Iterative methods for solving linear systems on supercomputers

O. Tissot

PhD advisor

L. Grigori

INRIA-ALPINES

15 December 2016

Outline

What is a supercomputer?

Solving linear systems

Enlarged Conjugate Gradient

Outline

What is a supercomputer?

Solving linear systems

Enlarged Conjugate Gradient

Simple description

- What does it look like?

(a) Taihulight (Wuxi, China)

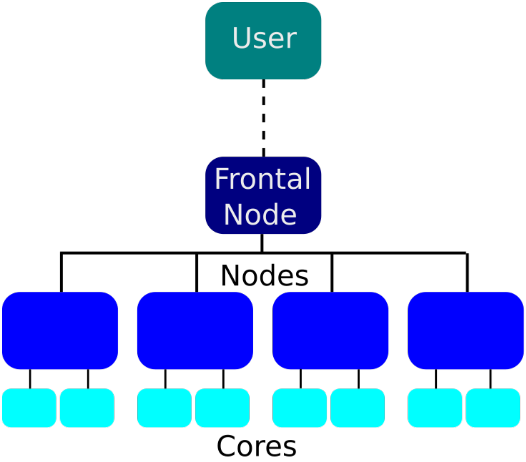
(b) Pangea (Pau, France)

→ A room full of closets with fancy lights...

- What is inside the closets?

→ Many, many, many processors like the one you have in your laptop, linked through a super fast network.

Architecture



How to use a supercomputer?

1. Write a parallel program using Message Passing Interface (MPI)
2. Ask for an access to the machine
3. Connect to the frontal node
   - `ssh username@fancyclustername`
4. Compile your program
   - `mpicc -o superprogram superprogram.c`
5. Submit a job (never run it directly on the frontal node!!!)
   - `sh submitjob.sh`
6. Wait until your job is launched
   - ~~vlc got-s06-e10.avi~~ `emacs juniorseminarslides.tex`
7. Wait until your job is finished
   - ~~vlc twd-s07-e08.avi~~ `open complicatedarticle.pdf`
8. Get back the results on your local machine
   - `scp username@fancyclustername:~/myresults.txt .`
9. ... or go to 4 if your job crashed, failed, ...

Toward exascale

Rank	Country	Cores	R_{max} (TFlop/s)	R_{peak} (TFlop/s)	Power (kW)
1	China	10,649,600	93,014.6	125,435.9	15,371
2	China	3,120,000	33,862.7	54,902.4	17,808
3	US	560,640	17,590.0	27,112.5	8,209
4	US	1,572,864	17,173.2	20,132.7	7,890
5	US	622,336	14,014.7	27,880.7	3,939
16	France	220,800	5,283.1	6,712.3	4,150

Table: Top 5 supercomputers in the world in November 2016[1].

- Objective: an exaflop/s (10^{18} operations per second) machine around 2020!

$\Longrightarrow$ more and more cores

$\Longrightarrow$ less and less power by core

[1] https://www.top500.org/

The communication wall

- Time to move data >> time per floating-point operation (flop)
  - Gap steadily and exponentially growing over time[2]

	Petascale (2009)	Predicted exascale	Factor improvement
System Peak	2.10^{15} flops	10^{18} flops	~ 1000
Memory Bandwith	25 GB/s	0.4-4 TB/s	$\sim$ 10-100
Interconnect Bandwith	3.5 GB/s	100-400 GB/s	~ 100
Memory Latency	100 ns	50 ns	~ 1
Interconnect Latency	$1\mu s$	$0.5\mu s$	~ 1

- Communication-avoiding (CA) algorithms
  - Minimize communications instead of flop
  - Most is known for dense linear algebra (CA-LU, CA-QR)
  - A lot of open problems in sparse linear algebra

[2] Table taken from slides of E. Carson (datas from P. Beckman (ANL), J. Shalf (LBL) and D. Unat (LBL))

Outline

Problem

Find $x_* \in \mathbb{R}^n$ such that $Ax_* = b$ where $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$.

