Trading Off Performance for Energy in Linear Algebra Operations with Applications in Control Theory

Peter Benner
Max Planck Institute for Dynamics of Complex Technical Systems,
Magdeburg, Germany, D-39106,
benner@mpi-magdeburg.mpg.de

and

Pablo Ezzatti
Facultad de Ingeniería, Universidad de la República,
Montevideo, Uruguay, 11300,
pezzatti@fing.edu.uy

and

Enrique S. Quintana-Ortí
Departamento de Ingeniería y Ciencia de Computadores,
Universidad Jaume I,
Castellón, Spain, 12.071,
quintana@icc.uji.es

and

Alfredo Remón
Max Planck Institute for Dynamics of Complex Technical Systems,
Magdeburg, Germany, D-39106,
remon@mpi-magdeburg.mpg.de

Abstract

We analyze the performance-power-energy balance of a conventional Intel Xeon multicore processor and two low-power architectures –an Intel Atom processor and a system with a quad-core ARM Cortex A9+NVIDIA Quadro 1000M– using a high performance implementation of Gauss-Jordan elimination (GJE) for matrix inversion. The blocked version of this algorithm employed in the experimental evaluation mostly comprises matrix-matrix products, so that the results from the evaluation carry beyond the simple matrix inversion and are representative for a wide variety of dense linear algebra operations/codes.

Keywords: Dense Linear Algebra, Gauss–Jordan, Power, Energy

1 Introduction

General-purpose multicore architectures and graphics processor units (GPUs) dominate today's landscape of high performance computing (HPC), offering unprecedented levels of raw performance when aggregated to build the systems of the Top500 list [1]. While the performance-power trade-off of HPC platforms has also enjoyed considerable advances in the past few years [2] —mostly due to the deployment of heterogeneous platforms equipped with hardware accelerators (e.g., NVIDIA and AMD graphics processors, Intel Xeon Phi) or the adoption of low-power multicore processors (IBM PowerPC A2, ARM chips, etc.)— much remains to

be done from the perspective of energy efficiency. In particular, power consumption has been identified as a key challenge that will have to be confronted to render Exascale systems feasible by 2020 [3, 4, 5]. Even if the current progress pace of the performance-power ratio can be maintained (a factor of about 5× in the last 5 years [2]), the ambitious goal of yielding a sustained ExaFLOPS (i.e., 10^{18} floating-point arithmetic operations, or flops, per second) for 20–40 MWatts by the end of this decade will be clearly exceeded.

In recent years, a number of HPC prototypes have proposed the use of low-power technology, initially designed for mobile appliances like smart phones and tablets, to deliver high MFLOPS/Watt rates [6, 7]. Following this trend, in this paper we investigate the performance, power and energy consumption of two low-power architectures, concretely an Intel Atom and a hybrid system composed of a multicore ARM processor and an NVIDIA 96-core GPU, and a general-purpose multicore processor, using as a workhorse matrix inversion via Gauss-Jordan elimination (GJE) [8]. While this operation is key for the solution of important matrix equations arising in control theory via the matrix sign function [9, 10], the relevance of this study carries beyond the inversion operation/method or these specific applications. In particular, a blocked implementation of matrix inversion via GJE casts the bulk of the computations in terms of the matrix-matrix product, so that its performance as well as power dissipation and energy consumption are representative for many other dense linear algebra operations such as, e.g., the solution of linear systems, linear-least squares problems, eigenvalue computations, etc.

The rest of the paper is structured as follows. In Section 2 we briefly review matrix inversion via the GJE method and an applications of this particular operation. Specifically, we introduce the sign function, which plays an important role for the resolution of several scientific problems arising in control theory. Next, in Section 3, we describe the specific implementation of the GJE method on the two low-power architectures selected for our study: *i)* an Intel Atom processor not much different, from the programming point of view, from a mainstream multicore processor like the Intel Xeon or the AMD Opteron; and *ii)* a hybrid board with ARM+NVIDIA technology that can be viewed as a low-power version of the heterogeneous platforms equipped with hardware accelerators that populate the first positions of the Top500 list. Finally, Sections 4 and 5 contain, respectively, the experimental evaluation and a few concluding remarks resulting from this investigation.

