



THE UNIVERSITY OF
CHICAGO

Lecture 5

January 18, 2011

4.3 GENERAL CASE: SOLVING THE ACTUAL TR PROBLEM (DOGLEG DOES NOT QUITE DO IT)

Trust Region Equation

Theorem 4.1.

The vector p^ is a global solution of the trust-region problem*

$$\min_{p \in \mathbb{R}^n} m(p) = f + g^T p + \frac{1}{2} p^T B p, \quad \text{s.t. } \|p\| \leq \Delta, \quad (4.7)$$

if and only if p^ is feasible and there is a scalar $\lambda \geq 0$ such that the following conditions are satisfied:*

$$(B + \lambda I)p^* = -g, \quad (4.8a)$$

$$\lambda(\Delta - \|p^*\|) = 0, \quad (4.8b)$$

$$(B + \lambda I) \quad \text{is positive semidefinite.} \quad (4.8c)$$

Theory of Trust Region Problem

Global convergence
away from saddle
point

Theorem 4.8.

Suppose that the assumptions of Theorem 4.6 are satisfied and in addition that f is twice continuously differentiable in the level set S . Suppose that $B_k = \nabla^2 f(x_k)$ for all k , and that the approximate solution p_k of (4.3) at each iteration satisfies (4.52) for some fixed $\gamma > 0$. Then $\lim_{k \rightarrow \infty} \|g_k\| = 0$.

If, in addition, the level set S of (4.24) is compact, then either the algorithm terminates at a point x_k at which the second-order necessary conditions (Theorem 2.3) for a local solution hold, or else $\{x_k\}$ has a limit point x^ in S at which the second-order necessary conditions hold.*

Fast Local
Convergence

Theorem 4.9.

Let f be twice Lipschitz continuously differentiable in a neighborhood of a point x^ at which second-order sufficient conditions (Theorem 2.4) are satisfied. Suppose the sequence $\{x_k\}$ converges to x^* and that for all k sufficiently large, the trust-region algorithm based on (4.3) with $B_k = \nabla^2 f(x_k)$ chooses steps p_k that satisfy the Cauchy-point-based model reduction criterion (4.20) and are asymptotically similar to Newton steps p_k^N whenever $\|p_k^N\| \leq \frac{1}{2}\Delta_k$, that is,*

$$\|p_k - p_k^N\| = o(\|p_k^N\|). \quad (4.53)$$

Then the trust-region bound Δ_k becomes inactive for all k sufficiently large and the sequence $\{x_k\}$ converges superlinearly to x^ .*

How do we solve the subproblem?

- Very sophisticated approach based on theorem on structure of TR solution, eigenvalue analysis and/or an “inner” Newton iteration.
- Foundation: Find Solution for

$$p(\lambda) = -(B + \lambda I)^{-1}g$$

$$\|p(\lambda)\| = \Delta.$$

How do I find such a solution?

$$B = Q\Lambda Q^T \quad \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n),$$

$$p(\lambda) = -Q(\Lambda + \lambda I)^{-1}Q^T g = -\sum_{j=1}^n \frac{q_j^T g}{\lambda_j + \lambda} q_j,$$

, by orthonormality of q_1, q_2, \dots, q_n :

$$\|p(\lambda)\|^2 = \sum_{j=1}^n \frac{(q_j^T g)^2}{(\lambda_j + \lambda)^2}.$$

TR problem has a solution

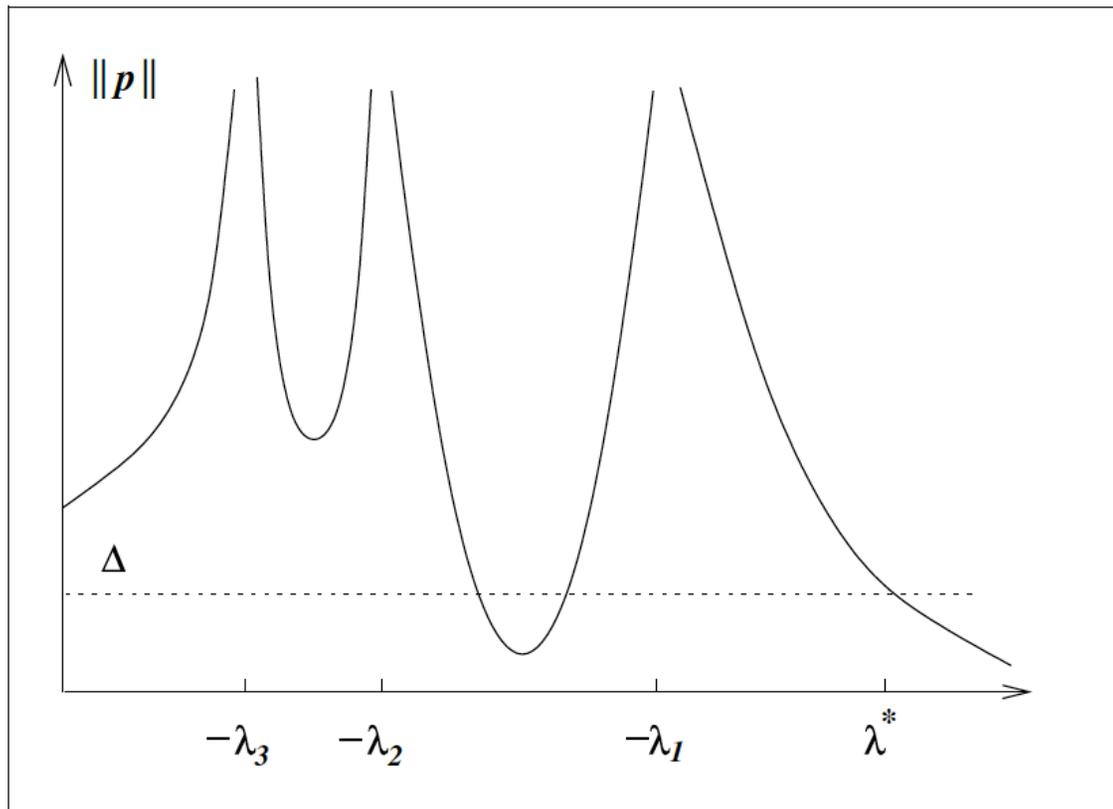


Figure 4.5 $\|p(\lambda)\|$ as a function of λ .

