Resolution of a large number of small random symmetric linear systems in single precision arithmetic on GPUs

Stef Graillat

LIP6/PEQUAN, Sorbonne Universités, UPMC Univ Paris 06, CNRS

Joint work with Lokmane Abbas-Turki (UPMC - LPMA)

International Conference on Monte Carlo techniques

July 5-8th 2016, Paris, France

Outline of the talk

1. Introduction - motivations
2. Solving small linear systems on GPU
3. Conclusion

Outline of the talk

Motivations for HPC

- HPC in banking institutions
  - Rather distribution than parallelization,
  - Organized around clusters with small nodes,
  - Use the .NET C, C++ and C#.
- Emergence of new solutions
  - The efficiency of GPUs becomes undeniable,
  - Nodes become bigger and bigger,
  - Virtualization and cloud computing.
- Challenges
  - Code management.
- A solution for the Credit Valuation Adjustment (CVA)

Motivations: Credit Valuation Adjustment (CVA)

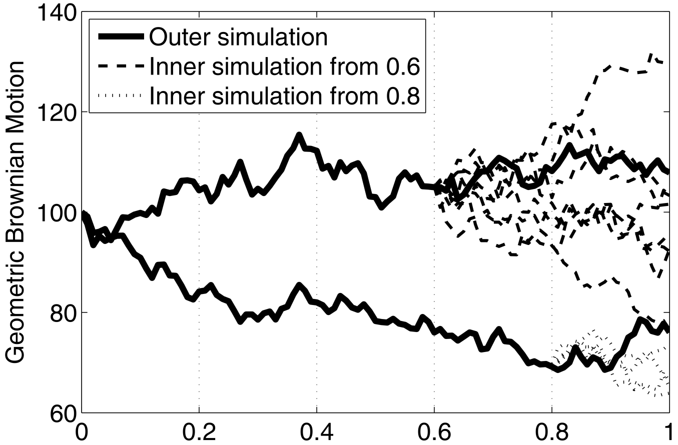
Definition (Credit Valuation Adjustment)

In a financial transaction between a party C that has to pay another party B some amount V, the CVA value is the price of the insurance contract that covers the default of party C to pay the whole sum V.

$$CVA_{t,T} = (1-R)E_t\left(V_\tau^+ \mathbb{1}_{t<\tau\leq T}\right)$$

- *R is the recovery to make if the counterparty defaults (Assume $R = 0$),*
- *τ is the random default time of the counterparty,*
- *T is the protection time horizon.*

Simulation for American options



Standard methods cannot be used directly (1/2)

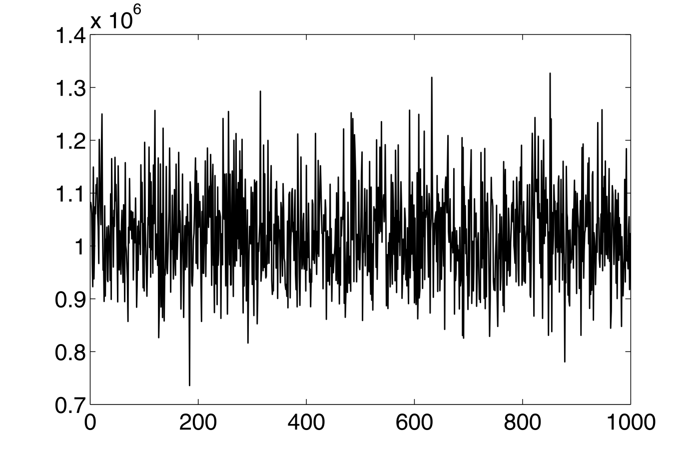
The reason

- Large number of small random linear systems: The size does not exceed 64 and the communication is reduced.
- Some of these random systems could be ill-conditioned.

$$\widehat{A}_{k,l} = \frac{1}{M_k} \sum_{j=1}^{M_k} \psi^l(S_{t_k}^{(j)}) \psi^l(S_{t_k}^{(j)})^t$$

Standard methods cannot be used directly (2/2)

Typical condition numbers for linear regression $n = 30$ in the Black & Scholes model



