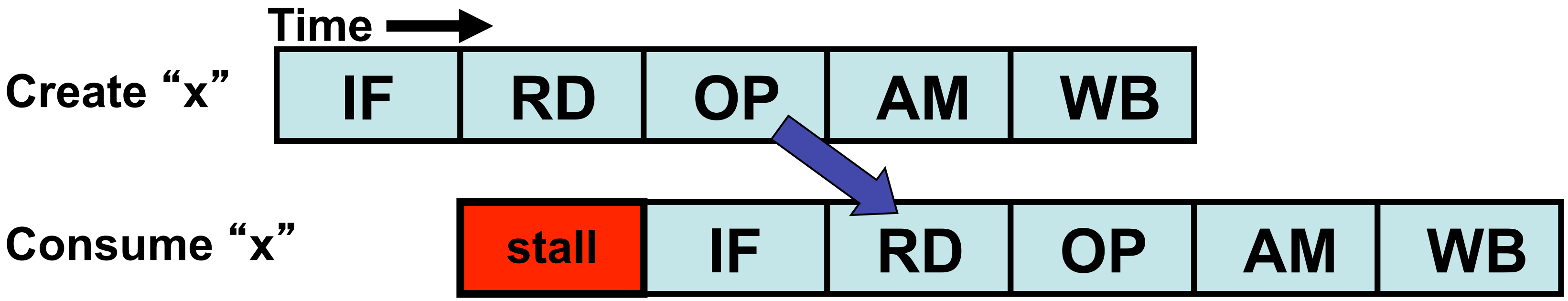
Create “x” IF RD OP AM WB

Consume “x” IF RD OP AM WB

- Conditionals may require partial results to be discarded
- If pipe is not kept full, the extra hardware is wasted, and machine is slow

Problems with pipelining

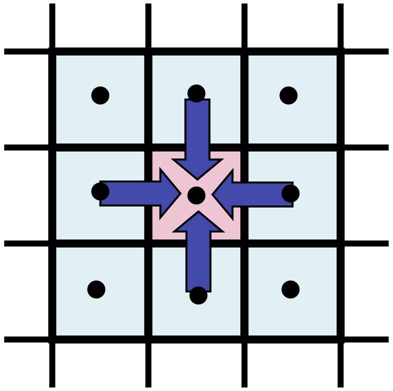
- Must find many operations to do independently, since results of earlier scheduled operations are not immediately available for the next; waiting may stall pipe



- Conditionals may require partial results to be discarded
- If pipe is not kept full, the extra hardware is wasted, and machine is slow

Parallelism

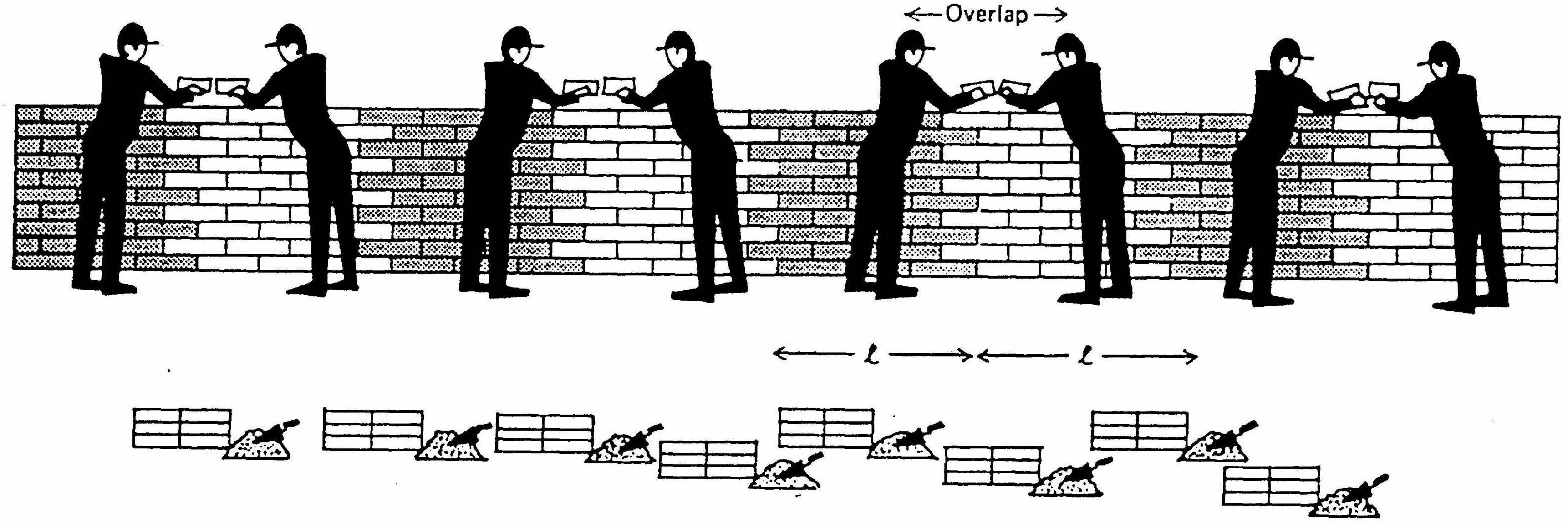
- Often, a large group of operations can be done concurrently, without memory conflicts
- In our simple explicit airplane example, each cell update involves only cells at the previous level
  - All cells at the l+1st level can be updated independently



No purple cell quantities are involved in each other's stencil updates.

Parallelism in building a wall

Concurrent construction of a wall using N = 8 bricklayers.
Decomposition by vertical section.

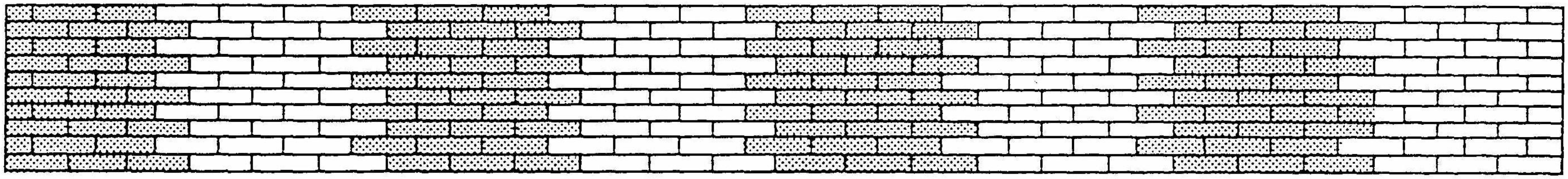


Each worker has an interior "chunk" of independent work, but workers require periodic coordination with their neighbors at their boundaries. One slow worker will eventually stall the rest. Potential speedup is proportional to the number of workers, less coordination overhead.

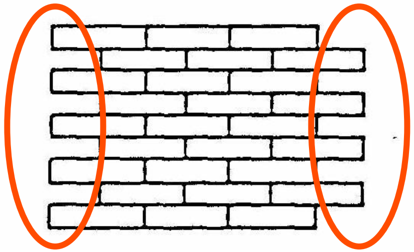
c/o G. Fox

Vertical task decomposition

The complete problem.



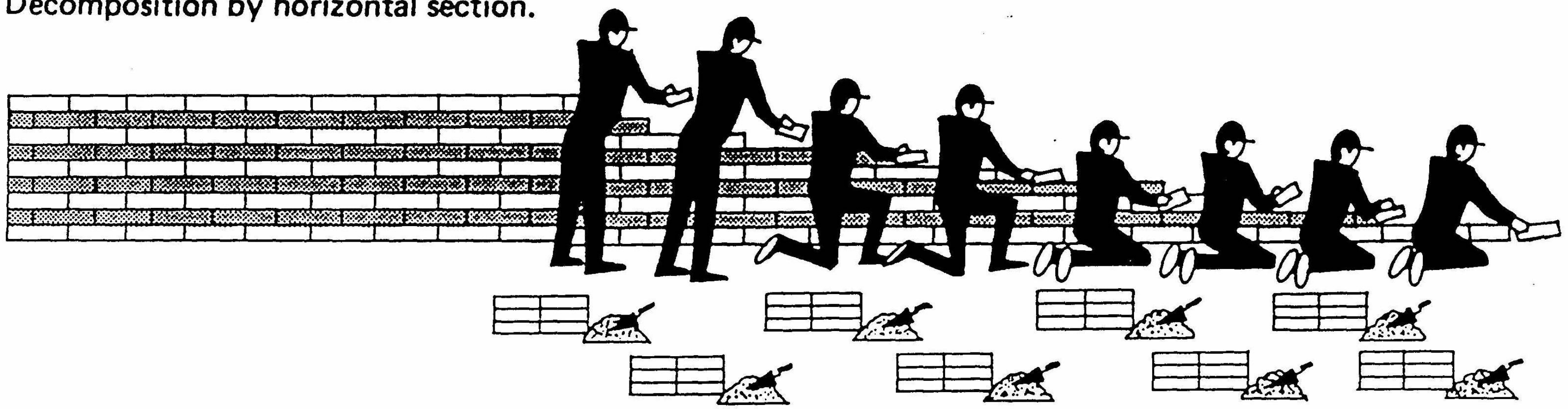
The sub task performed by an individual bricklayer.



overlap zones

c/o G. Fox

Multiple decompositions possible

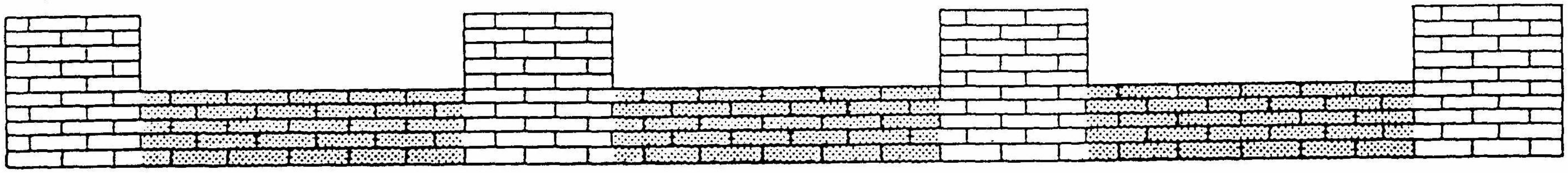


Concurrent Construction of a wall using N = 8 bricklayers.
Decomposition by horizontal section.