2 Matrix Inversion via GJE and its Applications to Control Theory

The traditional approache to compute a matrix inverse is based on the LU factorization and consist of the following three steps:

1. Compute the LU factorization $PA = LU$, where $P \in \mathbb{R}^{n\times n}$ is a permutation matrix, and $L \in \mathbb{R}^{n\times n}$ and $U \in \mathbb{R}^{n\times n}$ are, respectively, unit lower and upper triangular factors [11].
2. Invert the triangular factor $U \rightarrow U^{-1}$.
3. Solve the system $XL = U^{-1}$ for X.
4. Undo the permutations $A^{-1} := XP$.

An alternative approach to compute a matrix inverse is the GJE, an appealing method for matrix inversion in current architecture, becauase it presents a computational cost and numerical properties analogous to those of traditional approaches [8] but superior performance on a variety of parallel architectures, including clusters [12], general-purpose multicore processors and GPUs [9].

Figure 1 shows a blocked version of the GJE algorithm for matrix inversion using the FLAME notation. There $m(A)$ stands for the number of rows of matrix A while, for details on the notation, we refer the reader to [13, 14]. A description of the unblocked version of GJE, called from inside the blocked routine, can be found in [12]; for simplicity, we do not include the application of pivoting during the factorization, but details can be found there as well. Given a square (nonsingular) matrix of size $n = m(A)$, the cost of matrix inversion using this algorithm is $2n^3$ flops, performing the inversion in-place so that, upon completion, the entries of A are overwritten with those of its inverse.

Our primary interest for the GJE matrix inversion method is twofold. First, most of the computations of the blocked algorithm are matrix-matrix products (see Figure 1). Therefore, the conclusions from our power–energy–performance evaluation can be extended to many other dense linear algebra kernels such as the solution of linear systems via the LU and Cholesky factorizations, and least-squares computations using the QR factorization [11], among others.

Additionally, explicit matrix inversion is required during the computation of the sign function of a matrix A using the Newton iteration method [10], which we describe brief in the next sub-section.

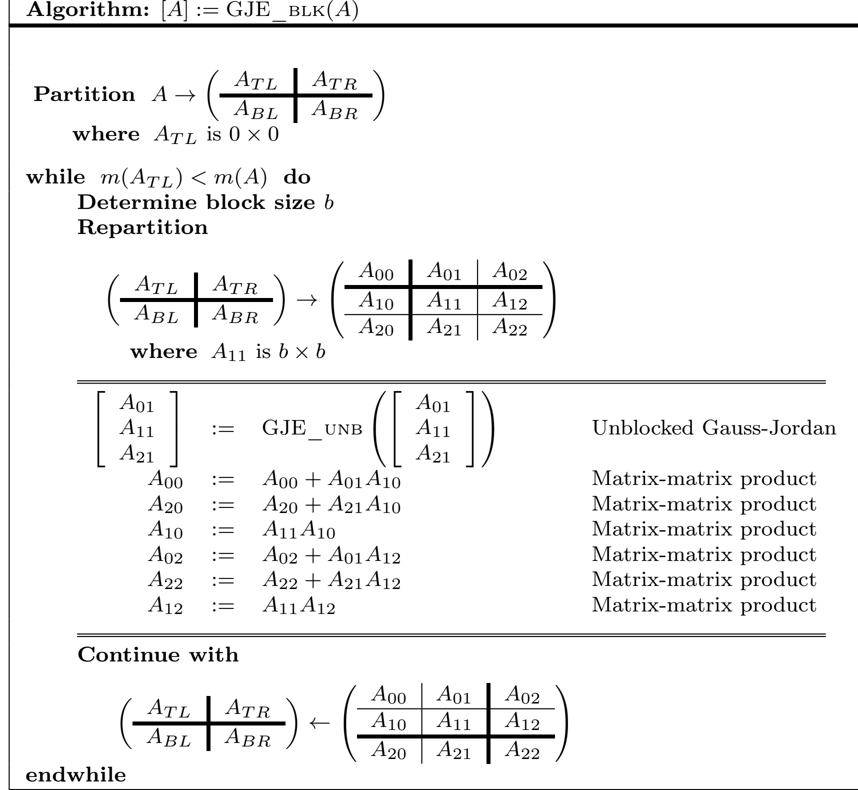

Figure 1: Blocked algorithm for matrix inversion via GJE without pivoting.

