Cours MPRI 2-12-2

Lecture 3/5: Sparse linear algebra

(lecturer for part 2/3): E. Thomé

Nov. 19th, 2012

Plan

Sparse linear algebra

Let M be an $N \times N$ matrix over a finite field K. We want to find:

$$w \in K^N \text{ s.t. } Mw = 0.$$

- Factoring or DL: M is sparse: $O(\log^2 N)$ non-zeroes per row.
- Factoring: $K = \mathbb{F}_2$; DL over $\mathbb{F}_q$: $K = \mathbb{F}_\ell$ with $\ell \mid (q-1)$.
- Space complexity for storing M: $O(N \log^2 N)$.

Linear system solving:

- Gauss: time $O(N^3)$, space $O(N^2)$.
- Recursive, using matrix multiply: time $O(N^\omega)$, space $O(N^2)$:
  - Strassen $w = \log_2 7 = 2.81$,
  - Coppersmith-Winograd $w = 2.38$.
  - Conjecturally $w = 2$, but the algorithm is yet to be stated. (Cohn-Kleinberg-Szegedy-Umans, 2005).
- None of the options above exploit sparsity.

Sparse linear algebra

For matrices arising from crypto contexts, fill-in cannot be tolerated.

Some figures from RSA-768

- 192 796 550 rows/columns ;
- 27 797 115 920 non-zero coefficients.
- 105 gigabytes as a sparse matrix.
- $>$4000 terabytes as a dense bit matrix.

The matrix cannot be modified in the course of the computation. We may only use black-box algorithms. No access to M itself.

$$v \longrightarrow \boxed{M} \longrightarrow M \times v$$

Can we do something with black boxes ?

An example in numerical analysis.

- Take a random vector v.
- Iterate $v \leftarrow Mv/\|Mv\|$.
- If M has a dominant eigenvalue λ, $\frac{\|Mv\|}{\|v\|} \to |\lambda|$.

If we can do such things, no doubt we can do more.

We present two important black-box algorithms:

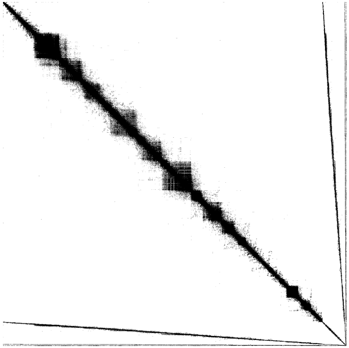
- Lanczos ;
- Wiedemann.

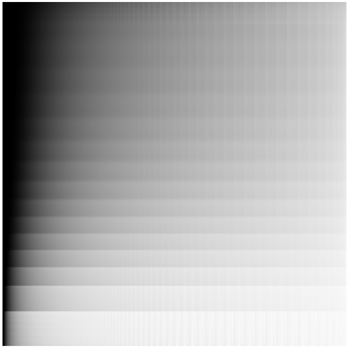
Comparison with numerical world

Exact linear algebra differs much from linear algebra over $\mathbb{C}$.

- No notion of approximate solution.
- No notion of convergence.

The matrices are not the same either:


(some PDE example)


(a factoring matrix)

Krylov subspace

black-box algorithms are good at computing $M^k v$.

Krylov subspace $\mathcal{K}_{M,v}$

We define $\mathcal{K}_{M,v} = \left\langle v, Mv, \ldots, M^k v, \ldots \right\rangle$.

- $\mathcal{K}_{M,v}$ is a subspace of K^N.
- Computing generating vectors is easily done with a loop.
    - output v;
    - $v \leftarrow Mv$;
    - repeat.

- Lanczos: mimick Gram-Schmidt process on $\mathcal{K}_{M,v}$.
- Wiedemann: try to find the min. poly. of M on $\mathcal{K}_{M,v}$.

Plan

Lanczos

Here $K = \mathbb{F}_q$, with q large: “almost characteristic zero”.

- Will try to keep close to Gram-Schmidt orthogonalization.
- Build a symmetric matrix: let $A = {}^tMM$
  (A is never computed, but known as a black box !)
- (pseudo-) scalar product associated to A: $(u, v)_A \overset{\text{def}}{=} {}^tuAv$.
- Note over a finite field, there are isotropic vectors (exercise: find one !)

Gram-Schmidt orthogonalization process:

- build an orthogonal basis from an arbitrary one.
- defined in characteristic zero for a real scalar product, but let’s see.

GSO in positive characteristic

We take the method for its merits.