$$\lim_{\lambda \rightarrow \infty} \|p(\lambda)\| = 0. \quad q_j^T g \neq 0 \implies \lim_{\lambda \rightarrow -\lambda_j} \|p(\lambda)\| = \infty.$$

Practical (INCOMPLETE) algorithm

$$\phi_2(\lambda) = \frac{1}{\Delta} - \frac{1}{\|p(\lambda)\|}, \quad \lambda^{(\ell+1)} = \lambda^{(\ell)} - \frac{\phi_2(\lambda^{(\ell)})}{\phi_2'(\lambda^{(\ell)})}.$$

Algorithm 4.3 (Trust Region Subproblem).

Given $\lambda^{(0)}$, $\Delta > 0$:

for $\ell = 0, 1, 2, \dots$

Factor $B + \lambda^{(\ell)}I = R^T R$;

Solve $R^T R p_\ell = -g$, $R^T q_\ell = p_\ell$;

Set

$$\lambda^{(\ell+1)} = \lambda^{(\ell)} + \left(\frac{\|p_\ell\|}{\|q_\ell\|} \right)^2 \left(\frac{\|p_\ell\| - \Delta}{\Delta} \right);$$

end (for).

It generally gives a machine precision solution in 2-3 iterations
(Cholesky)

The Hard Case

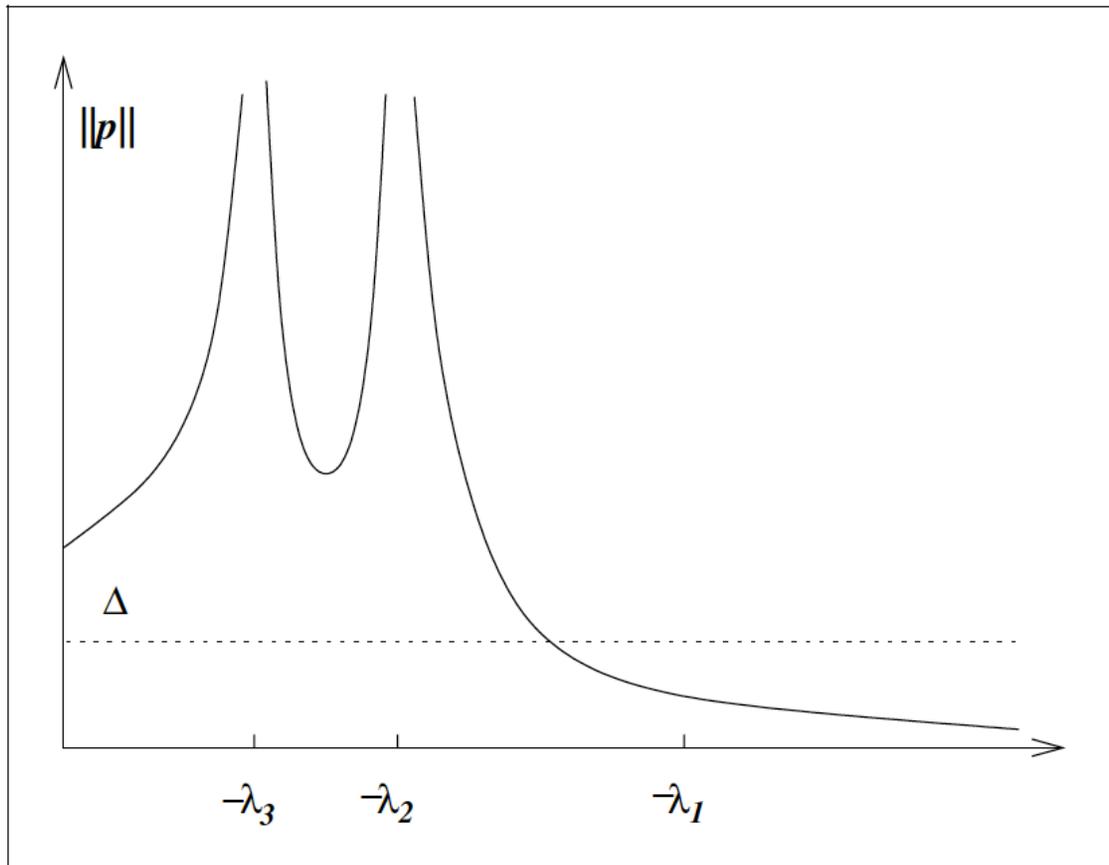


Figure 4.7 | The hard case: $\|p(\lambda)\| < \Delta$ for all $\lambda \in (-\lambda_1, \infty)$.

$$q_j^T g = 0$$

$$\lambda = -\lambda_1 \Rightarrow p = \sum_{j:\lambda_j \neq \lambda_1} \frac{q_j^T g}{\lambda_j - \lambda_1} q_j$$



$$p(\tau) = \sum_{j:\lambda_j \neq \lambda_1} \frac{q_j^T g}{\lambda_j - \lambda_1} q_j + \tau q_1$$



$$\exists \tau \quad \|p(\tau)\| = \Delta^k$$

If double root, things continue to be complicated ...

Summary and Comparisons

- Line search problems have easier subproblems (if we modify Cholesky).
- But they cannot be guaranteed to converge to a point with positive semidefinite Hessian.
- Trust-region problems can, at the cost of solving a complicated subproblem.
- Dogleg methods leave “between” these two situations.