A horizontal decomposition, rather than vertical, looks like pipelining. Each worker must wait for the previous to begin; then all are busy until near the end. Potential speedup is proportional to number of workers in the limit of an infinitely long wall.

c/o G. Fox

Nonuniform tasks



Decomposition of wall for an irregular geometry.
Equalize number of bricks per mason, not length of wall per mason.

In the two previous examples, all workers “ran the same program” on “data” in different locations: single-program, multiple-data (SPMD). In the example above, there are two types of programs: one for odd-numbered workers, another for even-numbered.

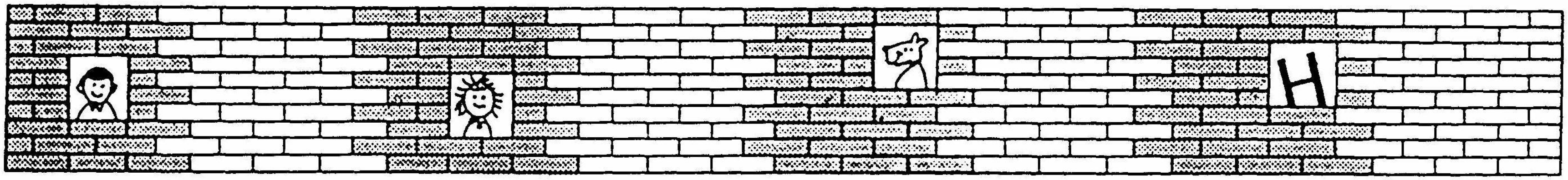
(Actually, these are *two different parameterizations* of basically the same program.)

Observe that the work is load-balanced; each worker has the same number of bricks to lay.

c/o G. Fox

Inhomogeneous tasks

An inhomogenous wall with decoration.
Best decomposition uncertain.



For this highly irregular wall, the differently placed holes may require very different amounts of time to position. It may be *a priori* difficult to estimate a load-balanced decomposition of concurrent work.

Building this wall may require dynamic decomposition to keep each worker busy.

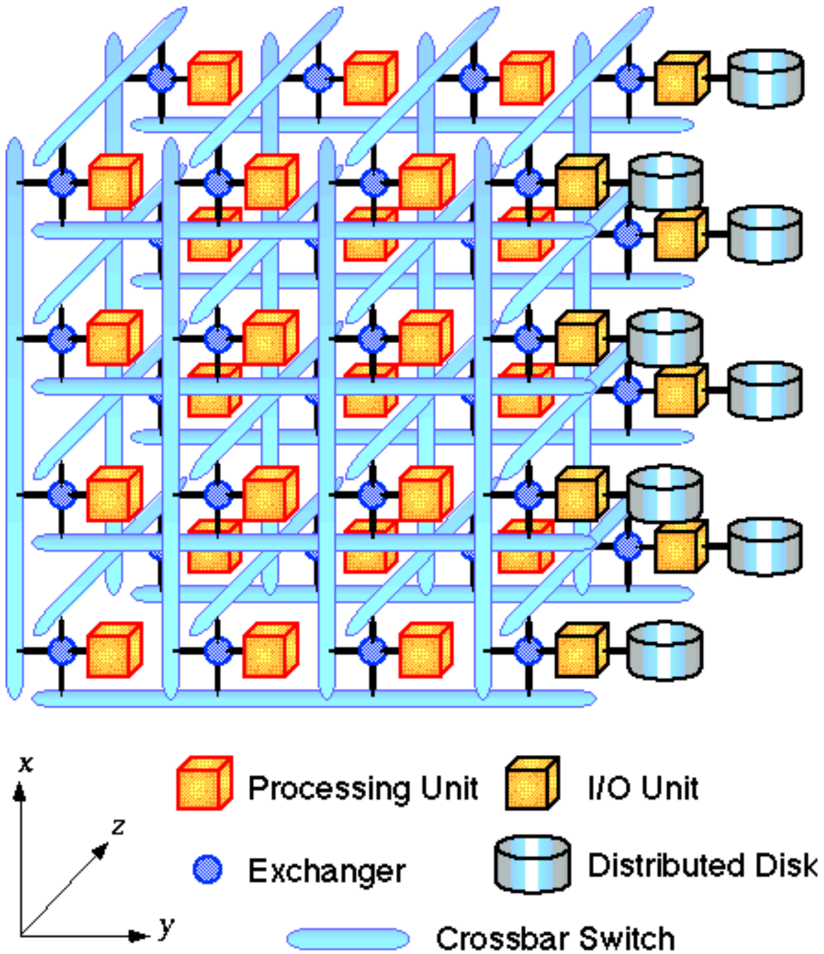
There is a tension between *concurrency* and *irregularity*. Orders are much harder to give for workers on this wall.

c/o G. Fox

Benefits of parallelism

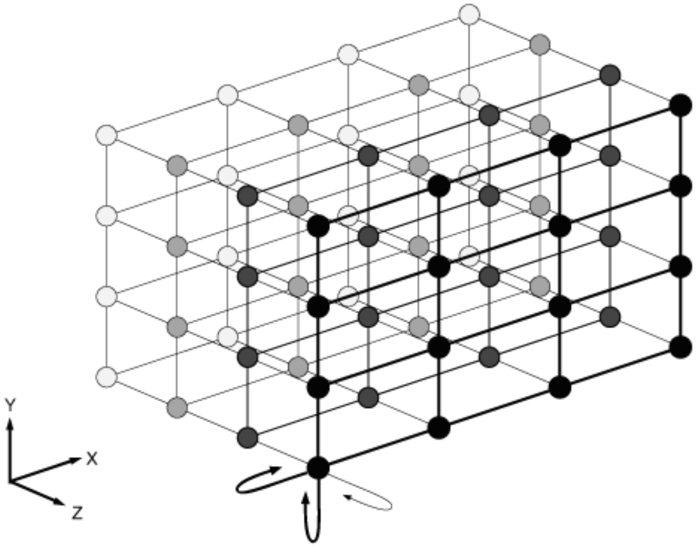
- Allows the computer to be physically larger
- If we had one million computers, then each computer would only have to do 8×10^9 operations per second
  - say 4 operations per clock on a 2 GHz processor
- This would allow the computers to be about 3 cm apart

Parallel processor configurations



In the airplane example, each processor in the 3D array (left) can be made responsible for a 3D chunk of space.

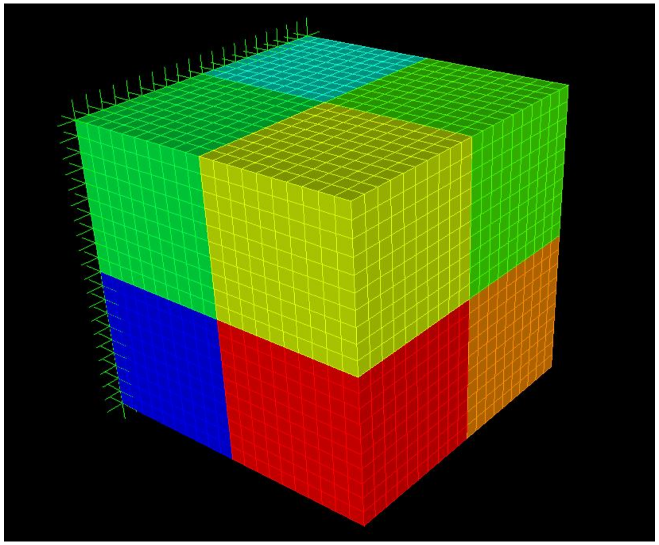
The global cross-bar switch is overkill in this case. A mesh network (below) is sufficient.



