HPCLatAm 2013, pp. 80-88 (full paper)
Session: GPU Architecture and Applications
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Towards a Distributed GPU-Accelerated Matrix Inversion

Gerardo Ares[1,2], Pablo Ezzatti[2], and Enrique S. Quintana-Ortí[3]

[1] Bull, 01227-901-São Paulo, Brazil. `gerardo.ares@lam-bull.com`
[2] Instituto de Computación, Universidad de la República, 11.300-Montevideo, Uruguay. `{gares,pezzatti}@fing.edu.uy`
[3] Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain. `quintana@icc.uji.es`

Abstract. We present an extension of a GPU-based matrix inversion algorithm for distributed memory contexts. Specifically, we implement and evaluate a message-passing variant of the Gauss-Jordan method (GJE) for matrix inversion on a cluster of nodes equipped with GPU hardware accelerators. The experimental evaluation of the proposal shows a significant runtime reduction when compared with both the distributed non-GPU implementation of GJE and a conventional method based on the LU factorization (as implemented in ScaLAPACK). In addition to this, our proposal leverages the aggregated capacity of the GPU memories in the cluster to overcome the constraints imposed by the reduced memory space of these devices.

1 Introduction

Though the explicit inversion of matrices can often be by-passed by solving the corresponding systems of linear equations, there are situations where the inverse of a matrix itself is of interest. Examples include earth sciences [14], the matrix sign function method for spectral decomposition [13], the low rank radiosity technique for computer graphics [8], and other disciplines (see [11] for a detailed list). Here, we focus on the inversion of large-scale general matrices (say, more than several thousands of rows/columns). This operation, like many other dense linear algebra computations, requires an important computational effort in terms of execution time and memory, which motivates the use of distributed-memory platforms and graphics processors (GPUs) [15, 2, 3].

The conventional strategy for matrix inversion is based on Gaussian elimination (i.e., the LU factorization) and is available in LAPACK [1] for shared-memory platforms and ScaLAPACK [6] for distributed (i.e., message-passing) contexts. However, following previous work on matrix inversion implementation on clusters [12] and studies about the adoption of GPUs to speed up this computation [4], we focus on the Gauss-Jordan elimination algorithm (GJE). This alternative method is, in essence, a reordering of Gaussian elimination, which casts the bulk of computations in terms of matrix-matrix products, and presents several other appealing properties.

Taking into account the previous exposition, in this paper we analyze the extension of the GJE method for (general) matrix inversion to a distributed platform composed of hybrid CPU+GPU nodes. Specifically, we present and evaluate high performance implementations of the GJE method on two different systems, a cluster of multi-core CPUs and a cluster composed of hybrid CPU+GPU nodes. Numerical experiments illustrate that our GPU-based version outperforms the distributed CPU version as well as the shared-memory implementation in LAPACK. On the other hand, both variants exhibit reasonable *weak scalability*, and the GPU-based version offers acceptable *strong scalability* for medium size problems, and high performance for large problems. Furthermore, our proposal aggregates the memory of all GPUs in the cluster, thus avoiding for this matrix operation one of the most commonly bottlenecks in GPU computing, namely the limited capacity of the accelerator memories.

The rest of the paper is structured as follows. In Section 2 we review the GJE approach for matrix inversion. Additionally, we offer an introduction to matrix inversion on distributed contexts and review the efforts related to the use of GPUs to accelerate this operation. Several high performance implementations of distributed GJE method are described and evaluated in Sections 3 and 4, respectively. Finally, a few concluding remarks and future work is presented in Section 5.

2 Matrix Inversion: Methods and Parallelization

GJE for matrix inversion is, in essence, a reordering of the computations performed by the traditional matrix inversion method based on Gaussian elimination. Therefore, it is natural that both procedures feature the same arithmetic cost: $2n^3$ flops (floating-point arithmetic operations) [9, 12], where n denotes the matrix dimension. Figure 1 illustrates a blocked version of the GJE algorithm for the inversion of a matrix A using the FLAME notation. In the illustration, $m(A)$ stands for the number of rows of matrix A. We believe the rest of the notation is intuitive but, for further details, see [10, 5]. A description of the unblocked version of GJE, called from inside the blocked one, can be found in [12]. It should be noted that, upon termination, the contents of A are overwritten with those of its inverse (i.e., this is an in-place procedure). Also, all the developed implementations that are evaluated next include partial pivoting which, in practice, ensures numerical stability. For simplicity, though, we do not consider the application of pivoting during the following presentation, but details can be found in [12].