- It builds a sequence of vectors with $(e_i, e_j)_A = 0$ if $i \neq j$.
- We believe for a moment that nothing fails.
- We'll see what might fail and why.

Apply GSO to the basis $\left(A^i b\right)_i$ of $\mathcal{K}_{A,b}$.

$$e_0 \leftarrow b,$$

$$e_{j+1} \leftarrow A^{j+1}b - \sum_{i \leq j} \frac{(A^{j+1}b, e_i)_A}{(e_i, e_i)_A} e_i = A^{j+1}b - \sum_{i \leq j} \frac{{}^t b A^{j+2} e_i}{{}^t e_i A e_i} e_i.$$

Two key facts

- $(e_i, e_j)_A = 0$ if $i \neq j$.
- $\langle e_0, \ldots, e_j \rangle = S_j = \langle b, \ldots, A^j b \rangle$.

GSO on Krylov subspaces

$$e_0 \leftarrow b,$$
$$e_{j+1} \leftarrow A^{j+1}b - \sum_{i \le j} \frac{(A^{j+1}b, e_i)_A}{(e_i, e_i)_A} e_i = A^{j+1}b - \sum_{i \le j} \frac{{}^t bA^{j+2}e_i}{{}^t e_i A e_i} e_i.$$

Two key facts

- $(e_i, e_j)_A = 0$ if $i \neq j$.
- $\langle e_0, \ldots, e_j \rangle = S_j = \langle b, \ldots, A^j b \rangle$.

Important: we may replace $A^{j+1}b$ by Ae_j.

- Explanation: $A^{j+1}b = Ae_j +$ something in AS_{j-1}.
- $AS_{j-1} \subset S_j$, so contribution will be canceled.

Lanczos (cont'd)

$$e_{j+1} \leftarrow A^{j+1}b - \sum_{i\leq j} \frac{(Ae_j, e_i)_A}{(e_i, e_i)_A} e_i = Ae_j - \sum_{i \leq j} \frac{{}^t e_j A^2 e_i}{{}^t e_i A e_i} e_i,$$

Getting rid of many terms

For $i \leq j-2$, we have:

$$Ae_i \in S_{j-1} \subset e_j^{\perp} \Rightarrow (e_j, Ae_i)_A = (Ae_j, e_i)_A = 0.$$

We may restrict to $i \in \{j-1, j\}$.

$$\begin{aligned} e_{j+1} &\leftarrow Ae_j - \frac{(Ae_j, e_j)_A}{(e_j, e_j)_A} e_j - \frac{(Ae_j, e_{j-1})_A}{(e_{j-1}, e_{j-1})_A} e_{j-1}, \\ &\leftarrow Ae_j - \frac{{}^t e_j A^2 e_j}{{}^t e_j A e_j} e_j - \frac{{}^t e_j A^2 e_{j-1}}{{}^t e_{j-1} A e_{j-1}} e_{j-1} \end{aligned}$$

Lanczos over $\mathbb{F}_p$: failure cases

Algorithm. compute the sequence e_j, maintaining $O(1)$ vectors.

Two possible reasons for stopping:

- We may reach an isotropic (a.k.a. self-orthogonal) vector: $(e_i, e_i)_A = 0$.
  - We have $(e_i, e_i)_A = {}^t e_i A e_i = {}^t(Me_i)Me_i = 0$.
  - Me_i might be isotropic for the "standard" bilinear form, but heuristically Prob $\approx \frac{1}{q}$ only.
- Eventually, we reach $e_i = 0$ at the end. This means success.
  - This implies that $\langle e_0, \ldots, e_{i-1}\rangle = \langle b, Ae_0, \ldots, Ae_{i-1}\rangle$.
  - Let z be a solution to $Az = b$ (z is not known). Let $w = \sum_{j<i} \frac{(e_j, z)}{(e_j, e_j)} e_j = \sum_{j<i} \frac{{}^t e_j b}{{}^t e_j A e_j} e_j$.
  - By construction, $\forall j,\ (e_j, w - z) = 0$.
    Thus $w - z \in \operatorname{Ker} M$ (and $Aw = b$) with proba $\approx \frac{1}{q}$.
  - If we started with $b = Az$ (z known), this gives $w - z \in \operatorname{Ker} M$.

Lanczos: remarks

Note: As is, the Lanczos algorithm does not work over $\mathbb{F}_2$ because for $q = 2$, a failure probability of $\frac{1}{q}$ at each step is a lot.

Complexity:
- N products $A \cdot v$,
- hence $2N$ products M (or ${}^tM$) times v.