THE UNIVERSITY OF
CHICAGO

Section 5

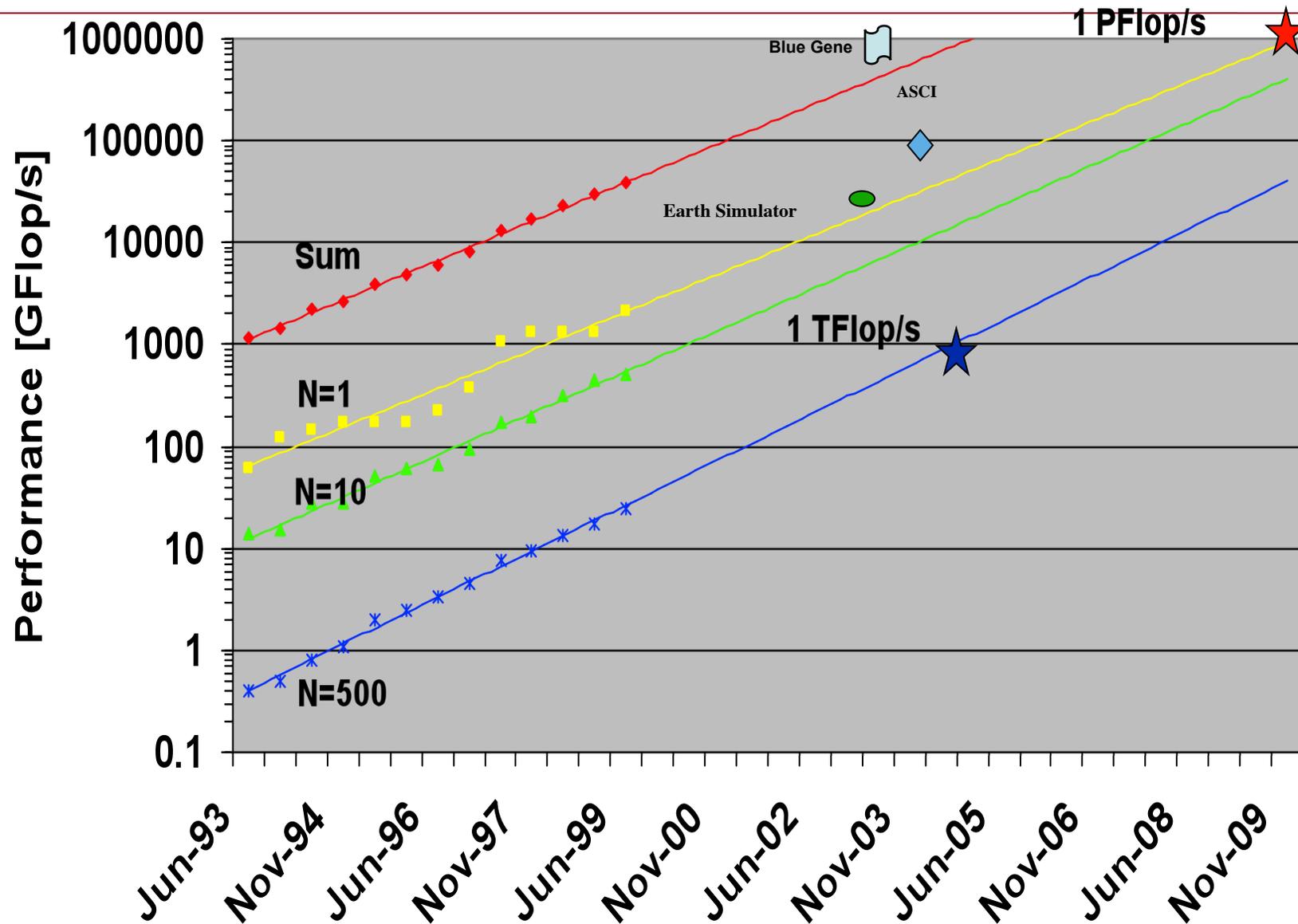
Matrix-free methods

Mihai Anitescu

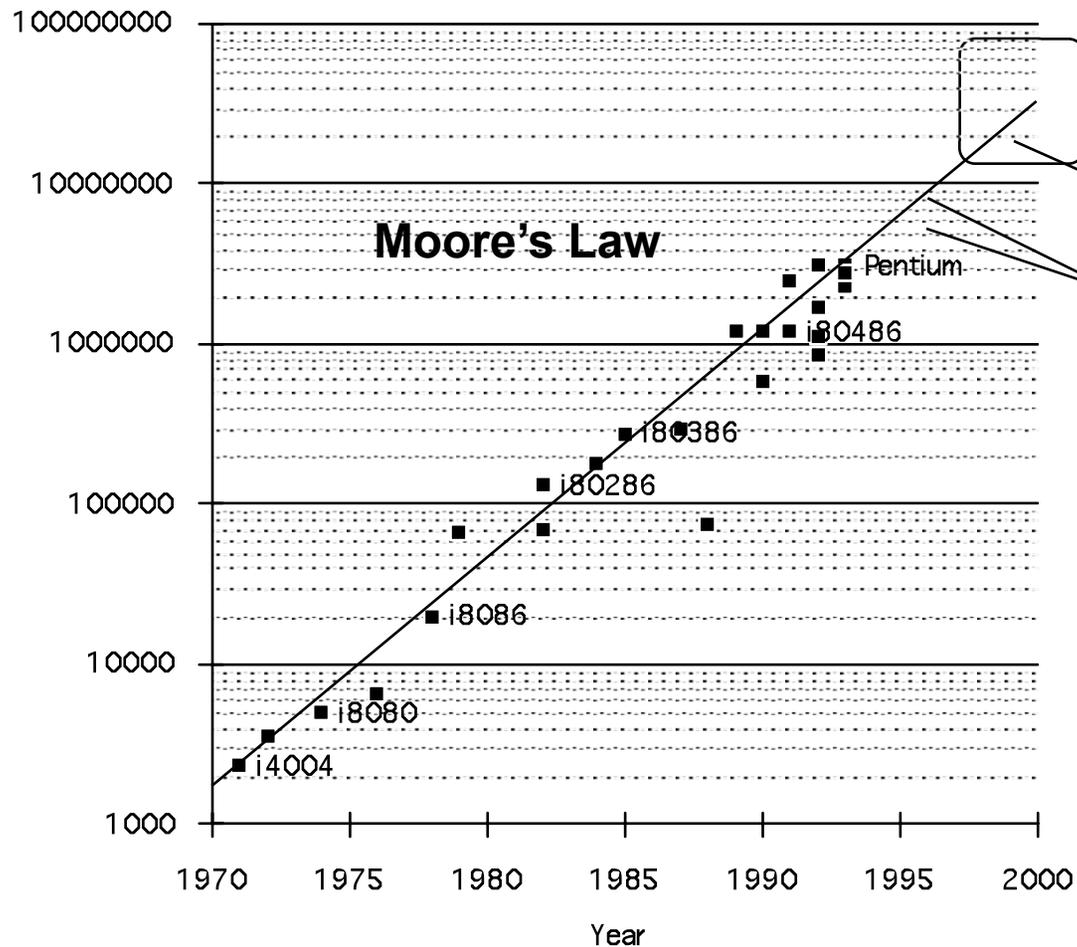
5.1 PARALLEL COMPUTING AND MATRIX-FREE METHODS