2.1 Matrix inversion on distributed contexts

Inversion of a general matrix $A \in \mathbb{R}^{n\times n}$ via the LU factorization can be accomplished in three steps. First the matrix is decomposed as $A = LU$, where $L \in \mathbb{R}^{n\times n}$ and $U \in \mathbb{R}^{n\times n}$ are unit lower and upper triangular factors, respectively; the upper triangular factor is then explicitly inverted: $U \rightarrow U^{-1} = \bar{U}$; and

```
Algorithm: $[A] := \text{GJE\_BLK}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$
while $m(A_{TL}) < m(A)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
        where $A_{11}$ is $b \times b$

    $\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix} := \text{GJE\_UNB}\left(\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix}\right)$    Unblocked Gauss-Jordan
        $A_{00} := A_{00} + A_{01}A_{10}$    Matrix-matrix product
        $A_{20} := A_{20} + A_{21}A_{10}$    Matrix-matrix product
        $A_{10} := A_{11}A_{10}$             Matrix-matrix product
        $A_{02} := A_{02} + A_{01}A_{12}$    Matrix-matrix product
        $A_{22} := A_{22} + A_{21}A_{12}$    Matrix-matrix product
        $A_{12} := A_{11}A_{12}$             Matrix-matrix product

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
endwhile
```

Fig. 1. Blocked algorithm for matrix inversion via GJE without pivoting.

finally, the (lower unit) triangular system $A^{-1}L = \bar{U}$ is solved for the sought-after inverse A^{-1}.

LAPACK [1] is a high-performance linear algebra library (for shared-memory multiprocessors) which provides routines that cover the functionality required in the previous steps. In particular, routine `getrf` obtains the LU factorization (with partial pivoting) of a nonsingular matrix, while routine `getri` computes the matrix inverse using the triangular factors obtained by `getrf`. Additionally, ScaLAPACK extends most of LAPACK's functionality to distributed memory contexts, providing analogous routines.

The traditional method for matrix inversion, using the LU factorization (and ScaLAPACK), presents some difficult features for high performance parallel computing. In particular, this approach operates with triangular matrices which often yields load imbalance. This problem is identified in [12] where both methods, Gaussian elimination and GJE, are compared. Concretely, the authors had evaluated the inversion of general and SPD matrices in a cluster with 32 processors, concluding that GJE clearly outperforms the LU-factorization and Cholesky-based methods.

2.2 Matrix inversion accelerated with GPUs

In [4], several strategies to improve the performance of matrix inversion were evaluated on a hybrid CPU+GPU system. That work considers inversion of general, SPD and symmetric indefinite matrices, and leverages techniques such as hybrid computation, concurrent execution and look-ahead during the computation of this particular operation. The results show speed ups of up to $4.5\times$ on a Intel Xeon E5520 quad-core connected to an NVIDIA Tesla C2050 over the non-GPU implementation.

The utilization of several GPUs, up to four, connected to only one host to invert general matrices is addressed in [7], illustrating the benefits that a multi-GPU platform can render for fast inversion of large general matrices.

In summary, the results in both articles confirm that GJE is a highly appealing approach to invert general matrices on massive parallel contexts.

3 Distributed Implementations of Matrix Inversion

We have implemented two variants, both based on GJE, for distributed matrix inversion. The homogeneous variant is designed to compute on traditional distributed-memory hardware platforms with CPU nodes, and the hybrid one is conceived to take advantage of clusters with nodes that include GPUs.

3.1 Matrix inversion on (homogeneous) CPU clusters

GJE presents the appealing property that all iterations of the basic algorithm (see Figure 1) perform the same number of operations. Therefore, a parallel implementation of this algorithm does not require a cyclic distribution of the data to balance the workload and, instead, a blocked data layout suffices for this purpose. On the other hand, while 2D distributions are, in general, to be preferred for scalability reasons, given that the control theory problems that we are addressing in practice do not involve matrices with more a few ten of thousands rows/columns, for simplicity we restrict our implementations to use a 1D block data distribution among the cluster nodes.