Outline of the talk

Three main methods for large symmetric matrices

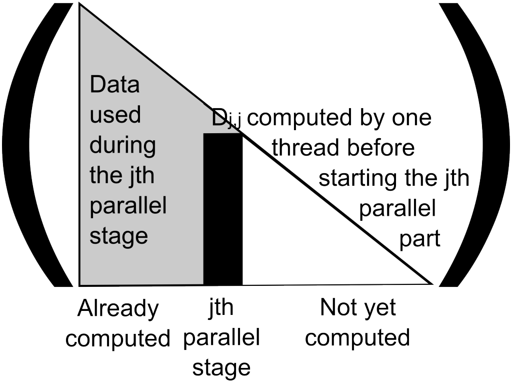
- **Cholesky factorization**
  - V. Volkov and J. Demmel. LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs. Berkeley Technical Report. 2008.
  - G. Ballard, J. Demmel, O. Holtz and O. Schwartz, Communication-Optimal Parallel and Sequential Cholesky Decomposition. SIAM J. SCI. COMPUT. 32(6), 3495–3523. 2010.
- **Tridiagonal form + cyclic reduction**
  - Y. Zhang , J. Cohen and J. D. Owens. 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 127–136. 2010.
  - D. Goddeke and R. Strzodka. Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed Precision Multigrid. Parallel and Distributed Systems, IEEE Trans. 22(1), 22–32. 2010.
- **Tridiagonal form + eigenproblem**
  - C. Vomel, S. Tomov and J. Dongarra. Divide & Conquer on Hybrid Gpu-Accelerated Multicore Systems. SIAM J. SCI. COMPUT. 34(2), 70–82. 2012.

Standard LDLt parallel strategy

Shared occupation $n(n+1)/2+n$ and complexity $O(n^3/6)$

$$A = LDL^t, \quad D_{j,j} = A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2 D_{k,k},$$

$$L_{i,j} = \frac{1}{D_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} D_{k,k} \right) \quad \text{if } i > j.$$

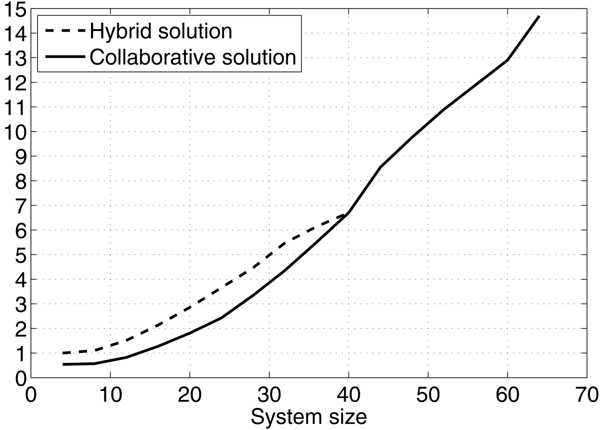


Three different versions (1/2)

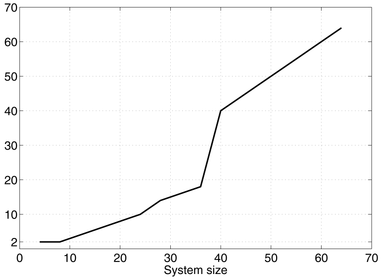
1. An SIMD version that requires only independent threads, one for each linear system.
2. A collaborative version that involves n collaborative threads for each linear system with n unknowns.
3. An optimal hybrid solution that involves n^* ($n^* < n$) collaborative threads for each linear system with n unknowns.

Three different versions (2/2)

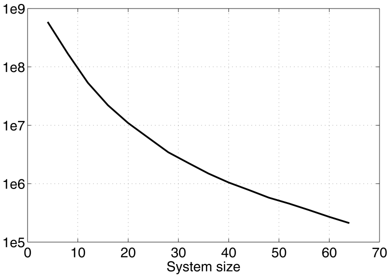
The speedup of the collaborative and the hybrid versions when compared to the SIMD implementation.