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 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
  - simply differentiate time estimate in terms of *(N,P)* with respect to *P*, equate to zero, and solve for *P* as a function 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 $C\ P^{(1/d)}$, where *d* is dimension
  - *C* includes synchronization frequency
- Same 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

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

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

or (with $\theta \equiv \frac{32B^3}{729A^2C}$),

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

- p can grow linearly with n
- In the limit as $B/C \to 0$, $p_{opt} = \left(\frac{A}{C}\right)^{1/3}\cdot n$

3D stencil computation illustration

Rich local network, tree-based global reductions

- Optimal running time:

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

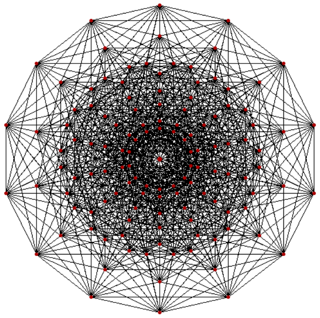
where

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

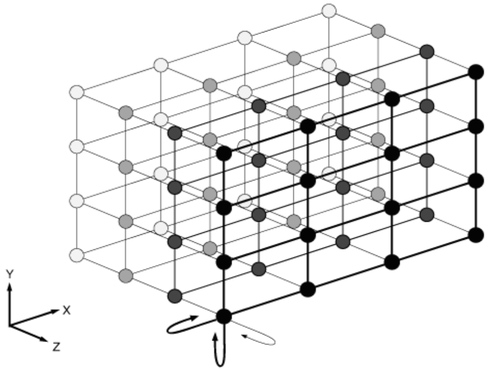
- Limit of infinite neighbor bandwidth, zero neighbor latency (B, θ approach 0):

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

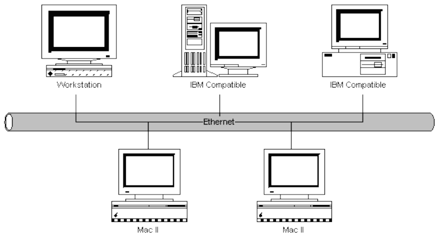
Scalability results for 3D stencil computations



- 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")

Moore’ s Law

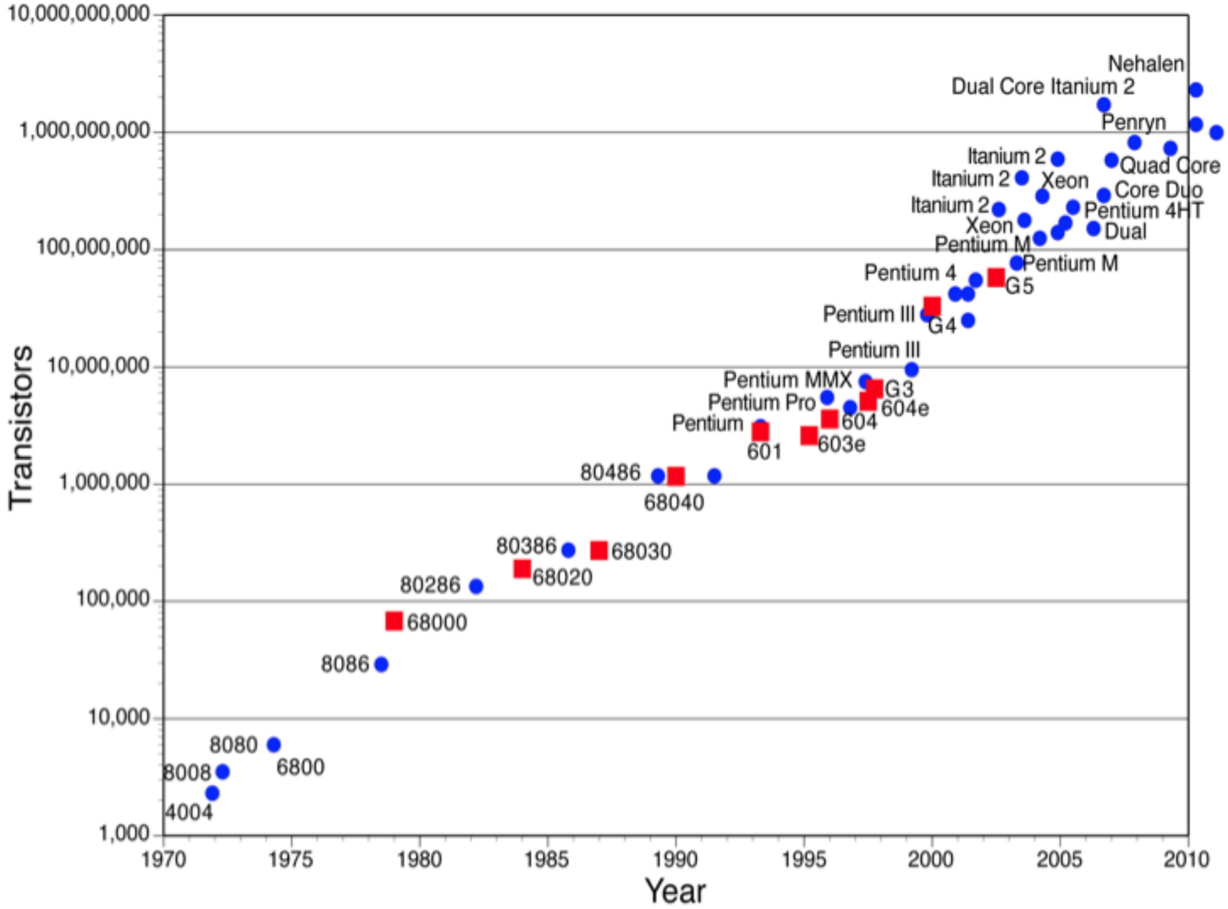
In 1965, Gordon Moore of Intel observed an exponential growth in the number of transistors per integrated circuit and optimistically predicted that this trend would continue.

It did.

For about 40 years…

“Moore’ s Law” informally refers to a doubling of transistors per chip every 18-24 months, which translates into performance, though not quite at the same rate.

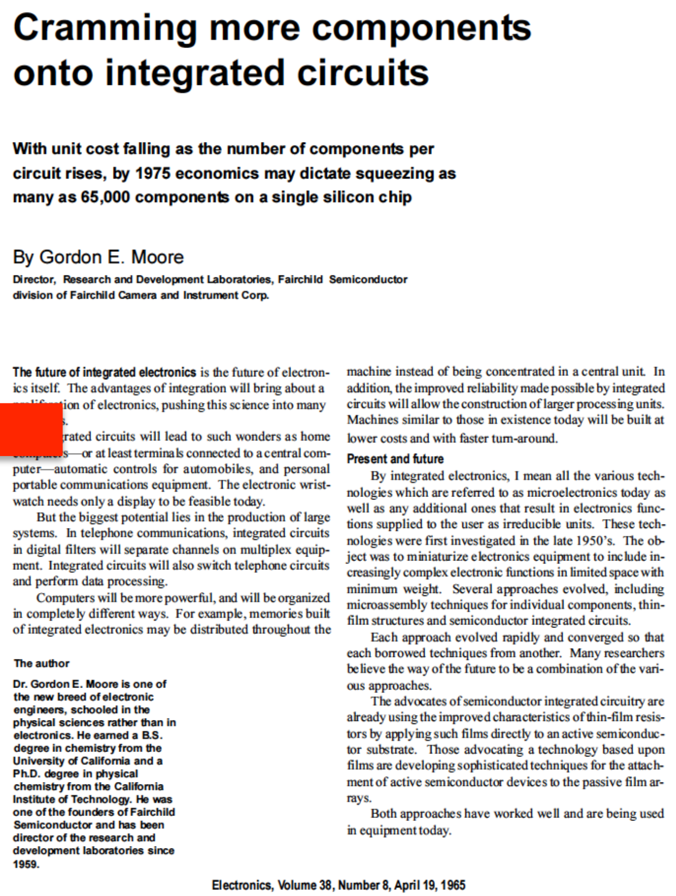
CPU Transistor Counts, 1970-2010 and Moore’s Law



c/o Intel

The original Moore paper (1965)

“Integrated circuits will lead to such wonders as home computers […], automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today... Computers will be more powerful, and will be organized in completely different ways. For example, memories built of integrated electronics may be distributed throughout the machine instead of being concentrated in a central unit.”



Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

The future of integrated electronics is the future of electronics itself. The advantages of integration will bring about a proliferation of electronics, pushing this science into many areas.

Integrated circuits will lead to such wonders as home computers—or at least terminals connected to a central computer—automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today.

But the biggest potential lies in the production of large systems. In telephone communications, integrated circuits in digital filters will separate channels on multiplex equipment. Integrated circuits will also switch telephone circuits and perform data processing.

Computers will be more powerful, and will be organized in completely different ways. For example, memories built of integrated electronics may be distributed throughout the machine instead of being concentrated in a central unit. In addition, the improved reliability made possible by integrated circuits will allow the construction of larger processing units. Machines similar to those in existence today will be built at lower costs and with faster turn-around.

Present and future

By integrated electronics, I mean all the various technologies which are referred to as microelectronics today as well as any additional ones that result in electronics functions supplied to the user as irreducible units. These technologies were first investigated in the late 1950's. The object was to miniaturize electronics equipment to include increasingly complex electronic functions in limited space with minimum weight. Several approaches evolved, including microassembly techniques for individual components, thin-film structures and semiconductor integrated circuits.

Each approach evolved rapidly and converged so that each borrowed techniques from another. Many researchers believe the way of the future to be a combination of the various approaches.

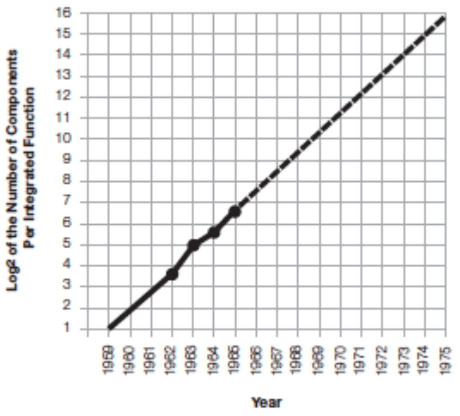
The advocates of semiconductor integrated circuitry are already using the improved characteristics of thin-film resistors by applying such films directly to an active semiconductor substrate. Those advocating a technology based upon films are developing sophisticated techniques for the attachment of active semiconductor devices to the passive film arrays.

Both approaches have worked well and are being used in equipment today.