2.1 Matrix sign function

Consider a matrix $A \in \mathbb{R}^{n \times n}$ with no eigenvalues on the imaginary axis, and let

$$A = T^{-1} \begin{pmatrix} J_- & 0 \\ 0 & J_+ \end{pmatrix} T, \tag{1}$$

be its Jordan decomposition, where the eigenvalues of $J_- \in \mathbb{R}^{j \times j}$ and $J_+ \in \mathbb{R}^{(n-j)\times(n-j)}$ have negative and positive real parts [11] respectively.

The *matrix sign function* of A is then defined as

$$\text{sign}(A) = T^{-1} \begin{pmatrix} -I_j & 0 \\ 0 & I_{n-j} \end{pmatrix} T, \tag{2}$$

where I denotes the identity matrix of the order indicated by the subscript. The matrix sign function is a useful numerical tool for the solution of control theory problems (model reduction, optimal control) [15], and the bottleneck computation in many lattice quantum chromodynamics computations [16] and dense linear algebra computations (block diagonalization, eigenspectrum separation) [11, 17]. Large-scale problems as those arising, e.g., in control theory often involve matrices of dimension $n \to O(10,000 - 100,000)$ [18].

There are simple iterative schemes for the computation of the sign function. Among these, the Newton iteration, given by

$$\begin{aligned} A_0 &:= A, \\ A_{k+1} &:= \tfrac{1}{2}(A_k + A_k^{-1}),\ k = 0, 1, 2, \ldots, \end{aligned} \tag{3}$$

is specially appealing for its simplicity, efficiency, parallel performance, and asymptotic quadratic convergence [17, 19, 20]. However, even if A is sparse, $\{A_k\}_{k=1,2,\ldots}$ are in general full dense matrices and, thus, the scheme in (3) roughly requires $2n^3$ floating-point arithmetic operations (flops) per iteration.

3 High Performance Implementation of GJE on Multicore and Manycore Architectures

As previously stated, the GJE algorithm for matrix inversion casts the bulk of the computations in terms of matrix-matrix products; see Figure 1. In particular, provided that the block size b is chosen there as $64 \leq b \ll n$, the computational cost of the factorization of the "current" panel $\hat{A} = \left[A_{01}^T; A_{11}^T; A_{21}^T\right]^T$, performed inside the routine GJE_UNB, is negligible compared with that of the update of the remaining matrix blocks following that operation.

Therefore, the key to attaining high performance with GJE algorithm primarily relies on using a highly tuned implementation of the matrix-matrix product and, under certain conditions on parallel architectures, the reduction of the serial bottleneck that the factorization of $\hat{A}$ represents applying, e.g., a look-ahead strategy [21].

Fortunately, there exist nowadays highly efficient routines for the matrix-matrix multiplication, embedded into mathematical libraries such as Intel MKL, AMD ACML, IBM ESSL, or NVIDIA CUBLAS; but also as part of independent development efforts like GotoBLAS2 [22] or OpenBLAS [23]. Therefore, for the implementation of GJE in the Atom processor (as well as for the Intel Xeon processor that will be used for reference in the experimental evaluation), we simply leverage the matrix-matrix product kernel `sgemm` in a recent version of Intel MKL.

Let us consider next the hybrid SECO development kit [24], which combines a quad-core NVIDIA Tegra3/ARM Cortex A9 processor and an NVIDIA Quadro 1000M GPU with 96 cores, both processors in single board (see Figure 2).

Figure 2: SECO board, which includes a quad-core ARM processor and a Quadro GPU.

The properties of the GJE algorithm and the hybrid nature of the target platform ask for an implementation that harnesses the concurrency of the operation while paying special attention to diminish the negative impact of communications between the memory address spaces of the Cortex A9 processor and the Quadro GPU. In a previous work we introduced a CPU-GPU implementation of the GJE algorithm for matrix inversion [9], and demonstrated the benefits of mapping each operation to the most convenient device: multicore processor or manycore accelerator. In this work we apply a similar approach to obtain a tuned implementation of the GJE algorithm for the SECO platform. The highly parallel matrix-matrix products are computed in the GPU. On the other hand, the panel factorizations performed with the unblocked algorithm GJE_UNB, which consist of fine grain operations, are computed in the multicore CPU. This algorithm is summarized in Figure 3 (note that, for simplicity, pivoting is ommitted in the figure, though partial column pivoting is performed in all our implementations). The block size b is tuned for the architecture and also for each problem dimension.

Additionally, we include a look-ahead technique that allows to overlap the factorization in step $(k+1)$-th with part of the updates performed during iteration k, and also keeps a low communication overhead.