5.1.1 SOME DISCUSSION FOR PARALLEL COMPUTING/ SUPERCOMPUTING

The appeal of Supercomputing: Top500



Technology Trends: Microprocessor Capacity



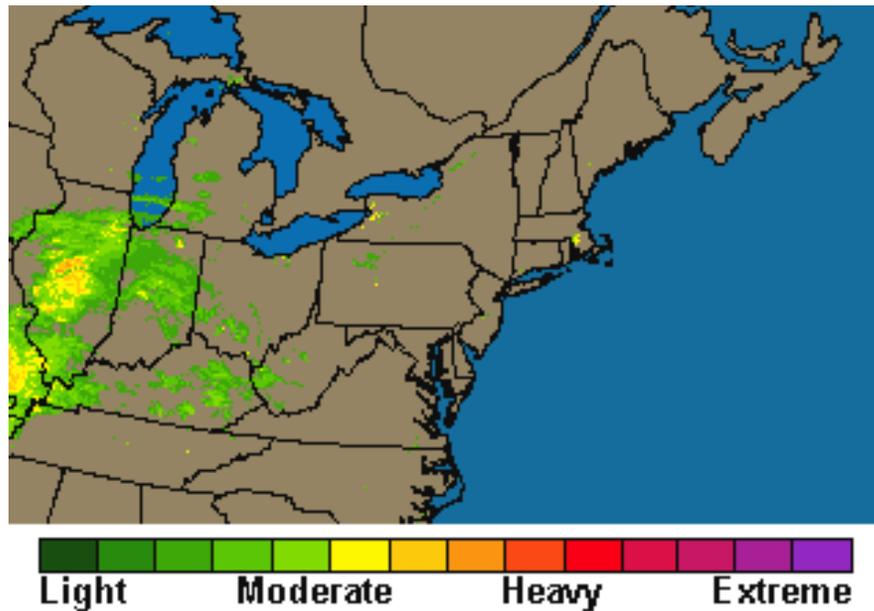
Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

2X transistors/Chip
Every 1.5 years

Called "Moore's Law":

What are They Used For

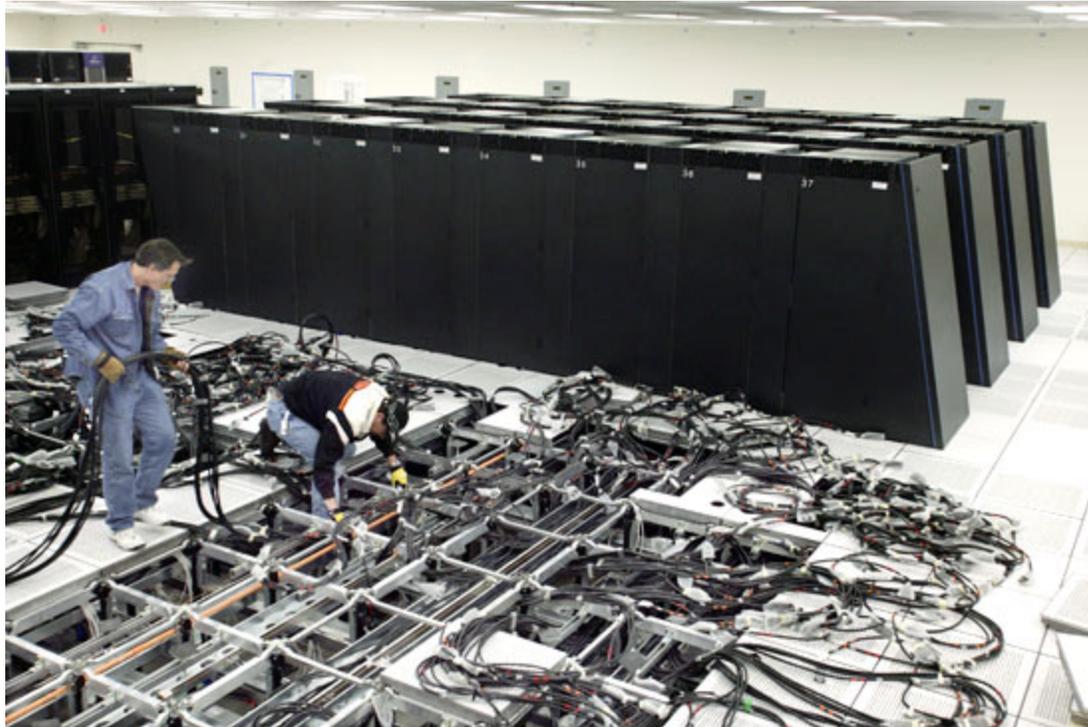
- Climate prediction & Weather forecasting



What are They Used For (cont.)