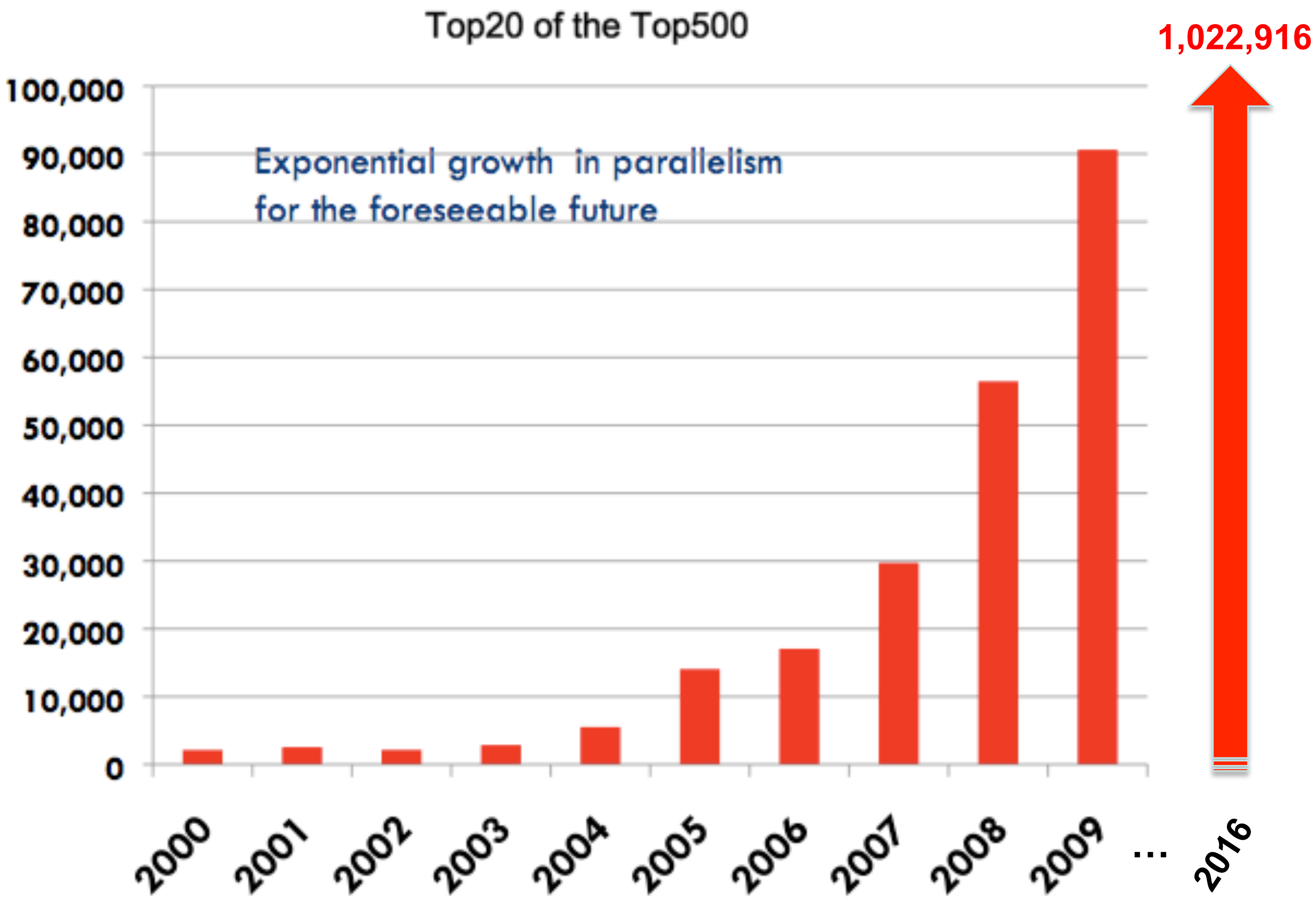
The author

Dr. Gordon E. Moore is one of the new breed of electronic engineers, schooled in the physical sciences rather than in electronics. He earned a B.S. degree in chemistry from the University of California and a Ph.D. degree in physical chemistry from the California Institute of Technology. He was one of the founders of Fairchild Semiconductor and has been director of the research and development laboratories since 1959.

Electronics, Volume 38, Number 8, April 19, 1965



Average number of processor cores in a supercomputer



Aside

- Most of the world's supercomputers are ranked against each other about every six months, at ISC'xy (June) and SC'xy (November), respectively
- 500 top computers are ranked by the High Performance Linpack (HPL) benchmark
- HPL measures the rate at which a computer can perform dense Gaussian elimination with partial pivoting
- We take a brief look at the Top 10 computers ranked by the High Performance Linpack (HPL) benchmark

#1-ranked TaihuLight by Sunway / NRCPC

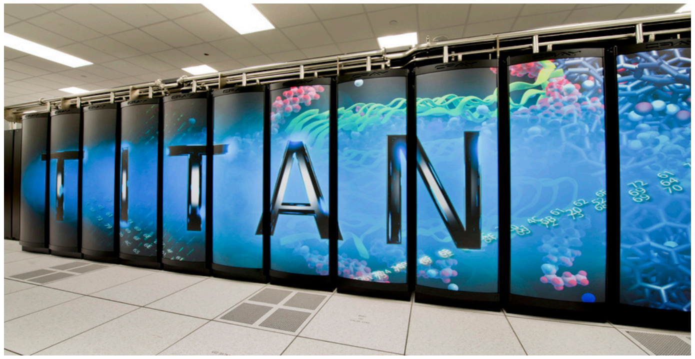
10,649,600 processor cores on
40,960 SW26010 chips (125.4 PetaFlop/s)
Global #1 ranked system (China first entered Top 50 in 2008)

#2-ranked Tianhe-2 by Inspur / NUDT / Intel

3,120,000 processor cores on
32K Ivy Bridge + 48K Xeon Phi chips (54.9 PetaFlop/s)

US DOE's Top 10 petascale systems

#3 Cray XK7
"Titan" (ORNL)



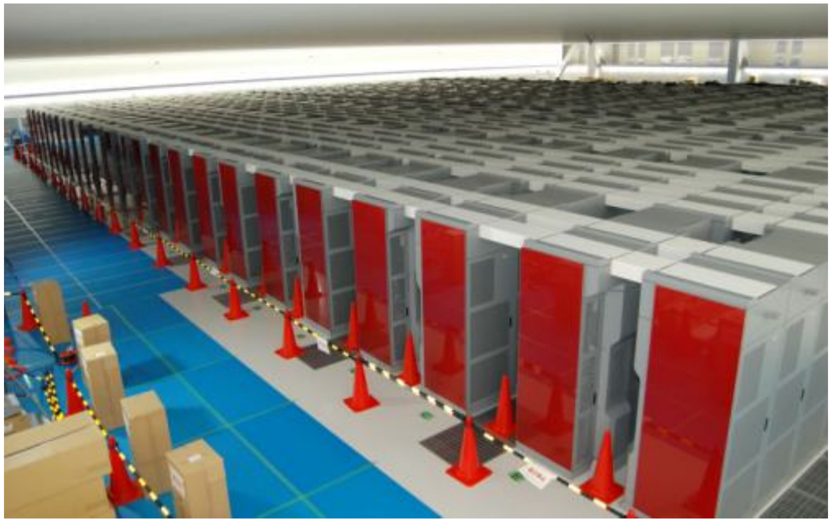
#4 IBM BlueGene/Q
"Sequoia" (LLNL)

#6 IBM BlueGene/Q
"Mira" (ANL)

#7 Cray XC40
"Trinity" (LANL)

More Top 10 petascale systems

#5 Fujitsu
“Kei” (Riken)



#8 Cray XC30
“Piz Daint” (CSCS)

#9 Cray XC40
“Hazel Hen” (HLRS)

KAUST's Top 10 system

#10 Cray XC40 "Shaheen"

#10 on HPL: High Performance Linpack (76% of theoretical peak)
#12 on HPCG: High Performance Conjugate Gradient (1.6% of theoretical peak)
#4 on HPGMG: High Performance Geometric Multigrid (~0.65% of theoretical peak)

We shall study the basis for all three benchmarks: dense direct solvers, sparse Krylov solvers, and multilevel solvers