Performance results



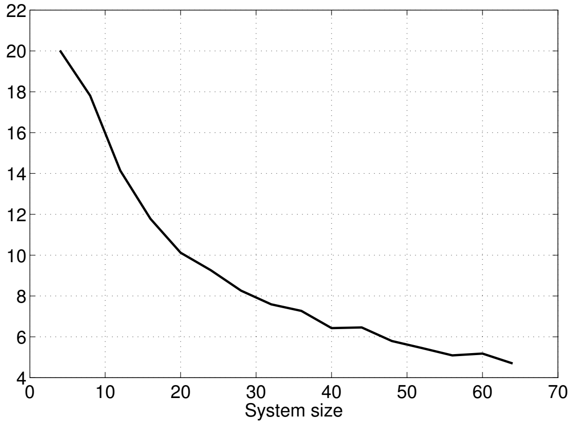
Optimal number of collaborative threads



Number of systems solved per s

Performance results

LDLt resolution: The speedup of CUDA/GPU implementation compared to OpenMP/CPU. This speedup is measured in term of the number of solved systems per second



Householder tridiagonalization + PCR

Householder tridiagonalization: Shared occupation $n^2 + 2n$ and complexity $O(4n^3/3)$

1. An SIMD version that requires only independent threads, one for each linear system.
2. A collaborative version that involves n collaborative threads for each linear system with n unknowns.

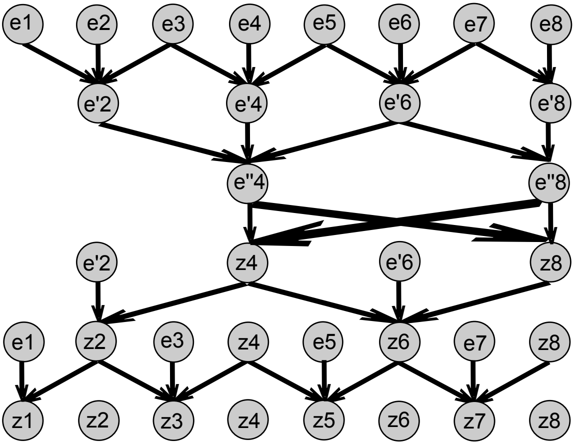
For symmetric A

$$U = H_3^t \dots H_n^t A H_n \dots H_3 = \begin{pmatrix} d_1 & c_1 & & & & \\ c_1 & d_2 & c_2 & & 0 & \\ & c_2 & d_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & c_{n-1} & d_n \end{pmatrix},$$

with each Householder matrix H given by $H = I - uu^t/b$, $b = u^t u/2$.