- Arise in a lot of numerical problems
  - Discretizations of differential equations
  - Optimization problems
  - ...
- Simple from a mathematic point of view
  - Chinese know Gaussian elimination since at least the 1^{st} century
- Very time consuming in simulations
- A lot of work on it since the 50*s*
  - Krylov methods: CG (1952), GMRES (1986), BiCG-STAB (1992)
  - Preconditioners (1968)
  - Multigrid (1964)

Overview of the methods

- Direct methods based on Gaussian elimination
  - Stable
  - High complexity
  - High memory cost

$$A = \underbrace{\begin{pmatrix} 1 & & \\ \times & 1 & \\ \times & \times & 1 \end{pmatrix}}_{\mathsf{L}} * \underbrace{\begin{pmatrix} \times & \times & \times \\ & \times & \times \\ & & \times \end{pmatrix}}_{\mathsf{U}}$$

- Iterative methods based on successive projections
  - Unstabilities
  - Low complexity
  - Low memory cost

1. Find $\tilde{x} \in \mathcal{K}$ such that $b - A\tilde{x} \perp \mathcal{L}$ (Petrov-Galerkin condition)
2. Increase $\mathcal{K}$ and $\mathcal{L}$ and go to 1

Krylov methods

- $\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, ..., A^{k-1} r_0\}$ is called Krylov subspace
- Find $x_k \in x_0 + \mathcal{K}_k(A, r_0)$ such that $b - A\,x_k \perp \mathcal{L}_k$
- $b - A\,x_n \perp \mathbb{R}^n \implies x_n = x_*$
- Why searching the solution in $\mathcal{K}_k$?
  - Cayley-Hamilton theorem
  - Cheap to construct: a sequence of SpMVs (Sparse Matrix-Vector product)
- How to choose $\mathcal{L}_k$?
  - Its dimension has to be the same as the one of $\mathcal{K}_k$
  - It has to be cheap to construct

Conjugate Gradient

- Krylov method
  - A is symmetric positive definite
  - $||v||_A = \sqrt{v^\top A\, v}$ is a norm
  - $\mathcal{L}_k = \mathcal{K}_k$
- $||x_* - x_k||_A = \min_{y \in \mathcal{K}_k} ||x_* - y||_A$
- $\forall k \neq i,\ p_k^\top A\, p_i = 0$
- $||x_* - x_k||_A \leq ||x_* - x_0||_A \left(\frac{\sqrt{\kappa - 1}}{\sqrt{\kappa + 1}} \right)^n$
  - Condition number $\kappa = \lambda_{\max} / \lambda_{\min}$
  - Preconditioning to reduce κ

Classic CG

$r_0 = b - Ax_0$

$p_1 = \frac{r_0}{\sqrt{r_0^\top A r_0}}$

while $||r_{k-1}||_2 > \varepsilon ||b||_2$ **do**

$\quad \alpha_k = p_k^\top r_{k-1}$

$\quad x_k = x_{k-1} + p_k \alpha_k$

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

$\quad p_{k+1} = r_k - p_k (p_k^\top A r_k)$

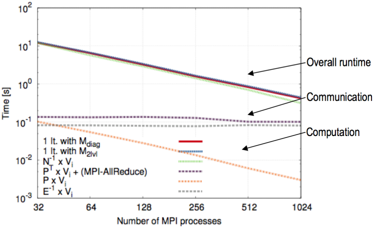
$\quad p_{k+1} = \frac{p_{k+1}}{\sqrt{p_{k+1}^\top A p_{k+1}}}$

end while

Performance bottleneck

Each iteration requires

- AXPY ($y \leftarrow \alpha x + y$)
  - No communication
  - BLAS 1
- SpMV ($y \leftarrow \alpha A x + \beta y$)
  - Point-to-point communication
  - BLAS 2
- Dot products ($\alpha \leftarrow x^\top y$)
  - Global communication
  - BLAS 1



Map making, with R. Stompor, M. Szydlarski
Results obtained on Hopper, Cray XE6, NERSC

$\Rightarrow$ less than 10% of the peak performance on supercomputers[3]!