Aside

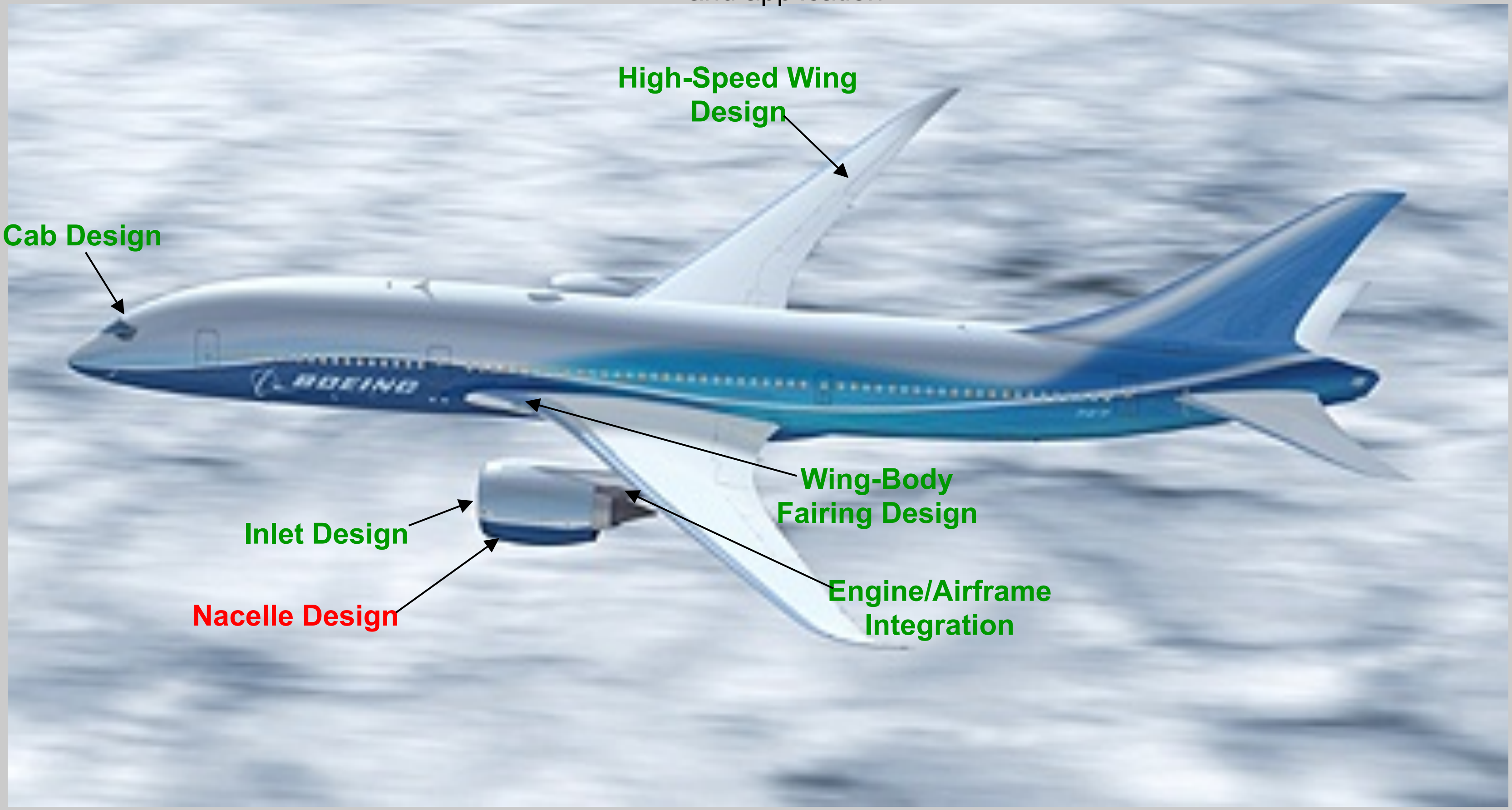
- The primary supercomputer used for exercises in this course, Blue Waters, hosted by UIUC for the National Center for Supercomputer Applications (NCSA) would be approximately #5 on the TOP500 list, if it participated
- Since HPL is not representative of the majority of large scale computational science and engineering applications, and since it requires significant resources to run, the NCSA has declined to submit

1979: Computational Fluid Dynamics for B767

🟩 **Much CFD penetration.** Opportunities exist for higher accuracy and expanded complexity

🟦 **Some CFD penetration.** Opportunities exist for large increases in design process speed and application

🟥 **CFD penetration opportunity**



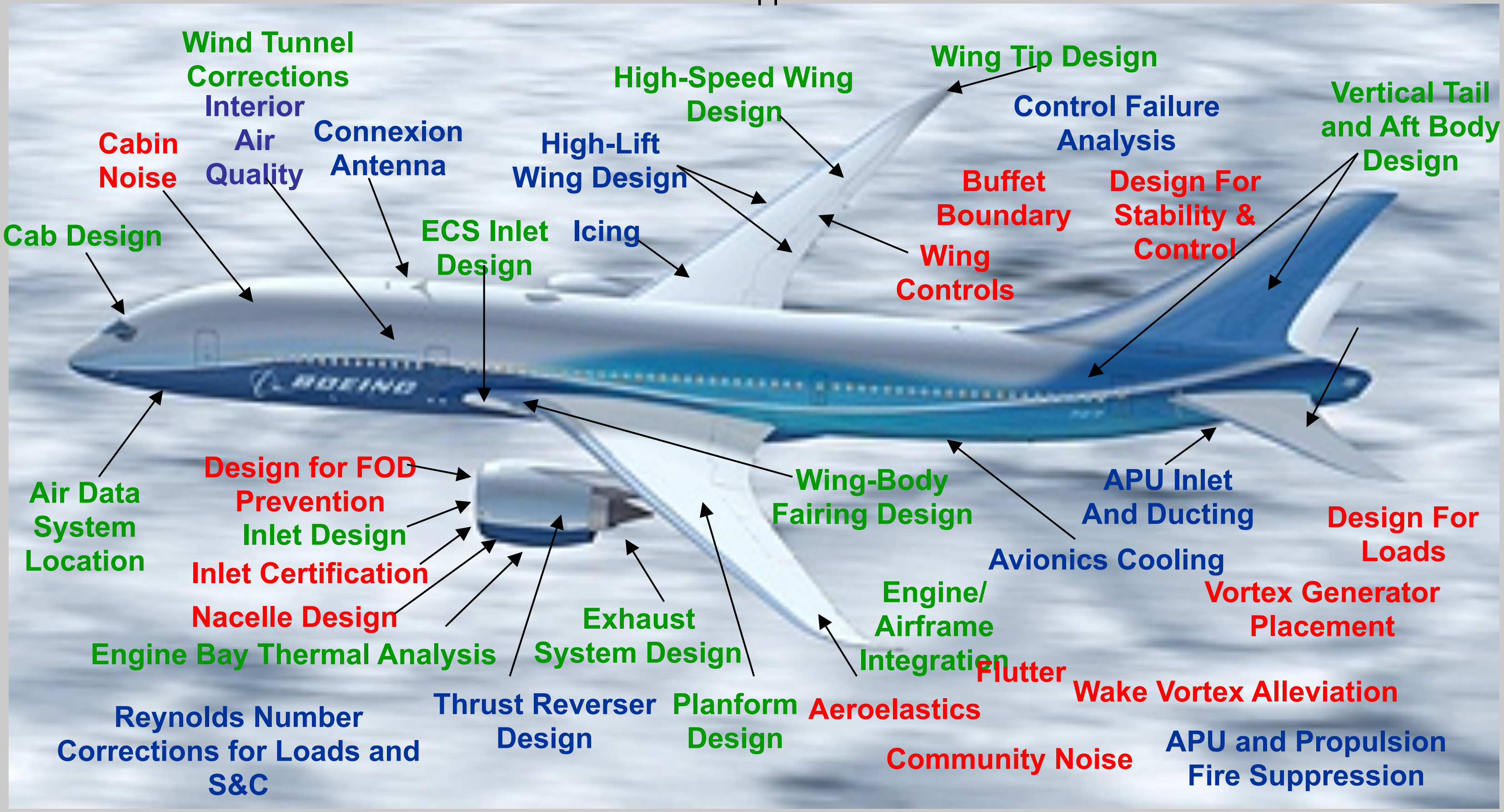
c/o Douglas Ball, Boeing

2005: Computational Fluid Dynamics for B787

🟩 **Much CFD penetration.**
Opportunities exist for higher accuracy and expanded complexity

🟦 **Some CFD penetration.**
Opportunities exist for large increases in design process speed and application

🟥 **CFD penetration opportunity**



Wind Tunnel Corrections

Interior Air Quality

Cabin Noise

Connexion Antenna

Wing Tip Design

High-Speed Wing Design

Control Failure Analysis

Vertical Tail and Aft Body Design

High-Lift Wing Design

Buffet Boundary

Design For Stability & Control

Cab Design

ECS Inlet Design

Icing

Wing Controls

Air Data System Location

Design for FOD Prevention

Inlet Design

Wing-Body Fairing Design

APU Inlet And Ducting

Design For Loads

Inlet Certification

Avionics Cooling

Engine/ Airframe Integration

Vortex Generator Placement

Nacelle Design

Exhaust System Design

Engine Bay Thermal Analysis

Flutter

Wake Vortex Alleviation

Thrust Reverser Design

Planform Design

Aeroelastics

Reynolds Number Corrections for Loads and S&C

Community Noise

APU and Propulsion Fire Suppression

c/o Douglas Ball, Boeing

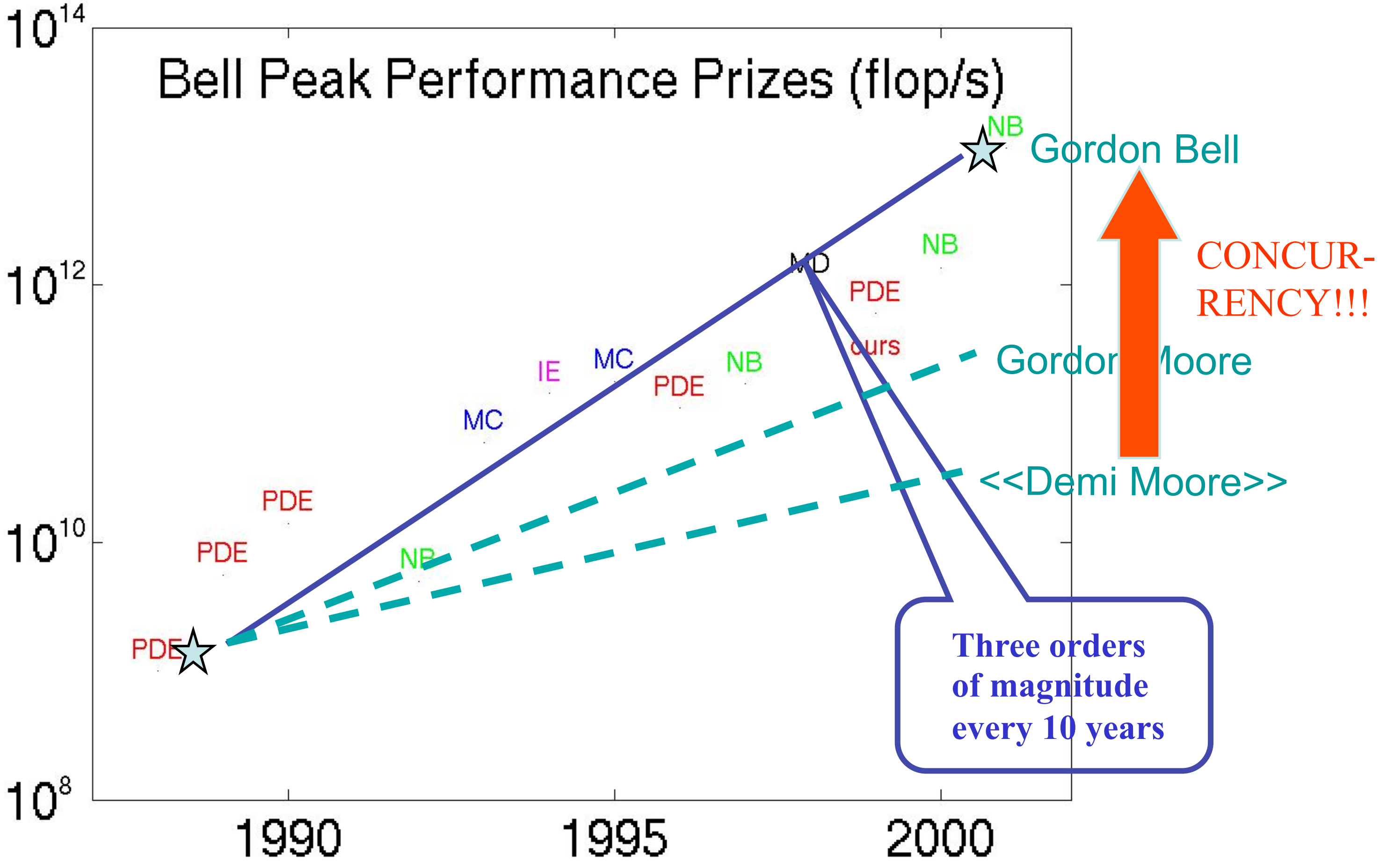
Simulation driven by price and capability