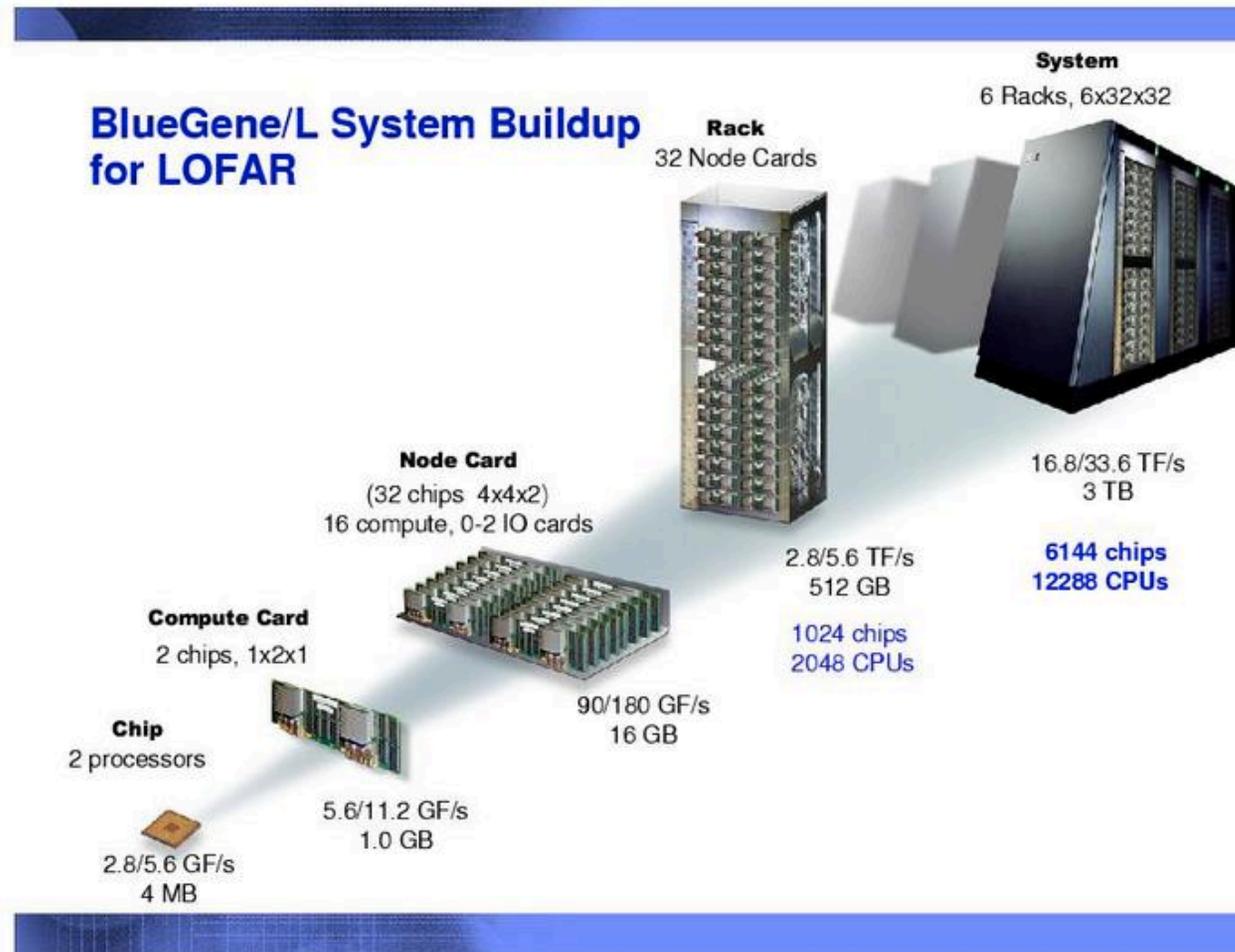
- Computational chemistry
- Crash analysis
- Cryptography
- Nuclear simulation
- Structural analysis