[3] http://www.hpcg-benchmark.org/

Outline

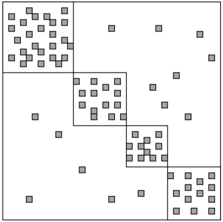
Increase performances of CG

- Overlap communications by flops
  - Pipelined algorithms
  - Limited in practice
- Construct several vector of the basis at once
  - s-step methods
  - Unstable
  - Difficult to use an efficient preconditioner

$\rightarrow$ Search the solution in a bigger space than Krylov subspace
  - Block Krylov methods
  - Derive new algoritms

Enlarged Krylov methods [Grigori et al., 2014]

- Partition the matrix into t domains
- Split the residual r_k into t vectors corresponding to the t domains



$$r_0 \rightarrow T(r_0) = \begin{bmatrix} * & 0 & & 0 \\ \vdots & \vdots & & \vdots \\ * & 0 & & 0 \\ 0 & * & & 0 \\ \vdots & \vdots & & \vdots \\ 0 & * & & 0 \\ & & \ddots & \\ 0 & 0 & & * \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & * \end{bmatrix}$$

- Generate t new basis vectors, obtain an enlarged Krylov subspace

$$\mathcal{K}_{t,k}(A, r_0) = \text{span}^{\square}\{T(r_0), AT(r_0), A^2T(r_0), \ldots, A^{k-1}T(r_0)\}$$

- Search for the solution of the system $Ax = b$ in $\mathcal{K}_{t,k}(A, r_0)$

Properties

- $b - Ax_n \perp \mathbb{R}^n \implies x_n = x_*$
  - $\exists k_{\max}$, such that $\forall q > 0$
    $\mathcal{K}_{t,1}(A, r_0) \subsetneq \cdots \subsetneq \mathcal{K}_{t,k_{\max}-1}(A, r_0) \subsetneq \mathcal{K}_{t,k_{\max}}(A, r_0) = \mathcal{K}_{t,k_{\max}+q}(A, r_0)$
- $\forall k,\ \mathcal{K}_k(A, r_0) \subset \mathcal{K}_{t,k}(A, r_0)$
- Construction: a sequence of SpMMs (Sparse Matrix-Matrix product)
- Find an approximation of $AX_* = T(b)$ where $X_* \in \mathbb{R}^{n \times t}$

  → A special case of block Krylov methods
  - Vectors → Matrices ($n \times t$)
  - Scalars → Matrices ($t \times t$)
- $x_k = \sum_{i=1}^{t} X_k^{(i)}$ (idem for r_k)

Enlarged Conjugate Gradient

- $||x_* - x_k||_A = \min_{y \in \mathcal{K}_{t,k}} ||x_* - y||_A$
- Conjugacy of P_k
  - $\forall k \neq i,\ P_k^\top A\, P_i = 0$
  - $P_i^\top A P_i = \mathbb{I}_{t \times t}$
- Breakdowns ?
  - No, for this algorithm
  - Yes, for other variants...
- $||x_* - x_k||_A \leq C \left(\frac{\sqrt{\kappa_t - 1}}{\sqrt{\kappa_t + 1}} \right)^n$
  - $\boxed{\kappa_t = \lambda_{\max} / \lambda_t}$
  - C is more complicated

EK-CG

```
R_0 = T(b - A x_0)
P_1 = A-orthonormalize(R_0)
while || sum_{i=1}^t R_k^(i) ||_2 < ε||b||_2 do
    α_k = P_k^t R_{k-1}                                  ▷ t × t
    X_k = X_{k-1} + P_k α_k                              ▷ n × t
    R_k = R_{k-1} - A P_k α_k                            ▷ n × t
    P_{k+1} = A P_k - P_k (P_k^t A A P_k) -
              P_{k-1} (P_{k-1}^t A A P_k)                ▷ n × t
    P_{k+1} = A-orthonormalize(P_{k+1})
end while
x = sum_{i=1}^t X_k^(i)                                  ▷ n × 1
```