By the Gordon Bell Prize, simulation *cost per performance* has improved by nearly a million times in two decades. Performance on *real applications* (e.g., mechanics, materials, petroleum reservoirs, etc.) has improved *more* than a million times.

Gordon Bell Prize: Price Performance Year	Cost per delivered Gigaflop/s
1989	$2,500,000
1999	$6,900
2009	$8

Gordon Bell Prize: Peak Performance Year	Gigaflop/s delivered to applications
1988	1
1998	1,020
2008	1,350,000

Gordon Bell Prize outpaces Moore’s Law



Gordon Bell

CONCUR-
RENCY!!!

Gordon Moore

<<Demi Moore>>

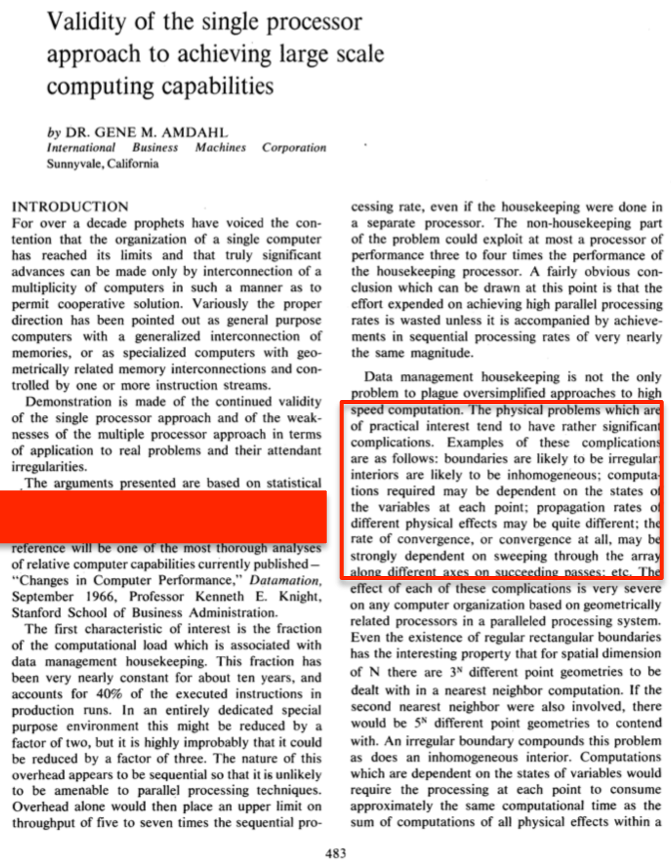
Three orders
of magnitude
every 10 years

Problems with parallelism

- Must find massive concurrency in the task
- Still need many computers, each of which must be fast
- Communication between computers becomes a dominant factor
- Amdahl's Law limits speedup available based on remaining non-concurrent work

The original Amdahl paper (1967)

"The physical problems which are of practical interest tend to have rather significant complications. Examples are as follows: boundaries are likely to be irregular; interiors are likely to be inhomogeneous; computations required may be dependent upon the states of the variables at each point; propagation rates of different physical effects may be quite different; the rate of convergence, or convergence at all, may be strongly dependent upon sweeping through the array along different axes on succeeding passes; etc."



Validity of the single processor approach to achieving large scale computing capabilities

by DR. GENE M. AMDAHL
International Business Machines Corporation
Sunnyvale, California

INTRODUCTION

For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution. Variously the proper direction has been pointed out as general purpose computers with a generalized interconnection of memories, or as specialized computers with geometrically related memory interconnections and controlled by one or more instruction streams.

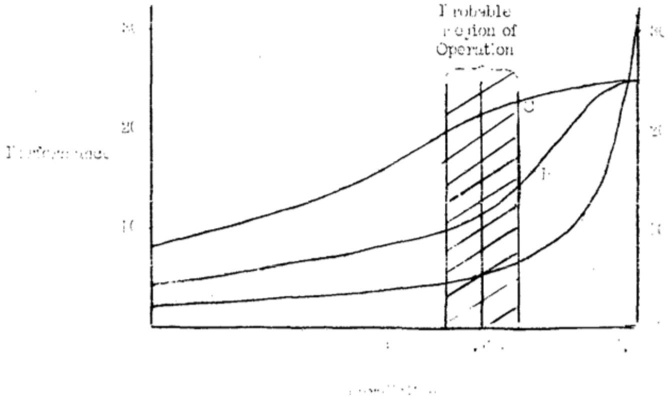
Demonstration is made of the continued validity of the single processor approach and of the weaknesses of the multiple processor approach in terms of application to real problems and their attendant irregularities.

The arguments presented are based on statistical

reference will be one of the most thorough analyses of relative computer capabilities currently published—"Changes in Computer Performance," *Datamation*, September 1966, Professor Kenneth E. Knight, Stanford School of Business Administration.

The first characteristic of interest is the fraction of the computational load which is associated with data management housekeeping. This fraction has been very nearly constant for about ten years, and accounts for 40% of the executed instructions in production runs. In an entirely dedicated special purpose environment this might be reduced by a factor of two, but it is highly improbably that it could be reduced by a factor of three. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor. The non-housekeeping part of the problem could exploit at most a processor of performance three to four times the performance of the housekeeping processor. A fairly obvious conclusion which can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.

Data management housekeeping is not the only problem to plague oversimplified approaches to high speed computation. The physical problems which are of practical interest tend to have rather significant complications. Examples of these complications are as follows: boundaries are likely to be irregular; interiors are likely to be inhomogeneous; computations required may be dependent on the states of the variables at each point; propagation rates of different physical effects may be quite different; the rate of convergence, or convergence at all, may be strongly dependent on sweeping through the array along different axes on succeeding passes; etc. The effect of each of these complications is very severe on any computer organization based on geometrically related processors in a paralleled processing system. Even the existence of regular rectangular boundaries has the interesting property that for spatial dimension of N there are 3^N different point geometries to be dealt with in a nearest neighbor computation. If the second nearest neighbor were also involved, there would be 5^N different point geometries to contend with. An irregular boundary compounds this problem as does an inhomogeneous interior. Computations which are dependent on the states of variables would require the processing at each point to consume approximately the same computational time as the sum of computations of all physical effects within a



Amdahl’s Law (1967)

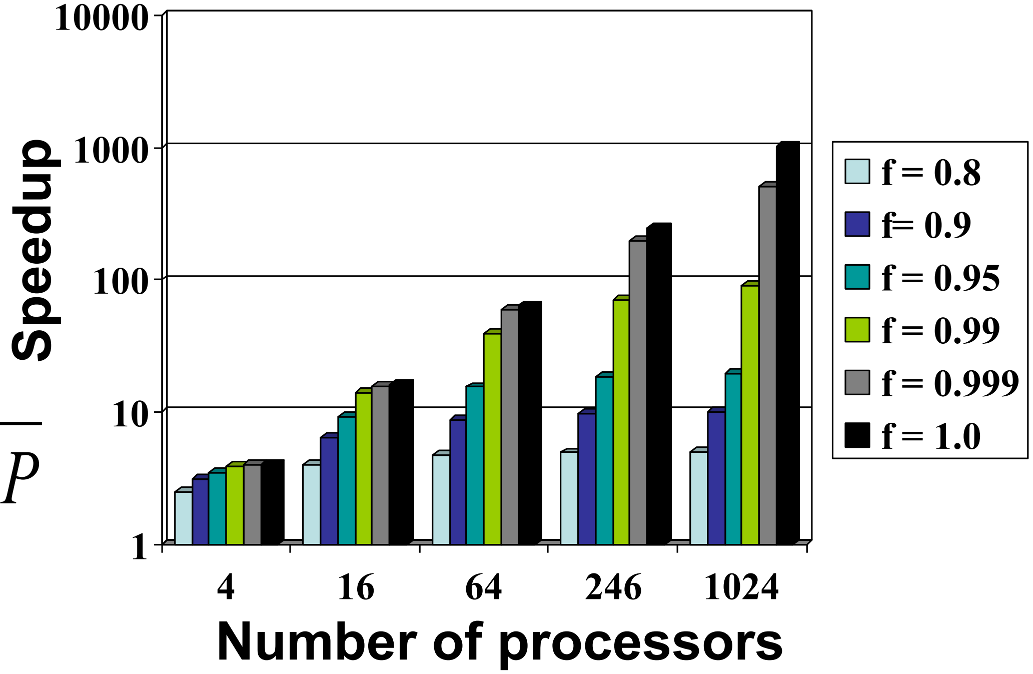
In 1967 Gene Amdahl of Cray Computer formulated his famous pessimistic formula about the speedup available from concurrency. If f is the fraction of the code that is parallelizable and P is the number of processors available, then the time T_P to run on P nodes as a function of the time T_1 to run on 1 is:

$$T_P = f\,\frac{T_1}{P} + (1-f)T_1$$

What is the speed-up available?