Blue Gene L



From http://i.n.com.com/i/ne/p/photo/BlueGeneL_03_550x366.jpg

Packaging and Scale-Up

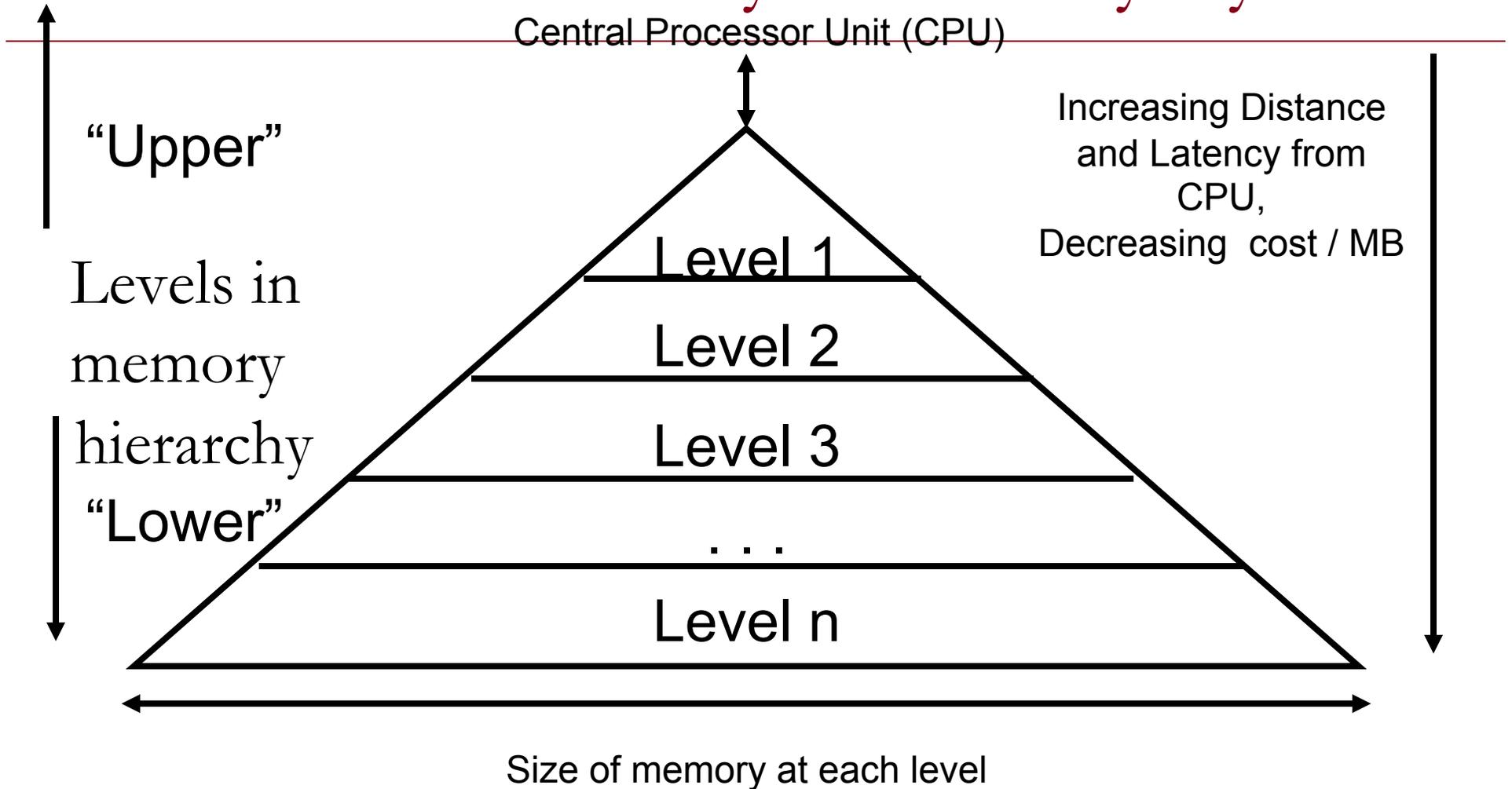


From http://www.rug.nl/cit/diensten/system_services/nieuwsbrief/200504/bouwbluegene-buildup-800x571.jpg

In what way are they different from programming regular computing.

- Normal users tend to ignore the effects of data movement (to/from RAM or to a less extent cache memory).
- In Supercomputers, processors (100ks) are connected by a network, which although fast, is nothing like processor to cache or processor to RAM.
- The memory hierarchy is “deep” with SLOW low levels (e.g RAM on a different board or disk). So the cost of data movement can no longer be ignored.

Memory Hierarchy Pyramid



(data cannot be in level i unless also in $i+1$)

5.1.2 LINEAR ALGEBRA ON MASSIVELY PARALLEL COMPUTERS

Solving Systems of Equation in Parallel, High Performance

Don'ts:

Computing ENVS

- **Don't** invert the matrix ($\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$). That's much more costly than solving directly, and the matrix storage cost will be unbearable.
- **Don't** write your own solver code. There are people who devote their whole careers to writing solvers. They know a lot more about writing solvers than we do.

Solving Do's

Do's:

- Do use standard, portable solver libraries.
- Do use a version that's tuned for the platform you're running on, if available.
- Do use the information that you have about your system to pick the most efficient solver.

Linear Algebra Libraries

- BLAS
- LAPACK
- ScaLAPACK
- PETSc

- The **Basic Linear Algebra Subprograms** (BLAS) are a set of low level linear algebra routines:
 - Level 1: Vector-vector (e.g., dot product).
 - Level 2: Matrix-vector (e.g., matrix-vector multiply).
 - Level 3: Matrix-matrix (e.g., matrix-matrix multiply)

BLAS

- Many linear algebra packages, including LAPACK, ScaLAPACK and PETSc, are built on top of BLAS.
- Most supercomputer vendors have versions of BLAS that are highly tuned for their platforms.

LAPACK

LAPACK (Linear Algebra PACKage) solves dense or special-case sparse systems of equations depending on matrix properties such as:

- Precision: single, double
- Data type: real, complex
- Shape: diagonal, bidiagonal, tridiagonal, banded, triangular, trapezoidal, Hessenberg, general dense
- Properties: orthogonal, positive definite, Hermetian (complex), symmetric, general

LAPACK is built on top of BLAS,

LAPACK Example

```
REAL, DIMENSION (numrows, numcols) :: A
REAL, DIMENSION (numrows)           :: B
REAL, DIMENSION (numrows)           :: X
INTEGER, DIMENSION (numrows)        :: pivot
INTEGER :: row, col, info, numrhs = 1
DO row = 1, numrows
  B(row) = ...
END DO
DO col = 1, numcols
  DO row = 1, numrows
    A(row, col) = ...
  END DO
END DO
CALL sgesv(numrows, numrhs, A, numrows, pivot, &
&          B, numrows, info)
DO col = 1, numcols
  X(col) = B(col)
END DO
```

LAPACK: a Library and an API

LAPACK is a library that you can download for free from the Web:

`www.netlib.org`

But, it's also an Application Programming Interface (API): a definition of a set of routines, their arguments, and their behaviors.

So, anyone can write an implementation of LAPACK.

LAPACK Performance

Because LAPACK uses BLAS, it's about as fast as BLAS.

In fact, an older version of LAPACK, called LINPACK, is used to determine the top 500 supercomputers in the world.

ScaLAPACK

ScaLAPACK is the distributed parallel (MPI) version of LAPACK. It actually contains only a subset of the LAPACK routines, and has a somewhat awkward Application Programming Interface (API).

Like LAPACK, ScaLAPACK is also available from

`www.netlib.org`.

- **PETSc** (Portable, Extensible Toolkit for Scientific Computation) is a solver library for sparse matrices that uses distributed parallelism (MPI).
- PETSc is designed for general sparse matrices with no special properties, but it also works well for sparse matrices with simple properties like banding and symmetry.
- It has a simpler, more intuitive Application Programming Interface than ScaLAPACK.