Classical *v.s.* Enlarged

Classic CG

1: $r_0 = b - Ax_0$
2: $p_1 = \frac{r_0}{\sqrt{r_0^t A r_0}}$
3: **while** $||r_{k-1}||_2 > \varepsilon ||b||_2$ **do**
4: $\quad \alpha_k = p_k^t r_{k-1}$
5: $\quad x_k = x_{k-1} + p_k \alpha_k$
6: $\quad r_k = r_{k-1} - A p_k \alpha_k$
7: $\quad p_{k+1} = r_k - p_k (p_k^t A r_k)$
8: $\quad p_{k+1} = \frac{p_{k+1}}{\sqrt{p_{k+1}^t A p_{k+1}}}$
9: **end while**

EK-CG

1: $R_0 = T(b - Ax_0)$
2: $P_1 = \text{A-orthonormalize}(R_0)$
3: **while** $|| \sum_{i=1}^t R_k^{(i)} ||_2 < \varepsilon ||b||_2$ **do**
4: $\quad \alpha_k = P_k^t R_{k-1}$ $\quad \triangleright\ t \times t$
5: $\quad X_k = X_{k-1} + P_k \alpha_k$ $\quad \triangleright\ n \times t$
6: $\quad R_k = R_{k-1} - A P_k \alpha_k$ $\quad \triangleright\ n \times t$
7: $\quad P_{k+1} = A P_k - P_k (P_k^t A A P_k) - P_{k-1} (P_{k-1}^t A A P_k)$ $\quad \triangleright\ n \times t$
8: $\quad P_{k+1} = \text{A-orthonormalize}(P_{k+1})$
9: **end while**
10: $x = \sum_{i=1}^t X_k^{(i)}$ $\quad \triangleright\ n \times 1$

BLAS 1&2 operations

\# messages per iteration

O(1) from SpMV +

O(log P) from dot prod + norm

BLAS 3 operations

\# messages per iteration

O(1) from SpMM +

O(log P) from BCGS + A-ortho

Parallel Implementation[3]

- Written in C/MPI
- Dependencies
  - MKL (sequential linear algebra)
  - Metis (graph partitioning)
  - **C Pa**rallel **L**inear **A**lgebra **Me**mory **M**anagement (parallel block Krylov building blocks, and more...)
- Several variants
  - EK-CG, BRRHS-CG, Coop-CG
  - Orthodir, Orthomin
- Matrix-free
- Interface for preconditioners
  - Only block diagonal for the moment

[3] Started during Cemracs 2016 summer school

Numerical Results

Method	Nb cores	Iter	Time (s)
EK-CG			
$t = 4$	16	875	17.7
	32	865	9.3
	64	1084	7.9
$t = 8$	16	304	8.9
	32	433	6.4
	64	480	4.8
$t = 16$	16	280	13.9
	32	255	6.2
	64	370	5.9
Petsc	16	1807	191.5
	32	2755	66.6
	64	4227	48.5

- Run on hpc2
  - LJLL's cluster
  - 320 cores
- Elasticity matrix
  - $n = 145\,563$
  - $nnz = 4\,907\,997$
- $\varepsilon = 10^{-5}$
- Block diagonal preconditioner

Conclusion and perspectives

- On today's supercomputers **communication** is the bottleneck for obtaining good performances
- For solving sparse linear systems on idea is to use **Enlarged Krylov** methods
  - Special kind of block Krylov methods
- First results of the parallel implementation are very promising
- Ongoing work
  - Test on large scale machines (around 10 000 cores)
  - Reduce dynamically the number of search directions
  - Generalize to non-symmetric case (Enlarged BiCG-STAB)

Thank you for your attention!

Questions?

References (1)

Grigori, L., Moufawad, S., and Nataf, F. (2014).
Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication.
Technical Report 8597, INRIA.