Assume thus that the matrix to invert is distributed among the c nodes of the cluster following a column block data layout, with $d = n/c$ consecutive columns assigned to each node (for simplicity, we assume that n is an integer multiple of c). At each iteration of the main loop in the GJE algorithm, the multicore CPUs of the node where the current column panel ($\left[A_{11}^T, A_{21}^T, A_{31}^T\right]^T$ of width $b \ll d$) is mapped to, collaborate to compute the factorization of this panel locally using algorithm GJE_UNB. After that the factored column panel and the local pivot indexes are broadcast to the remaining nodes, which then perform the update of their local blocks with a single matrix-matrix product. This procedure is repeated until the whole matrix is processed.

Within each node, multicore parallelism is exploited by invoking a multi-threaded version of BLAS to perform the matrix-matrix product required by the update and also part of the operations that appear in algorithm GJE_BLK.

3.2 Matrix inversion on hybrid CPU+GPU clusters

While this variant also leverages a 1D block distribution by columns, in this case the portion of the matrix assigned to one node is stored in the memory of the local GPU. Now, at each iteration, the factorization of the current panel is also performed by the multicore CPUs, which requires a previous transfer of these data from the local GPU memory to the local main memory of the node where this block resides. This result is then broadcast to the remaining nodes and all them then transfer it from their local main memory to their local GPU memory, where the updates of the remaining parts of the matrix are performed using the data-parallel implementation of the matrix-matrix product available in NVIDIA's BLAS.

4 Experimental Evaluation

4.1 Experimental platform

All the following experiments were performed using IEEE single precision on two different platforms:

- CPU cluster: 8 nodes equipped with 2 Intel Sandy-Bridge processors at 2.70 GHz (8 cores per processor) and 64 GBbytes of RAM, connected by an Infiniband QDR (40 Gbits/s), and running on OS bullx Linux Server release 6.3 (V1). The libraries include Intel v12.1.4 compiler, Intel MKL v11.0 library (for BLAS) and bullxmpi v1.2.4.1 (for communication, including MPI).
- GPU+CPU cluster: 4 nodes with 2 Intel Xeon E5640 (Nehalem) at 2.67 GHz (quad-core) and 24 GBbytes of RAM. Each node also features 2 NVIDIA Tesla M2050 (Fermi) GPUs (448 cores and 3 GBbytes of RAM per GPU), and the interconnect is Infiniband QDR. The software consists of bullx Linux Server release 6.3 (V1), Intel v12.1.4 compiler, Intel MKL v10.3 library, bullxmpi v1.2.4.1, and CUDA 4.0.17.

4.2 Experiments

In our first experiment, we evaluate the GFLOPS (10^9 flops/s) achieved by both distributed versions, $\mathrm{GJE}_{\mathrm{CPU}}$ and $\mathrm{GJE}_{\mathrm{GPU}}$, when inverting matrices with dimensions between 64K and 74K; see Figure 2. In this case, the $\mathrm{GJE}_{\mathrm{CPU}}$ version employs the whole platform, i.e. the 8 nodes of the CPU cluster, while $\mathrm{GJE}_{\mathrm{GPU}}$ leverages the 4 nodes–8 GPUs of GPU+CPU cluster. Additionally to this two distributed matrix inversion algorithms, for reference we also include the LU-based inversion method implemented in LAPACK, running in a single node, in our experimentation. In the results presented hereafter the transfer costs of moving data between CPU and GPU are always included in the total runtime of the GPU versions.

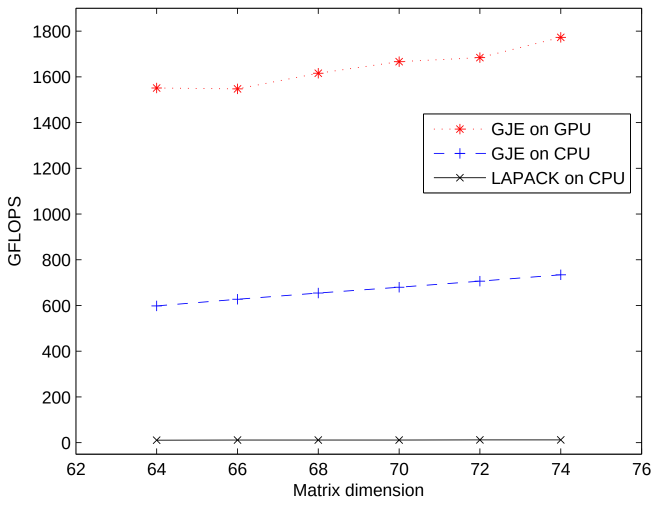

Fig. 2. Performance (in GFLOPS) of $\mathrm{GJE}_{\mathrm{GPU}}$, $\mathrm{GJE}_{\mathrm{CPU}}$ and the LAPACK-based variants when applied to invert matrices with dimensions between 64K and 74K.