$$Speedup \equiv \frac{T_1}{T_P} = \frac{1}{(1-f)+f/P}$$

$$\lim_{P\to\infty} Speedup = \frac{1}{(1-f)}$$



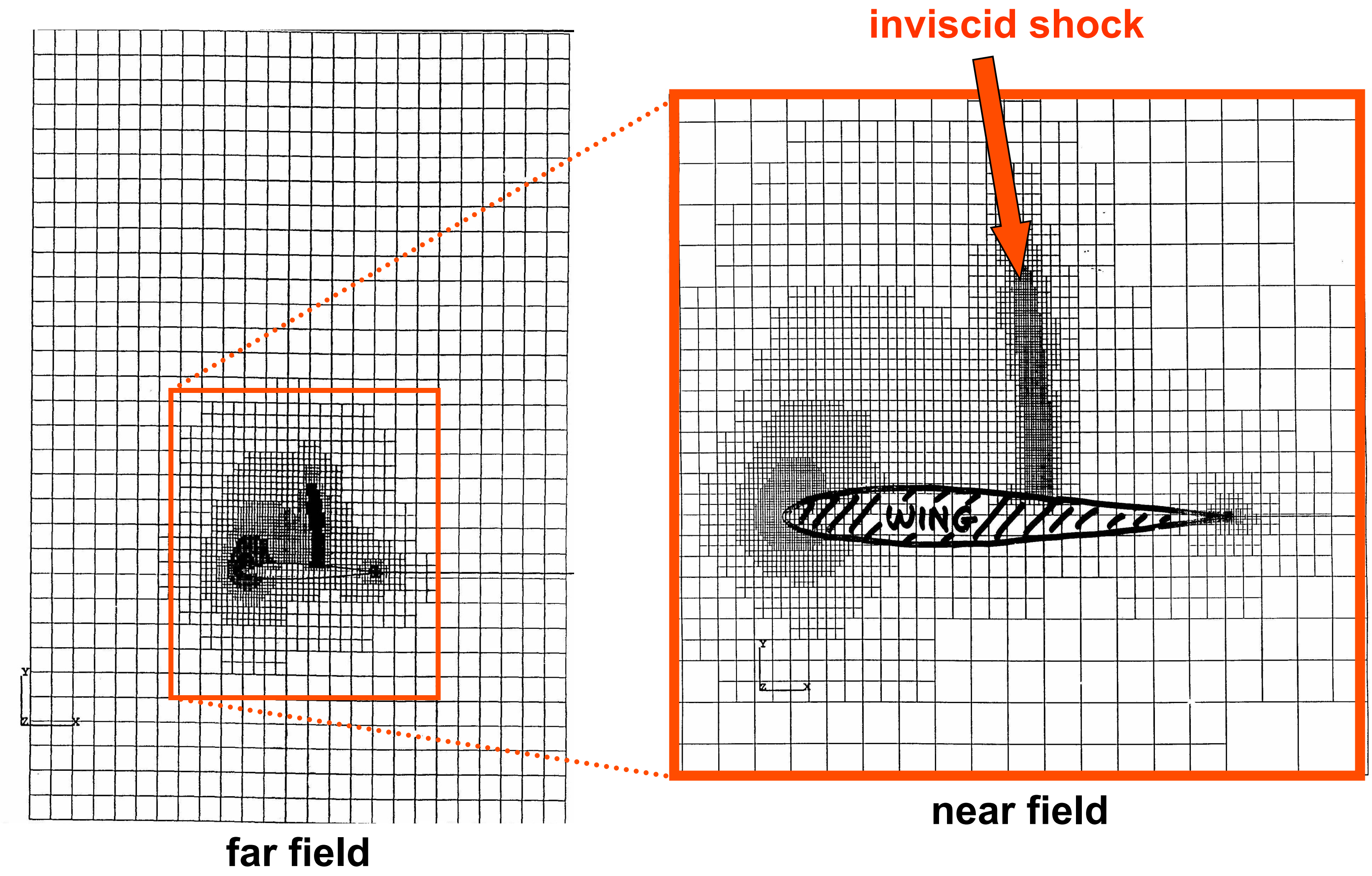
Most basic issue: algorithm!

- Our prime problem and opportunity, however, is not architectural!
- It is that we are computing more data than we need!
- We should compute only *where* needed and only *what* needed
- Algorithms that do this effectively, while controlling accuracy, are called *adaptive*

Adaptive algorithms

- For an airplane, we need 1 cm (or better) resolution only in *boundary layers* and *shocks*
  - Elsewhere, much coarser (e.g., 10 cm) mesh resolution is sufficient
- A factor of 10 less resolution in each dimension reduces computational requirements by 10^3