Important (mis-)features:

- Needs fast operations for ${}^tM$ and M.
  Often implies storing twice for best efficiency.
- Must keep track of several vectors.

Plan

Wiedemann

The Wiedemann algorithm is a different Krylov method.

- It does not originate from numerical analysis ;
- it does not require a symmetric matrix.

Suppose that we know the minimal polynomial μ_M of M.

- Write $\mu_M = X^\lambda \nu_M$ for some $k \geq 1$ (M assumed singular).
- Let z be a random vector.
- Compute $w = \nu_M(M)z$. We have $\mathrm{Prob}(w = 0) \approx \frac{1}{q}$.
- For some $i \in [\![1 \ldots k]\!]$, we have $M^i w = 0$ and $M^{i-1} w \neq 0$.

Problem: compute μ_M.
Working with $I, M, M^2, \ldots$ is prohibitively expensive. Avoid !

Wiedemann

Let x, y be arbitrary random vectors in K^N. Let:

$$a_i = {}^t x M^i y \in K.$$

Properties of the sequence $(a_i)_i$

- $(a_i)_i$ is linearly recurrent ;
- its generator divides $\hat{\mu}_M$.

Equivalently, the power series $A(T) = \sum a_i T^i$ is a rational fraction. Its denominator divides $\hat{\mu}_M$. In other words, $\hat{\mu}_M(T)A(T) \in K[T]$.

Most important:
- $O(N)$ terms suffice to compute the generator ;
- The generator does not differ much from $\hat{\mu}_M$.

Rational reconstruction

Finding f, g such that $A(T) = \frac{f(T)}{g(T)}$ is called rational reconstruction.

Thm. $2N$ terms are enough.

Proof: We know a rational form $A = f_0/g_0$ exists with $\deg g_0 \leq N,\ \deg f_0 < N$. Assume we obtain $Ag_1 = f_1 + O(X^{2N})$, with same degree bounds. Then $f_0g_1 - f_1g_0 \in O(X^{2N})$ implies equality.

Two ways to go:

- (truncated) EEA with inputs X^{2N} and $A \mod X^{2N}$.
- Berlekamp-Massey algorithm (from coding theory).

Truncated EEA for rational reconstruction

The steps of the extended Euclidean algorithm yield:

$$X^{2N} u_i + (A \mod X^{2N}) v_i = r_i.$$

Prop. $\deg v_i + \deg r_{i-1} = 2N$.

$$\text{Thus } \exists i_0, \quad \deg v_{i_0} \leq N < \deg v_{i_0+1},$$
$$\text{which implies:} \qquad \deg r_{i_0} < N.$$

Complexity:

- $O(N^2)$,
- or $O(\mathsf{M}(N) \log N) = O(N \log^2 N)$ with asymptotically fast algorithms.

Berlekamp-Massey algorithm

Over a field, essentially the same algorithm.

The algorithm works with candidate generators ϕ_0 and ϕ_1.

At step i, we have $\deg(\phi_k \cdot A) = (\deg < \deg \phi_k) + O(X^i)$.

- Use the lowest-degree candidate ϕ_k to cancel $[X^i]\phi_{1-k}A$: Do $\phi_{1-k} \leftarrow \phi_{1-k} - \lambda\phi_k$.
- Do $\phi_k \leftarrow X\phi_k$.

On average, $\deg \phi_i$ advances by only $\frac{1}{2}$. Output: a generator of degree N. We use $2N$ coefficients of A.

Complexity:

- $O(N^2)$,
- or $O(\mathsf{M}(N)\log N) = O(N\log^2 N)$ with asymptotically fast algorithms.

Wiedemann: end

Last step of the Wiedemann algorithm: compute $\nu_M(M)y$.

Total complexity:

- $2N$ products M times v for computing $2N$ terms of A.
- $O(N^2)$ or $O(N \log^2 N)$ for EEA/BM.
- N products M times v for the evaluation.

Failure probability:

- Can be computed exactly depending on the invariant factors of M.
  (see Wiedemann (1986), Kaltofen-Eberly-Villard (many), T. (2003)).
- Bottom line: Prob(failure) $= O(\frac{1}{q})$.

Plan

Block algorithms

- Neither plain Wiedemann nor plain Lanczos work over $\mathbb{F}_2$.
- Furthermore, working with bit vectors is wasteful.

Idea: Replace K by a vector space K^n, for e.g. $n = 64$.

- Goal 1: bring probability of failure from $\frac{1}{2}$ to 2^{-64}.
- Goal 2: achieve better computational efficiency.