Cyclic reduction

Shared occupation $3n$ and complexity $O(n \log_2(n))$



Step 1: Forward reduction to a 4-unknown system involving z2, z4, z6 and z8

Step 2: Forward reduction to a 2-unknown system involving z4 and z8

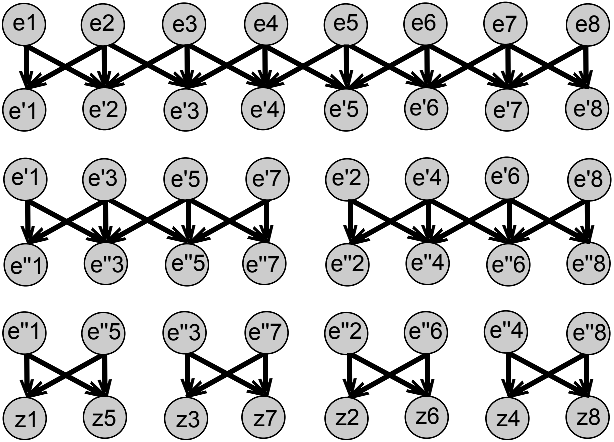
Step 3: Solve 2-unknown system

Step 4: Backward substitution to solve the rest 2 unknowns

Step 5: Backward substitution to solve the rest 4 unknowns

Parallel cyclic reduction

Shared occupation $4n$ and complexity $O(n \log_2(n))$

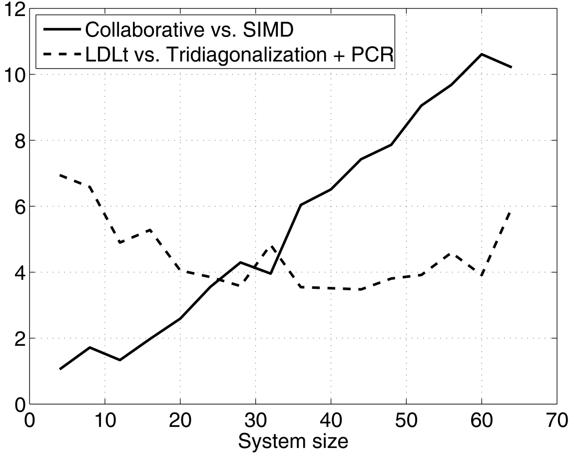


Step 1: Reduced to 2 systems of 4 unknowns

Step 2: Reduced to 4 systems of 2 unknowns

Step 3: Solve

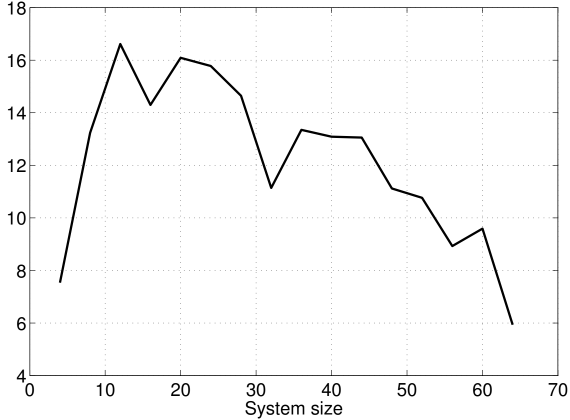
Comparisons



LDLt vs. tridiagonal + PCR

Comparisons

Householder reduction + PCR: The speedup of CUDA/GPU implementation compared to OpenMP/CPU. This speedup is measured in term of the number of solved systems per second.



Divide and conquer for eigenproblem

- Tridiagonal Householder decomposition $A = QUQ^t$ where Q is orthogonal and U is symmetric tridiagonal.
- Divide & conquer algorithm for symmetric tridiagonal eigenproblems to establish $U = ODO^t$ where O is orthogonal and D is diagonal.
- Discard the smallest eigenvalues of D that provide a condition number larger than 10^5.

$$
U = \left(\begin{array}{cccc|cccc}
d_1 & c_1 & & & & & & \\
c_1 & \ddots & \ddots & & & & & \\
 & \ddots & \ddots & c_{m-1} & & & & \\
 & & c_{m-1} & d_m - c_m & 0 & & & \\
\hline
 & & & 0 & d_{m+1} - c_m & c_{m+1} & & \\
 & & & & c_{m+1} & \ddots & \ddots & \\
 & & & & & \ddots & \ddots & c_{n-1} \\
 & & & & & & c_{n-1} & d_n
\end{array}\right) + c_m 1_{m,m+1} 1_{m,m+1}^t
$$

$$
= \left(\begin{array}{c|c} U_1 & 0 \\ \hline 0 & U_2 \end{array}\right) + c_m 1_{m,m+1} 1_{m,m+1}^t
$$

Divide and conquer for eigenproblem (1/2)

Shared occupation $2n(n+2) + 2^{1+\lfloor \log_2(n-1) \rfloor}$ and complexity $O(4n^3/3)$

1. $$U = \begin{pmatrix} O_1 & 0 \\ 0 & O_2 \end{pmatrix} \left(\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + c_m u u^t \right) \begin{pmatrix} O_1^t & 0 \\ 0 & O_2^t \end{pmatrix}$$
   $$\text{where } u = \begin{pmatrix} O_1^t & 0 \\ 0 & O_2^t \end{pmatrix} 1_{m,m+1} = \begin{pmatrix} \text{last column of } O_1^t \\ \text{first column of } O_2^t \end{pmatrix}.$$

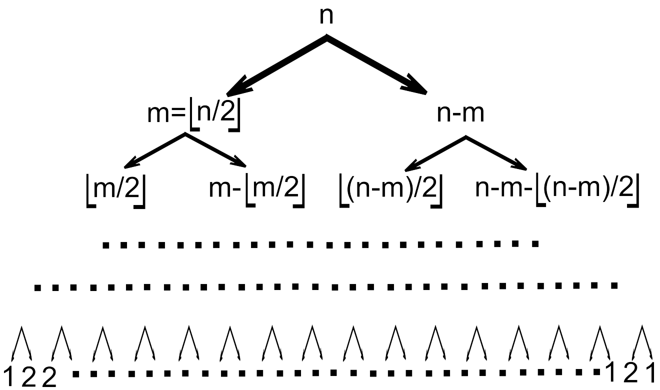
2. Let $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$, ordered family of eigenvalues of $\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}$.
   If $c_m \neq 0$ and the eigenvalue λ of U satisfies $\lambda \notin \Lambda$, then its value is obtained as a solution of the *secular equation*
   $$\sum_{i=1}^{n} \frac{u_i^2}{\lambda_i - \lambda} + \frac{1}{c_m} = 0.$$