The results in Figure 2 show that both distributed implementations of matrix inversion (CPU-based and hybrid) undoubtedly outperform the single-node LAPACK version (*shared-memory* parallelism), respectively obtaining speed-ups of up to 40× and 100× with respect to that. Moreover, the distributed GPU-based version presents a considerably higher GFLOPS rate (and, therefore, a much shorter runtime) than the distributed CPU-based variant, attaining up to 1,770 GFLOPS to invert the largest matrix covered in our experimental evaluation, which represents an speed-up of up to 2.6× with respect to the CPU-based variant.

The three plots in Figure 3 analyze the scalability of our proposals. Particularly, there we evaluate the GFLOPS achieved by the $\mathrm{GJE}_{\mathrm{CPU}}$ and $\mathrm{GJE}_{\mathrm{GPU}}$ implementations for an increasing numbers of nodes and GPUs respectively (1, 2, 4 and 8), and three different dimensions: small (10K), medium (32K) and large (74K) cases.

Consider first the *weak scalability*, i.e., how the solution time varies with the number of computational resources (CPUs or GPUs) for a fixed problem size per resource. Figure 3 show that both $\mathrm{GJE}_{\mathrm{CPU}}$ and $\mathrm{GJE}_{\mathrm{GPU}}$ perform reasonable

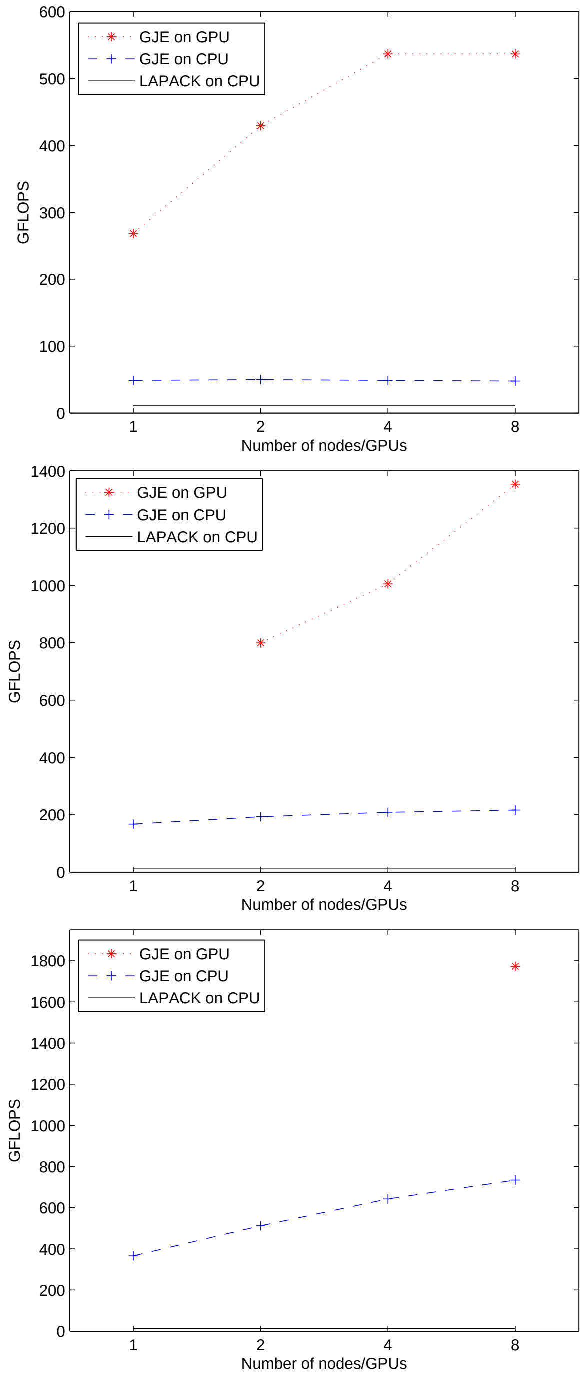

Fig. 3. Performance (in GFLOPS) of $\mathrm{GJE_{GPU}}$, $\mathrm{GJE_{CPU}}$ and LU-based variants to invert small, medium and large matrices (top, middle and bottom, respectively).

well in this metric as the addition of more resources to tackle larger problems results in a fair increase of performance.