Finally, the parallelism intrinsic to the linear algebra operations that appear in algorithms GJE_BLK and GJE_UNB are exploited using parallel implementations of the BLAS. In particular, we employ kernels from libraries CUBLAS and reference BLAS (parallelized with OpenMP directives), for the Quadro GPU and the ARM processor respectively.

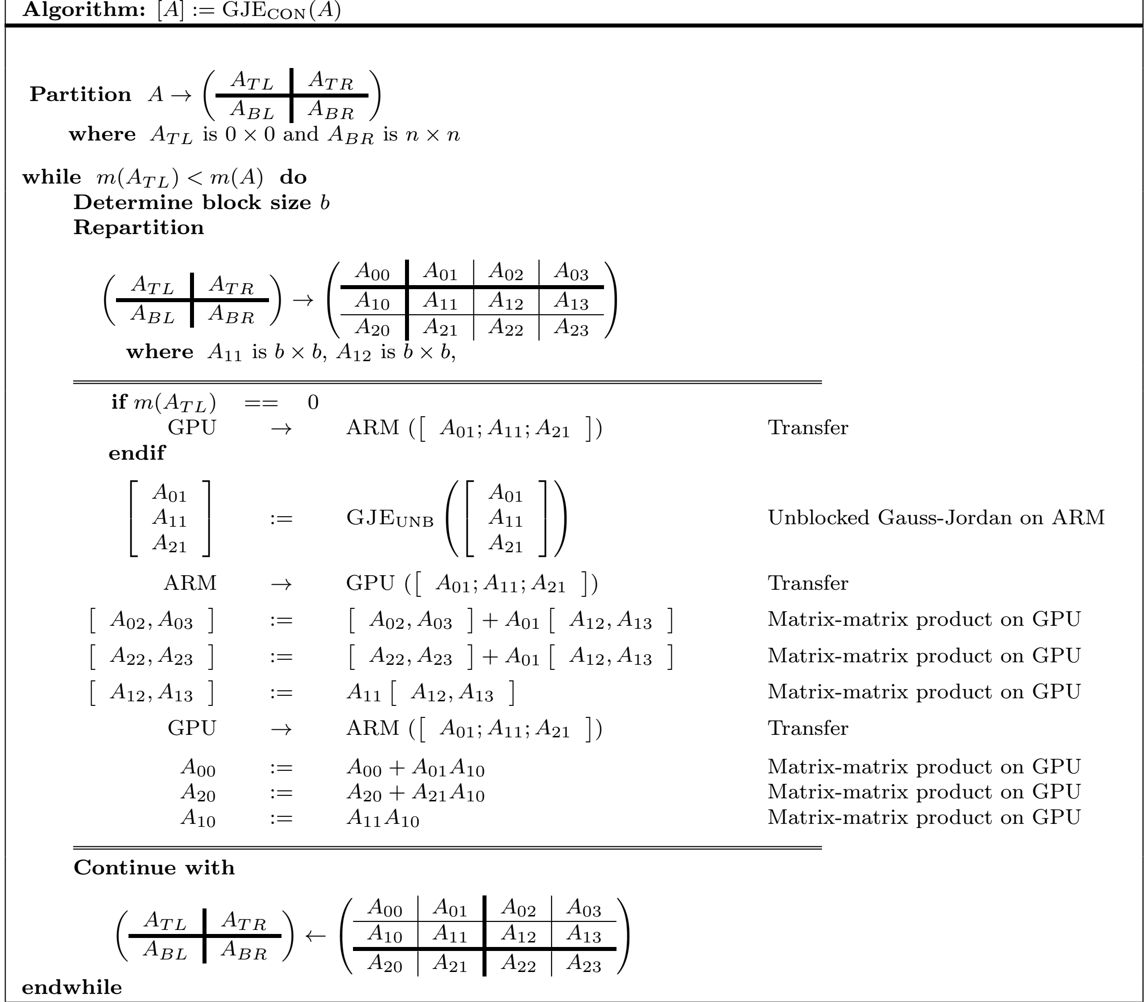

Figure 3: Blocked algorithm for matrix inversion via GJE without pivoting in the SECO platform.

4 Evaluation

This section is divided into three parts. First, we introduce the target platforms. This is followed by a description of the power measurement system. Finally, the results obtained are presented and analyzed.