Adaptive Cartesian mesh

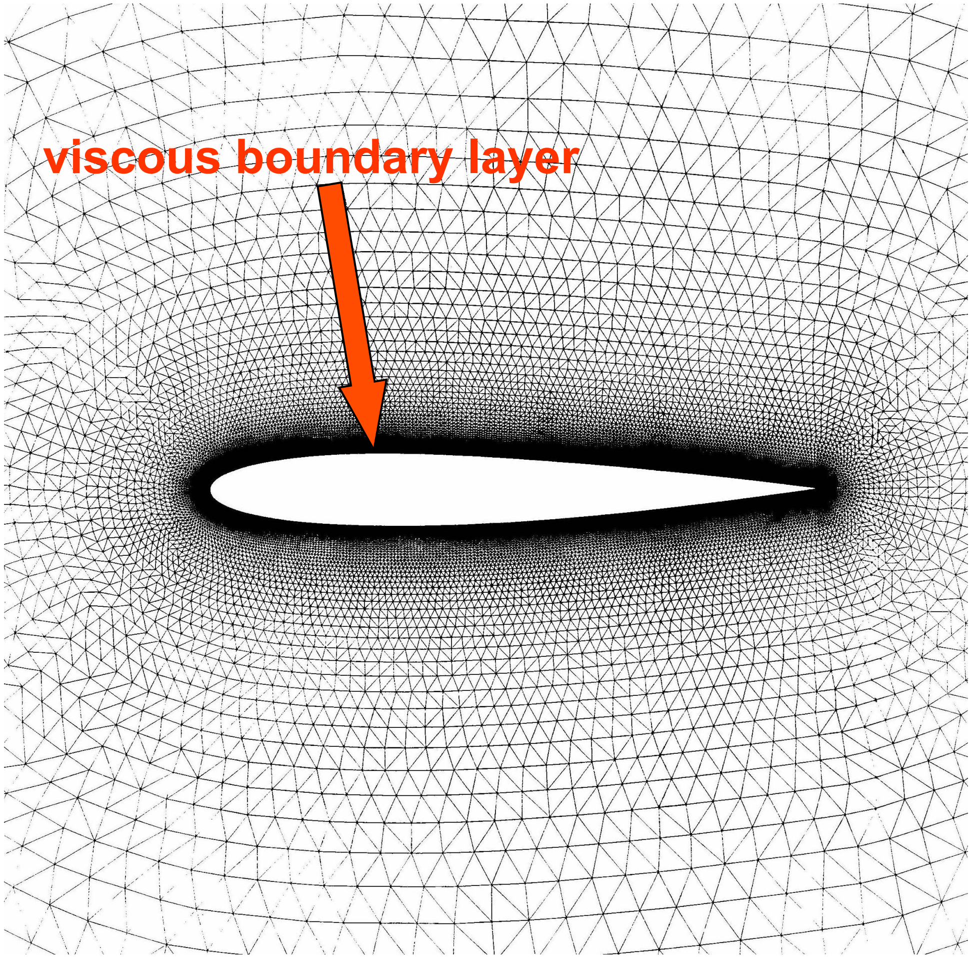


inviscid shock

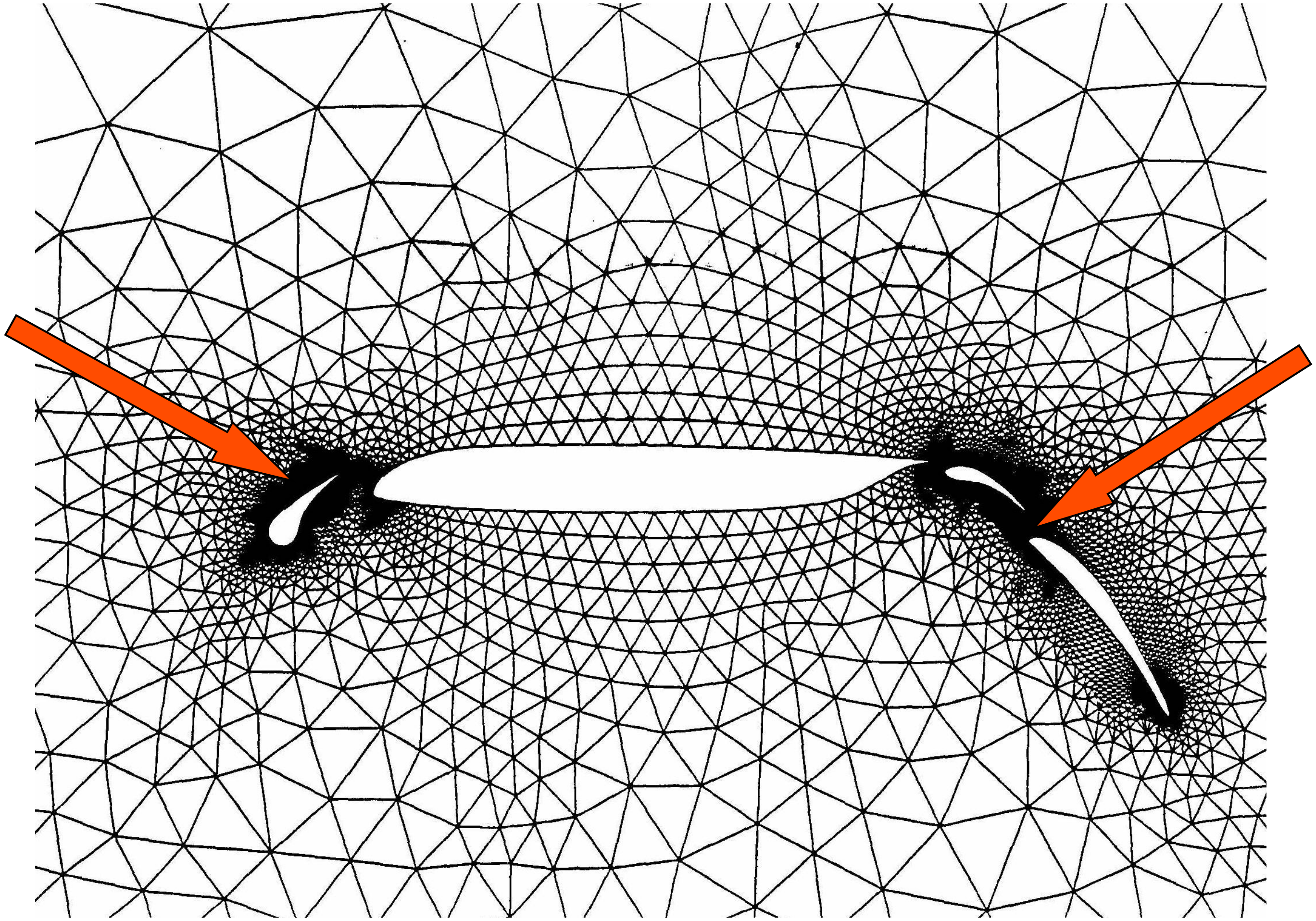
far field

near field

Adaptive triangular mesh



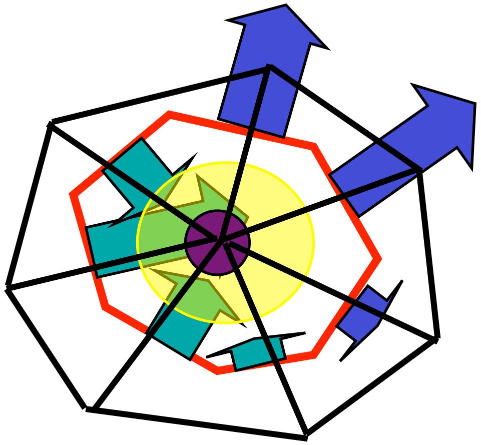
Unstructured grid for complex geometry



slat

flaps

How does the discretization work?
Just like before, except for geometry



Construct "grid" of triangles

Construct "control volumes" surrounding each vertex

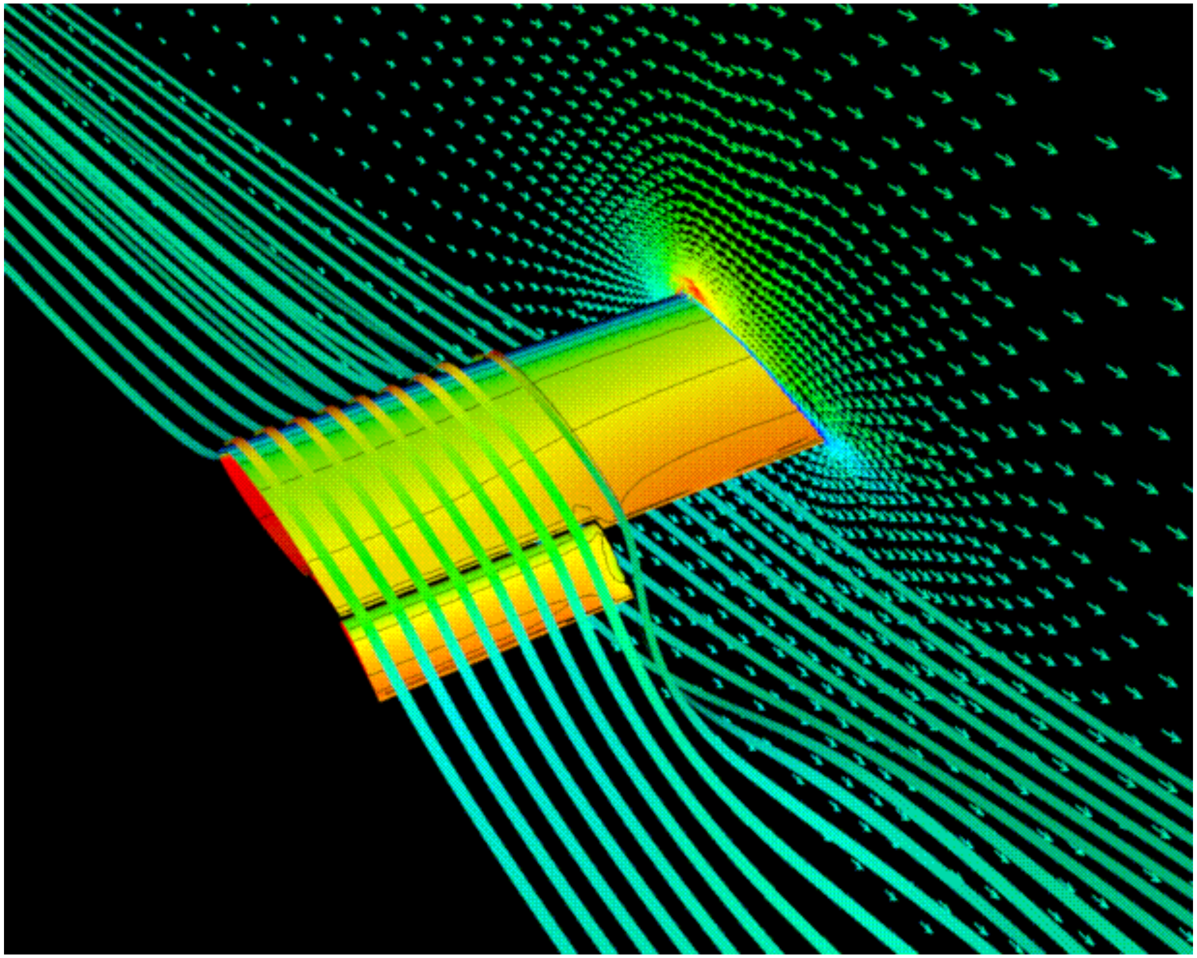
Compute effluxes

Compute influxes

Compute internal sources

Finally, sum all fluxes and sources (with proper sign) and adjust value at vertex; then loop over all such vertices.

Scientific visualization adds insight



Computer becomes an experimental laboratory, like a windtunnel, and can be outfitted with diagnostics and imaging intuitive to windtunnel experimentalists.

Benefits of adaptivity, cont.

- If adaptivity reduces storage and operation requirements by 1000, this leaves 8 × 10^{12} operations per second, or 8 Tflop/s
- This is readily available today
  - for a modest price compared to cost of plane
  - e.g., in a handful of GPUs (!)
  - drawing a few KiloWatts
- However, detailed aerodynamics codes are not routinely employed in automated real-time loops
  - providing accurate initial and boundary conditions is a challenge
  - reduced-order models based on ready observables are more practical

Problems with adaptivity

- Difficult to guarantee accuracy
  - much more mathematics to be done for realistic computer models
- Difficult to program
  - complex dynamic data structures
- Can’t always help
  - sometimes resolution really is needed everywhere, e.g., in wave propagation problems
- May not work well with pipelining and parallel techniques
  - tension between conflicting needs of local focusing of computation and global regularity

Conclusions

- Parallel networks of commodity pipelined microprocessors offer cheap, fast, powerful supercomputing
- Algorithm development offers better, more efficient ways to use *all* computers
- Riding the waves of architectural advancements and creating improved simulation techniques opens up new vistas for computational science across the spectrum

Summary

- Application – computational aerodynamics
- Numerical analysis – discretization of conservation laws
- Courant stability limit (1928) – speed of sound
- Hardware limit– speed of light
- Computer architecture – pipelining & parallelism
- Moore’s Law (1965)
- Amdahl’s Law (1967)
- Power of adaptive, optimal algorithms
- Bell Prizes (1988 onwards)
- Cost-effective future of simulation

Slide credits

- Kyle Anderson (NASA)
- David Patterson (UC Berkeley)
- Geoffrey Fox (U Indiana)
- Bill Gropp (Argonne Nat Lab)
- Douglas Ball, David Young, and Venkatasubramanian Venkatakrishnan (Boeing)