On the other hand, from the point of view of *strong scalability* (i.e., how the solution time varies with the number of computational resources for a fixed total problem size), this experiment reports poor results for the small case, as only GJE_{GPU} increases its GFLOPS rate and then, only for up to 4 GPUs. For the medium problem dimensions, there is a lack of *strong scalability* for GJE_{CPU} but GJE_{GPU} exhibits a fair behavior. Finally, for the large cases, GJE_{CPU} presents a good *strong scalability.* Unfortunately, the resolution of the 74K problem requires at least 22 GBbytes of memory to keep the whole matrix so this case cannot be run using less than 8 GPUs, making it impossible carry out the scalability study.

5 Concluding remarks and future work

We have conducted an initial study of the benefits of applying GPU acceleration to invert general matrices in distributed memory platforms. Specifically, we implemented a distributed (i.e., message-passing) version of the GJE method using MPI for the communication and synchronization, and CUDA for the interaction with the GPUs. The experimental analysis showed that the novel implementation significantly reduces the runtime of matrix inversion, with speed-ups of up to 2.6× and 100× when compared to the CPU-based and LAPACK (LU-based) versions, respectively. Furthermore, the GPU-based proposal exhibits a reasonable degree of weak scalability and acceptable values of strong scalability.

An additional highlight of our proposal is that the coordinated use of several GPUs increases the aggregated memory space, allowing us to tackle larger problems, and addressing an important bottleneck of GPU technology.

Future research lines resulting from this experience will include:

– Use of large platforms to further reduce the computational times and increase the dimension of the problems that can be tackled.
– Migrate the codes to a 2D block data distribution.
– Incorporate GPU Direct to advance in the scalability of the proposal.

Acknowledgments

The authors would like to Alfredo Remón for his technical support, and Bull for granting the access to the hardware facilities employed in the performance evaluation.

Enrique S. Quintana-Ortí was supported by the CICYT project TIN2008–06570-C04-01 and FEDER.

References

1. E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.

2. Sergio Barrachina, Maribel Castillo, Francisco D. Igual, Rafael Mayo, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. Exploiting the capabilities of modern GPUs for dense matrix computations. *Concurr. Comput. : Pract. Exper.*, 21:2457–2477, December 2009.
3. P. Benner, P. Ezzatti, E. S. Quintana, and A. Remón. Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function. In *Lecture Notes in Computer Science, 7th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks – HeteroPar'09*, volume 6043, pages 132–139. Springer Berlin / Heidelberg, 2010.
4. P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. Matrix inversion on CPU-GPU platforms with applications in control theory. *Concurrency & Computation: Pract. & Exp.*, 25(8):1170–1182, 2013.
5. P. Bientinesi, J. A. Gunnels, M. E. Myers, E. S. Quintana-Ortí, and R. A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft.*, 31(1):1–26, March 2005.
6. J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE Comput. Soc. Press, 1992.
7. P. Ezzatti, E.S. Quintana-Ortí, and A. Remón. High performance matrix inversion on a multi-core platform with several gpus. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 87–93, 2011.
8. E. Fernández and G. Besuievsky. Inverse lighting design for interior buildings integrating natural and artificial sources. *Computers & Graphics*, 36(8):1096–1108, 2012.
9. A. V. Gerbessiotis. Algorithmic and Practical Considerations for Dense Matrix Computations on the BSP Model. PRG-TR 32, Oxford University Computing Laboratory, 1997.
10. J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.
11. Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
12. E.S. Quintana-Ortí, G. Quintana-Ortí, X. Sun, and R.A. van de Geijn. A note on parallel matrix inversion. *SIAM J. Sci. Comput.*, 22:1762–1771, 2001.
13. J.D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. 32:677–687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).
14. B.D. Tapley, B.E. Schutz, and G.H. Born. *Statistical Orbit Determination.* Elsevier Academic Press, 2004.
15. Vasily Volkov and James Demmel. LU, QR and Cholesky factorizations using vector capabilities of GPUs. *Technical Report No. UCB/EECS*, 49, May 2008.