4.1 Hardware platforms

We evaluate the matrix inversion routines on three target hardware platforms: a state-of-the-art server equipped with two multicore Intel Xeon ("Nehalem") processors, an Intel Atom-based laptop, and a hybrid ARM+NVIDIA board from SECO. Details about the hardware and the compilers employed in each platform can be found in Table 1.

The inversion routines for the Xeon and Atom processors heavily rely on the matrix-matrix product kernel in Intel MKL (versions 10.3 and 11.0, respectively). The hybrid implementation for the SECO platform makes intensive use of the kernels in CUBLAS (version 5.0) and the legacy implementation of BLAS[1] parallelized with OpenMP. (We note, however, that the amount of computation that is performed in the cores of the Cortex A9 processor is small, and mostly based on BLAS-1 and BLAS-2 operations, so that we do not expect significant differences if a tuned version of BLAS was used for this architecture.)

The codes were compiled with the `-O3` optimization flag and all the computations were performed using single precision arithmetic.

[1] Available at `http://www.netlib.org/`.

Table 1: Architectures employed in the experimental evaluation and the average power dissipation while idle (P_I)

Plat.	Processor	#Cores	Freq. (GHz)	RAM size & type	Peak (GFs)	Comp.	P_I (Watts)
Xeon	2 × Intel Xeon E5504	8	2.0	32 GB DDR3	128.0	`icc` v12.1.3	67.0
Atom	Intel Atom N270	1	1.6	1 GB DDR2	3.2	`gcc` v4.6.3	11.6
SECO	ARM Cortex A9 NVIDIA Quadro 1000M	4 96	1.3 1.4	2 GB, DDR3L 2 GB, DDR3	270.0	`gcc` v4.5	11.2

4.2 Power measurement

In order to measure power, we connected a WattsUp?Pro wattmeter (accuracy of ±1.5% and a rate of 1 sample/sec.) to the power line from the electric socket to the power supply unit (PSU), collecting the results on a separate server. All tests were executed for a minimum of 1 minute, after a warm up period of 2 minutes.

Since some of the platforms where the processors are embedded contain other devices —e.g., disks, network interface cards, and on the Atom laptop even the LCD display— on each platform we calculated the average power while idle for 1 minute, P_I, and then used this value to calculate the *net energy*, corresponding to the consumption after subtracting P_I from the power samples. We expect this measure allows a fair comparison between the architectures, as in this manner we only evaluate the energy that is necessary to do the actual work.

4.3 Experimental evaluation

The experimental evaluation is performed in two stages. First, we analyze the performance and power-energy consumption of the GJE for matrix inversion algorithm. Next, we study the impact of Gauss-Jordan inversion method on the resolution of matrix sign function.

4.3.1 Matrix inversion

Tables 2, 3 and 4 collect the results obtained from the execution of the different implementations of the GJE matrix inversion algorithm on the three target platforms, for problems of dimension n varying from 256 to 8,192. The same information is refined and collected graphically, in terms of GFLOPS and GFLOPS/Watt, in Figures 4, 5 and 6.

The results characterize the different performance-power-energy balance of the platforms: The Intel Xeon is considerably faster than the Intel Atom, in factors that range from more than 255× for the smaller problem dimensions, to about 50.8× for the larger ones; but the power dissipated by the Atom architecture is, depending on the problem size, 9.8 to 12.4× lower than that of the Intel Xeon architecture. The outcome of the combination of these two factors is that, from the perspective of total energy, the Intel Atom spends between 4.25 and 22.0× more energy than the Intel Xeon to compute the inverse; but the excess is only between 1.77 and 8.46× if we consider net energy. On the other hand, the SECO board presents quite an interesting balance. While being clearly slower than the Intel Xeon (especially for the smaller problems), this platform also shows a remarkable advantage from the point of view of energy efficiency. Thus, when the problem size is larger than 2,048, the ratios for the total and net energy of these two platforms are, respectively, up to 2.04 and 1.94× in favor of the SECO system.