Pick Your Solver Package

- Dense Matrix
 - Serial: LAPACK
 - Shared Memory Parallel: vendor-tuned LAPACK
 - Distributed Parallel: ScaLAPACK
- Sparse Matrix: PETSc

Matrix Free Methods for Parallel Computations

- Many problems have simple, sparse matrices associated with the linear algebra (e.g., example from Homework 2).
- Nevertheless, factorization is extremely expensive on parallel architectures due to processor-to-processor **data motion** needs of pivoting, (as well as storage needs).
- **Solution? Use methods (e.g., Krylov space or relaxation methods) that only multiply the matrix A times a vector x , code to calculate $y=Ax$ can be written instead of storing the matrix A .**
- This reduces the cost of the computer (which is mostly memory chips) and allows for vastly larger simulations.
- **It also results in very fast code, if BLAS is optimized, which is a much easier proposition than optimizing factorization.**

5.2 CONJUGATED GRADIENTS METHOD (AND PRECONDITIONED)

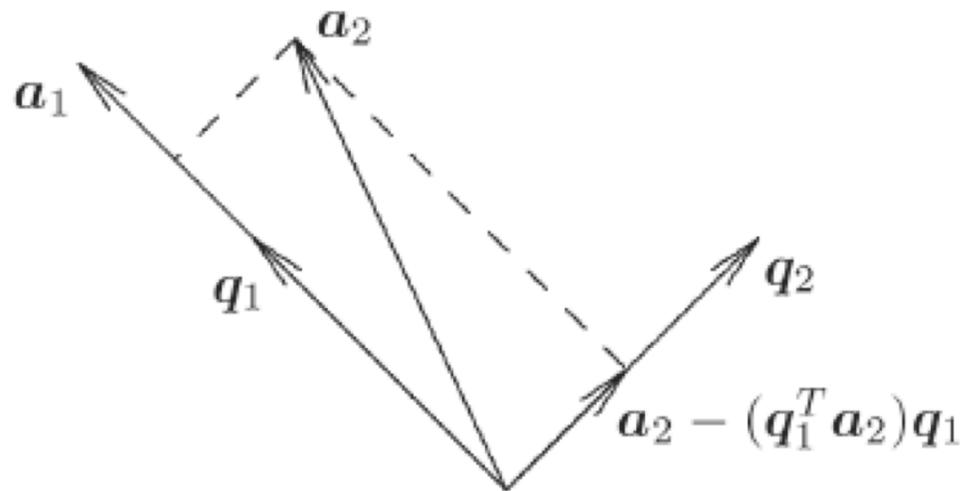
From “Conjugated Gradients without the Agonizing Pain”-

- “The Conjugate Gradient Method is the most prominent iterative method for solving sparse systems of linear equations. Unfortunately, many textbook treatments of the topic are written with neither illustrations nor intuition, and their victims can be found to this day babbling senselessly in the corners of dusty libraries. For this reason, a deep, geometric understanding of the method has been reserved for the elite brilliant few who have painstakingly decoded the mumblings of their forebears.” -Schewchuck

5.2.1 GRAM-SCHMIDT

Gram-Schmidt orthogonalization

- Given vectors a_1 and a_2 , we seek orthonormal vectors q_1 and q_2 having same span
- This can be accomplished by subtracting from second vector its projection onto first vector and normalizing both resulting vectors, as shown in diagram



Gram-Schmidt algorithm

- Process can be extended to any number of vectors a_1, \dots, a_k , orthogonalizing each successive vector against all preceding ones, giving *classical Gram-Schmidt* procedure

```
for  $k = 1$  to  $n$   
     $q_k = a_k$   
    for  $j = 1$  to  $k - 1$   
         $r_{jk} = q_j^T a_k$   
         $q_k = q_k - r_{jk} q_j$   
    end  
     $r_{kk} = \|q_k\|_2$   
     $q_k = q_k / r_{kk}$   
end
```

- Resulting q_k and r_{jk} form reduced QR factorization of A

QR Factorization

- Given $m \times n$ matrix A , with $m > n$, we seek $m \times m$ orthogonal matrix Q such that

$$A = Q \begin{bmatrix} R \\ O \end{bmatrix}$$

where R is $n \times n$ and upper triangular

- Linear least squares problem $Ax \cong b$ is then transformed into triangular least squares problem

$$Q^T Ax = \begin{bmatrix} R \\ O \end{bmatrix} x \cong \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = Q^T b$$

which has same solution, since

$$\|r\|_2^2 = \|b - Ax\|_2^2 = \|b - Q \begin{bmatrix} R \\ O \end{bmatrix} x\|_2^2 = \|Q^T b - \begin{bmatrix} R \\ O \end{bmatrix} x\|_2^2$$

5.2.2 METHOD OF CONJUGATED DIRECTIONS

Conjugated Search Directions

- Conjugated=orthogonal
- Positive definite A ; linear system is equivalent to minimization.

$$A \succ 0 \Rightarrow Ax = b \Leftrightarrow \min_{x \in \mathbb{R}^n} \phi(x) = \frac{1}{2} x^T Ax - b^T x$$

- **IDEA 1:** A -conjugate directions:

$$S_k = \{p_0, p_1, \dots, p_{k-1}\} \quad p_i^T A p_j = 0 \quad 0 \leq i \neq j \leq k-1$$

$$x_k = \min_{x \in x_0 + S_k} \phi(x); \quad \phi(x) = \frac{1}{2} x^T Ax - b^T x$$