Bit arithmetic with unsigned longs

Assume we take the `unsigned long` type to hold bits.

- 0 for bit 0
- 1 for bit 1

Then we have:

- Addition: `x ^ y`.
- Multiplication: `x & y`.
- Multiplication by non-zero: `x`.

In the context of black box linear algebra, we may:

- Add bits.
- Multiply bits by non-zero coefficients of the matrix.

Block algorithms: do this, but pack 64 bits in an unsigned long.

Same cost for main computation: `xor`.

Switching to block black box linear algebra

The black box operation becomes:

matrix $\times$ block of vectors $\rightarrow$ block of vectors.

- We make better use of the `unsigned long` type.
- The proper block width is prescribed by the hardware.
  - One may e.g. use SSE-2 types and instructions, block width 128.
  - Narrower block sizes may also be considered.
- Note: Wider block sizes means larger vectors ! The wider is not always the better.

Big question. Which algorithms can take advantage of this ?

Block algorithms

- Block Lanczos is a construction of orthogonal sub-spaces.
    - At each step, we must ensure that some rank does not drop.
    - Complexity: $2N/(n-0.76)$ matrix-times-vector products.
    - Collective operations at each step.
- Block Wiedemann computes $A \in K^{n\times n}[[X]]$.
    - Only $\frac{2N}{n} + O(1)$ terms of A need be computed.
    - Evaluation: $\frac{N}{n}$ products.
    - EEA does not work. BM works, but somewhat harder.
    - Can be distributed across k sites if $n = 64k$.

Plan

(harder)

Block algorithms

- Block Lanczos (Montgomery)
- Block Wiedemann (Coppersmith)

Block Lanczos algorithm

(harder)

- BL: one of the rare algorithms I know which is uglier than BW.
- Presenting plain Lanczos correctly not always easy, so BL...

Let n be a block width (e.g. $n = 64$).

Starting point of BL:
- Start with an $N \times N$ matrix M.
- Want to solve $Mv = 0$ (of $Mv = b$).
- Let $A = M^T M$.

Definition: orthogonal subspaces

Let W and W' be two $N \times n$ matrices defining two n-dimensional subspace $\mathcal{W}$ and $\mathcal{W}'$ of $\mathbb{F}_2^N$.
$\mathcal{W}$ and $\mathcal{W}'$ are A-orthogonal ($\mathcal{W} \perp_A \mathcal{W}'$) if $W^T A W' = 0$.

Principle of BL

(harder)

Let V_0 be a random initial $N \times n$ matrix.
V_0 defines a subspace $\mathcal{V}_0$, but $\mathcal{V}_0$ is not our focus.

First goal: build an interesting subspace $\mathcal{W}_0 \subset \mathcal{V}_0$

We want a matrix W_0 such that $W_0^T A W_0$ has full rank.

- Let $n_0 = \operatorname{rank}\left(V_0^T A V_0\right)$.
- Let W_0 be an $N \times n_0$ matrix such that $\operatorname{rank}\left(W_0^T A W_0\right) = n_0$.
  (this is easy: extract n_0 linearly indep. cols of $V_0^T A V_0$).
- Let $\mathcal{W}_0 = \langle W_0 \rangle$ be the spanned subspace.

Next step

(harder)

We want to grow $\mathcal{W}_0$ into a sequence of subspaces $\mathcal{W}_i$ which:

- are related to eachother.
- are mutually A-orthogonal.
- have dimension most often equal to n.

Starting point: AW_0 defines a new n_0-dimensional subspace "$A\mathcal{W}_0$".

Let V_1 be naively AW_0. We may build W_1 such that:

- $\mathcal{W}_1 \perp_A \mathcal{W}_0$:

$$W_1 \leftarrow V_1 - W_0 \left(W_0^T A W_0\right)^{-1} W_0^T A V_1.$$

- $\langle W_1 \rangle \subset A\mathcal{W}_0$, and $W_1^T A W_1$ full rank (same as for W_0).

and so on and so forth.