Divide and conquer for eigenproblem (2/2)

3. From u and the solutions of the secular equation, Löwner's Theorem provides vector $\widetilde{u}$ that is used to compute the eigenvector V_λ of $\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + c_m \widetilde{u}\widetilde{u}^t$

4. Let $W = (V_\lambda)_{\lambda \text{ eigenvalue of } U}$, we get the eigenvectors of U thanks to the multiplication $\begin{pmatrix} O_1 & 0 \\ 0 & O_2 \end{pmatrix} W$.

Additional details on step 1



Advantage: Pure divide and conquer algorithm, it prevents to have eigenvalues of multiplicity larger than two at each conquering step.

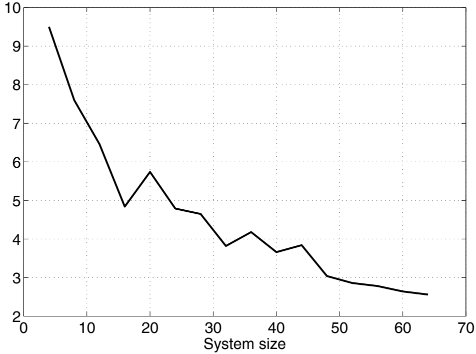
Additional details on step 2

Use of Gragg's scheme (based on Newton's method):

Choose h_k such that $h_k(\lambda) = x_{k,0} + x_{k,1}/(\lambda_k - \lambda) + x_{k,2}/(\lambda_{k+1} - \lambda)$ matches $\displaystyle\sum_{i=1}^{n} \frac{u_i^2}{\lambda_i - \lambda} + \frac{1}{c_m}$ at its root $\in (\lambda_k, \lambda_{k+1})$ up to the second derivative.

Advantage: Cubic monotonic convergence.

Comparison with Householder tridiagonalization



- Small matrices.
- Iterative algorithm to solve the secular equation.
- Divergence produced by deflation.

Must we systematically use Householder tridiagonalization with divide & conquer when we suspect the random linear systems to be ill-conditioned?

Our answer

- Perform Householder tridiagonalization $O(4n^3/3)$ and solve the linear systems cheaply using parallel cyclic reduction $O(n\log_2(n))$.
- Take a decision according to the value of the residue error:
  - If the residue error is small then we already have good solutions.
  - Otherwise, we must perform divide & conquer $O(4n^3/3)$ diagonalizations and discard the smallest eigenvalues.
- The next time we solve this same kind of linear systems:
  - If they used to be well-conditioned then we just process LDLt $O(n^3/6)$.
  - Otherwise we execute directly the combination of Householder tridiagonalization and divide & conquer diagonalization.

Outline of the talk

Summary of contributions

- CUDA source code of: LDLt, Householder reduction, parallel cyclic reduction that is not necessary a power of two and divide and conquer for eigenproblem.
- Execution time comparison of the different methods mentioned above.
- Original method to further optimize the adaptation of LDLt to our context.
- Original parallel cyclic reduction that can be used for any vector size and not only a power of two.
- Precise answer to the following question: Must we systematically use Householder tridiagonalization with divide & conquer when we suspect the random linear systems to be ill-conditioned?

Future work

- Studying the rounding errors and error propagation.
- Use CADNA library to test each procedure: `http://www-pequan.lip6.fr/cadna/`

Source code

- `http://www.proba.jussieu.fr/~abbasturki/soft.htm`

References

- L.A. Abbas-Turki and Stef Graillat. Resolution of a large number of small random symmetric linear systems in single precision arithmetic on GPUs: `https://hal.archives-ouvertes.fr/hal-01295549`