Arithmetic of A-Conjugated Search Directions

- A-conjugated direction search methods: the subspace problems are much simpler:

$$S_k = \{p_0, p_1, \dots, p_{k-1}\} \quad p_i^T A p_j = 0 \quad 0 \leq i \neq j \leq k-1$$

$$\Leftrightarrow \min_{\{\alpha_i\}_{i=0}^{k-1}} \frac{1}{2} \left(\sum_{i=0}^{k-1} \alpha_i p_i + x_0 \right)^T A \left(\sum_{i=0}^{k-1} \alpha_i p_i + x_0 \right) - b^T \left(\sum_{i=0}^{k-1} \alpha_i p_i + x_0 \right) =$$

$$\min_{\{\alpha_i\}_{i=0}^{k-1}} \frac{1}{2} \left(\sum_{i=0}^{k-1} \alpha_i p_i \right)^T A \left(\sum_{i=0}^{k-1} \alpha_i p_i \right) + (A x_0 - b)^T \left(\sum_{i=0}^{k-1} \alpha_i p_i \right) \quad \text{A-Conjugated Directions} =$$

$$\min_{\{\alpha_i\}_{i=0}^{k-1}} \sum_{i=0}^{k-1} \frac{1}{2} \alpha_i^2 p_i^T A p_i + \sum_{i=0}^{k-1} \alpha_i r_0^T p_i = \sum_{i=0}^{k-1} \left[\frac{1}{2} \alpha_i^2 p_i^T A p_i + \alpha_i r_0^T p_i \right];$$

$$\text{where } r_0 = (A x_0 - b) = \nabla_x \phi(x_0)$$

Arithmetic of A-Conjugated Search

Directions-Consequences

$$\min_{\{\alpha_i\}_{i=0}^{k-1}} \sum_{i=0}^{k-1} \left[\frac{1}{2} \alpha_i^2 p_i^T A p_i + \alpha_i r_0^T p_i \right]; \quad \Rightarrow \alpha_i = -\frac{r_0^T p_i}{p_i^T A p_i}; \Rightarrow$$

$$x_k = x_0 + \sum_{i=0}^{k-1} \left[-\frac{r_0^T p_i}{p_i^T A p_i} p_i \right] = x_{k-1} + \alpha_{k-1} p_{k-1} \quad (C1)$$

$$\nabla \phi(x_k) = A x_k - b = r_k = r_{k-1} + \alpha_{k-1} A p_{k-1} \quad (C2)$$

$$r_k^T p_i = \nabla \phi(x_k) p_i = \left. \frac{\partial \phi}{\partial \alpha_i} \right|_{x=x_k} = 0; \quad i = 1, 2, \dots, k-1 \quad (C3)$$

$$\phi(x_k) = \min_{\alpha} \phi(x_{k-1} + \alpha p_{k-1}) \quad (C4)$$

Conjugated Directions Methods

- Are truly minimization- line search methods but their quadratic nature makes the problem “simple”.
- The recursion in the variable is very simple.
- The residuals (the gradients) are orthogonal to the search direction space.
- But, how do I find the A -conjugate search directions?
- **IDEA 2:** The space of the search directions should coincide with the space of the residuals.
- In principle, achievable with Gram-Schmidt using the A inner product, **but maybe it is simpler**

5.2.3 KRYLOV SPACES

Obtaining the A-conjugated Search Directions

- **IDEA 2:** The space of the search directions should coincide with the space of the residuals.

$$S_k = \{p_0, p_1, \dots, p_{k-1}\} = \{r_0, r_1, \dots, r_{k-1}\}$$

- Consequence K1: the gradients (residuals themselves) are orthogonal, that is they are conjugated.

$$r_k \perp S_k (C3) \Rightarrow r_k^T r_i = 0 (K1); \quad 0 \leq i \neq k$$

- This fact gives the methods its name: The method of **conjugated gradients**.

Obtaining the A-conjugated Search Directions

- Consequence K2: the search space = the gradient space from a Krylov space.

$$S_k = \{p_0, p_1, \dots, p_{k-1}\} = \{r_0, r_1, \dots, r_{k-1}\}$$

$$r_k = r_{k-1} + \alpha_{k-1} A p_{k-1} \text{ (C2)} \Rightarrow S_{k+1} \subset S_k + A S_k$$

$$\text{By induction and (K1)} \quad S_k = \{r_0, A r_0, \dots, A^{k-1} r_0\} = K_k(A, r_0) \quad \text{(K2)}$$

- Consequence K3 (the subtlest): **computation of the next search direction needs only two terms !!!**

$$r_k \perp S_k \text{ (K1)} = K_k \text{ (K2)} \Rightarrow r_k \perp A S_{k-1} = A K_{k-1} \subset K_k \text{ (K2)} \Rightarrow r_k^T A p_i = 0, \quad i = 0, 1, \dots, k-2$$

$$d_k = -r_k + \beta_k p_{k-1} \Rightarrow d_k^T A p_i = 0 \quad i = 1, 2, \dots, k-2 \quad !!! \Rightarrow \beta_k = \frac{p_{k-1}^T A r_k}{p_{k-1}^T A r_k}; \quad p_k = -r_k + \beta_k p_{k-1}$$

METHOD OF CONJUGATED GRADIENTS VERSION 1

Algorithm 5.1 (CG–Preliminary Version).

Given x_0 ;

Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow -r_0$, $k \leftarrow 0$;

while $r_k \neq 0$

$$\alpha_k \leftarrow -\frac{r_k^T p_k}{p_k^T A p_k};$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$

$$r_{k+1} \leftarrow Ax_{k+1} - b;$$

$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T A p_k}{p_k^T A p_k};$$

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k;$$

$$k \leftarrow k + 1;$$

end (while)

SOME SIMPLIFICATIONS

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k;$$

$$r_k^T p_i = 0, \quad \text{for } i = 0, 1, \dots, k-1,$$

$$\alpha_k A p_k = r_{k+1} - r_k,$$

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}.$$



$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

I will ask you to prove it.

CG: PRACTICAL VERSION (MINIMAL STORAGE)

Algorithm 5.2 (CG).

Given x_0 ;

Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow -r_0$, $k \leftarrow 0$;

while $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k};$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$

$$r_{k+1} \leftarrow r_k + \alpha_k A p_k;$$

$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k};$$

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k;$$

$$k \leftarrow k + 1;$$

end (while)

NEEDS ONLY 1 MATRIX-VECTOR MULTIPLICATION PER STEP.
AX NEVER FORMED AS BEFORE.