Problem with BL

(harder)

- The procedure we have given does build a nice sequence of spaces, until it collapses.
- $\mathrm{rank}(W_i)$ decreases slowly to 0.

$$\begin{tikzcd}[row sep=tiny] & \mathcal{W}_0\text{, dimension } n_0 \leq n \\ V_0 \arrow[ur] \arrow[dr] & \\ & n - n_0 \text{ vectors dropped} \end{tikzcd}$$

$$\begin{tikzcd}[row sep=tiny] & \mathcal{W}_1\text{, dimension } n_1 \leq n_0 \\ V_1 = AW_0 \arrow[ur] \arrow[dr] & \\ & n_0 - n_1 \text{ vectors dropped} \end{tikzcd}$$

$$\begin{tikzcd}[row sep=tiny] & \mathcal{W}_2\text{, dimension } n_2 \leq n_1 \\ V_2 = AW_1 \arrow[ur] \arrow[dr] & \\ & n_1 - n_2 \text{ vectors dropped} \end{tikzcd}$$

Problem with BL

(harder)

- The procedure we have given does build a nice sequence of spaces, until it collapses.
- rank(W_i) decreases slowly to 0.

$$\begin{tikzcd} & \mathcal{W}_0\text{, dimension } n_0 \le n \\ V_0 \arrow[ur] \arrow[dr] & \\ & n - n_0 \text{ vectors dropped} \end{tikzcd}$$

$$\begin{tikzcd} & \mathcal{W}_1\text{, dimension } n_1 \le n_0 \\ V_1 = AW_0 \arrow[ur] \arrow[dr] & \\ & n_0 - n_1 \text{ vectors dropped} \end{tikzcd}$$

$$\begin{tikzcd} & \mathcal{W}_2\text{, dimension } n_2 \le n_1 \\ V_2 = AW_1 \arrow[ur] \arrow[dr] & \\ & n_1 - n_2 \text{ vectors dropped} \end{tikzcd}$$

Key difference between BL and Lanczos

(harder)

Main difference

In BL, in order to prevent the dimension collapse, we reinject in V_1 the vectors of V_0 which have been discarded when building W_0.

- $\langle V_1 \rangle$ is thus an n-dimensional subspace, like $\langle V_0 \rangle$.
- The subspace $\mathcal{W}_1$ extracted thus has dimension $n_1 \leq n$.

The rest is all ugly technicalities.

- How exactly reinjecting is formulated.
- How we orthogonalize V_k w.r.t. previous subspaces.
- How we shorten the sequence of orthogonalizing computations.

BL and bookkeeping

(harder)

- The BL iterations needs to keep several vector blocks.
  - V_{i+1}, V_i, W_{i-1}, W_{i-2}, W_{i-3}.
  - Even an extra vector block if solving inhomogeneous system.
- Each iteration shuffles the vector columns: W_{i+1} is an extraction from V_{i+1}.
- Both multiplications by M and M^T (since $A = M^T M$).
- Scalar products, multiplication by $n \times n$ matrices, ...

Homogeneous BL (T. 2003 ??)

A natural idea.

- The sequence eventually reaches the point where $W_i^T A W_i = 0$, from which we can extract vectors of Ker M.
- This saves one vector block for bookkeeping, and some scalar products.

Number of iterations of BL

(harder)

The dimension of W_i is the rank of $V_i^T A V_i$.

Theorem

Let $n - \text{rank}(X)$ be the rank defect of an $n \times n$ matrix M.

$$\mathbb{E}\left(\text{rank defect}(M),\ M \text{ symmetric}\right) = 0.76,$$
$$\mathbb{E}\left(\text{rank defect}(M,\ M \text{ general})\right) = 0.85.$$

Thus BL runs until $\langle \mathcal{W}_0, \mathcal{W}_1, \dots \mathcal{W}_k \rangle = \mathbb{F}_2^N$, which means:

$$k \approx \frac{N}{n - 0.76}.$$

Plan

(harder)

Block algorithms

- Block Lanczos (Montgomery)
- Block Wiedemann (Coppersmith)

Block Wiedemann

(harder)

BW is a direct translation of Wiedemann to using vector blocks.

Issues:

- properly define the notion of linear generator.
- show that using vector blocks reduces the number of needed iterations.

The expected benefits are clear:

- Better use of arithmetic power of CPUs (block operations).
- Hopefully better success probability.

BW workplan

(harder)

Let n be a block width.

- Initial setup. Choose starting blocks of vectors x and y.
- Sequence computation. Want L first terms of the sequence:

$$a_i = x^T M^k y \text{ (}a_i\text{ are } n \times n \text{ matrices !).}$$

  - Computing one term after another, this boils down to our black box $v \mapsto Mv$.
  - This computation can be split into several independent parts (which all know M).
- Compute some sort of minimal polynomial.
- Build solution as:

$$v = \sum_{k=0}^{\deg f} M^k y f_k.$$

  - Again, this uses the black box.
  - Can be split into many independent parts (which all know M).

BW operations

(harder)

For the sequence computation, the only operations are:

- Matrix times vector product $v_i \leftarrow Mv_{i-1}$.
- Dot product $a_i \leftarrow x^T v_i$. (x typically taken very simple).

Required bookkeeping: only v_i and v_{i-1}.

Most important thing

Col. j or the matrices $(a_i)_i$ only depends on col. j of y.

- If y is split into several parts, this leads to several parts of the sequence which may be computed independently.
- Those different parts of the sequence need no synchronization or communication.
- Possibly on different clusters, sites, or countries.
- Block width $n = 64n'$: n' independent computations.

BW complexity

(harder)

For a block size n, BW on an N-dimensional matrix M:

- $\frac{2N}{n}$ matrix times vector products for sequence computation.
- $\frac{N}{n}$ extra matrix times vector products for the last step.
- Linear generator computation: nN^2 naively, asymptotically fast algorithm in $\approx nN(n + \log N)$.

Comparison with BL

On the back of the envelope, BW is slower than BL, but:

- Less bookkeeping,
- Only products by M, not by M^T,
- Much better distribution opportunities.

Plan

What are the algorithms good for ?

- Plain Lanczos. OK for $\mathbb{F}_p$.
  - Quite easy to implement.
  - Has been used for DL computations.
- Plain Wiedemann. OK for $\mathbb{F}_p$.
  - Reconstruction step add some implementation work.
  - On the other hand, recurrence is easier.
- Block Lanczos. Good for $\mathbb{F}_2$.
  - Need a large cluster.
  - Needs fast, parallel $M \times v$.
- Block Wiedemann. Good for $\mathbb{F}_2$.
  - Can accomodate several clusters.
  - Implementation is challenging.

2012: Wiedemann on GPU

DL record $\mathbb{F}_{2^{619}}$. Matrix step relatively easy.

- 635,000 rows and columns.
- About 100 non-zeroes per row.
- Field of definition: a 217-bit prime.

Wiedemann recurrence done on a Nvidia GeForce GTX580 GPU.

- About 30ms for each $M \times v$.
- Fault-tolerant software, because GTX580 is a mess.
- Orders of magnitude faster than CPU implementation.

1999: Block Lanczos for RSA-155

RSA-155: an important milestone for factoring (512 bits).

Matrix step (1999):

- (by then) a large matrix: 6.7M rows/cols, 62 nz/row.
- Solved on supercomputer Cray C916 (10 days).

Stumbling block: relying on a Cray-class supercomputer is cumbersome.

2009: Block Wiedemann for RSA-768

RSA-768: latest record in integer factorization.

Block Wiedemann algorithm used for matrix step.

- Requiring several mid-range computer resources is much more manageable than supercomputers.
- Used grids of computers in France, Switzerland, Japan.
- Approx. 3 months of computation.
- Novel approach, using varying clusters.

Linalg stats

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
Lausanne	56	2×AMD 2427	2.2	12	16	ib20g	12	144	4.3	4.8	40%
Tokyo	110	2×Pentium-D	3.0	2	5	eth1g	110	220	5.8	7.8	%
Grenoble	34	2×Xeon E5420	2.5	8	8	ib20g	24	144	3.7		30%
Lille	46	2×Xeon E5440	2.8	8	8	mx10g	36	144	3.1	3.3	31%
							32	256	3.8		38%
							24	144	4.4		33%
Nancy	92	2×Xeon L5420	2.5	8	16	ib20g	64	256	2.2	2.4	41%
							36	144	3.0	3.2	31%
							24	144	3.5	4.2	30%
							18	144		5.0	31%
							16	64		6.5	19%
Orsay	120	2×AMD 250	2.4	2	2	mx10g	98	196	2.8	3.9	32%
Rennes	96	2×Xeon 5148	2.3	4	4	mx10g	64	256	2.5	2.7	37%
							49	196	2.9	3.5	33%
Rennes	64	2×Xeon L5420	2.5	8	32	eth1g	49	196	6.2		67%
							24	144	8.4		67%
							18	144	10.0		68%
							8	64		18.0	56%

Table 1: Different per-iteration timings on various clusters. (a) Cluster location ; (b) Total cluster size (number of nodes) ; (c) Cluster CPU type ; (d) Node CPU frequency ; (e) Cores per node ; (f) RAM per node (GB) ; (g) Cluster interconnect ; (h) Job size (number of nodes) ; (i) Number of cores used per job ; (j) Time per iteration in seconds (stage 1) ;