Table 2: Time (in sec.); GFLOPS; average and maximum power consumption (P_{avg} and P_{max}, respectively, in Watts); and total and net energy (E_{tot} and E_{net}, respectively, in Joules) in Xeon

n	Time	GFLOPS	P_{avg}	P_{max}	E_{tot}	E_{net}
512	0.01	33.6	152.2	152.5	1.2	0.7
1,024	0.04	53.7	185.2	185.2	7.4	4.7
2,048	0.29	59.2	183.9	184.7	54.8	33.9
3,072	0.86	67.4	189.0	190.0	162.5	104.9
4,096	1.82	75.5	190.0	190.9	346.9	223.9
5,120	3.46	77.6	188.2	189.5	652.7	419.4
6,144	5.58	83.1	193.1	193.8	1,077.5	703.6
7,168	8.56	86.1	193.2	194.4	1,654.8	1,080.3
8,192	12.42	88.5	190.2	190.4	2,363.2	1,530.1

Table 3: Time (in sec.); GFLOPS; average and maximum power consumption (P_{avg} and P_{max}, respectively, in Watts); and total and net energy (E_{tot} and E_{net}, respectively, in Joules) in Atom

n	Time	GFLOPS	P_{avg}	P_{max}	E_{tot}	E_{net}
512	1.47	0.2	15.5	15.6	22.8	5.7
1,024	4.03	0.5	16.1	16.3	64.9	18.1
2,048	15.41	1.1	15.8	15.9	243.6	64.7
3,072	33.97	1.7	15.8	16.1	536.8	142.7
4,096	80.94	1.7	15.7	15.9	1,270.8	331.9
5,120	154.58	1.7	15.7	16.0	2,427.0	663.8
6,144	266.20	1.7	15.7	16.0	4,179.4	1,091.4
7,168	420.66	1.8	15.6	16.0	6,562.4	1,682.6
8,192	631.43	1.8	15.9	16.3	10,039.8	2,715.1

Table 4: Time (in sec.); GFLOPS; average and maximum power consumption (P_{avg} and P_{max}, respectively, in Watts); and total and net energy (E_{tot} and E_{net}, respectively, in Joules) in SECO

n	Time	GFLOPS	P_{avg}	P_{max}	E_{tot}	E_{net}
512	0.38	0.7	20.0	20.3	7.5	3.3
1,024	0.88	2.4	20.8	21.7	18.3	8.4
2,048	2.29	7.5	25.5	27.0	58.4	32.7
3,072	4.66	12.4	27.9	30.6	130.2	77.8
4,096	7.50	18.3	27.6	30.9	207.0	123.0
5,120	11.26	23.8	30.1	36.5	338.8	212.8
6,144	19.40	23.9	31.1	40.2	603.6	386.1
7,168	23.66	31.1	33.4	39.5	789.2	525.3
8,192	32.99	33.3	35.1	41.3	1,159.3	788.5

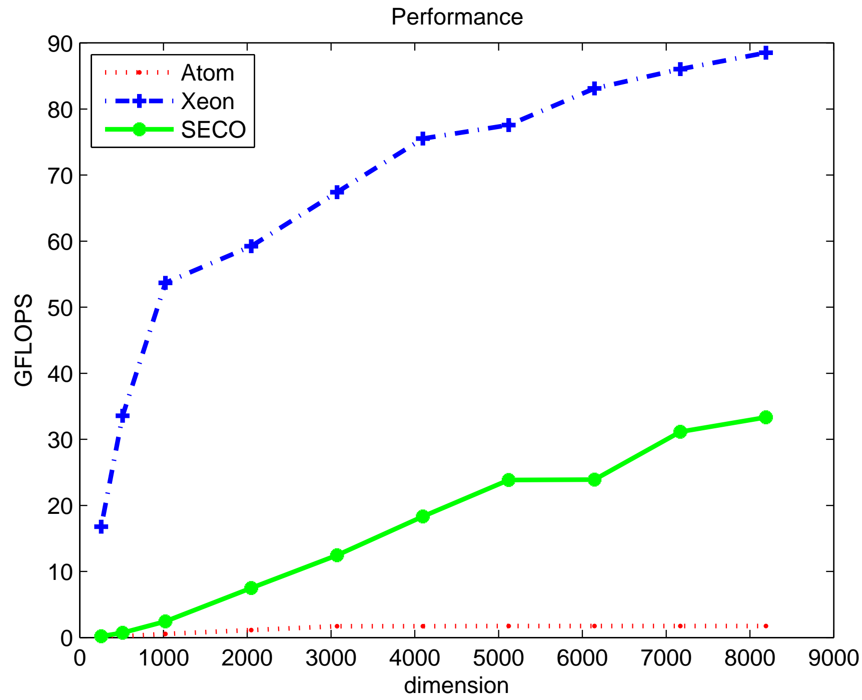

Figure 4: Performance in the target platforms

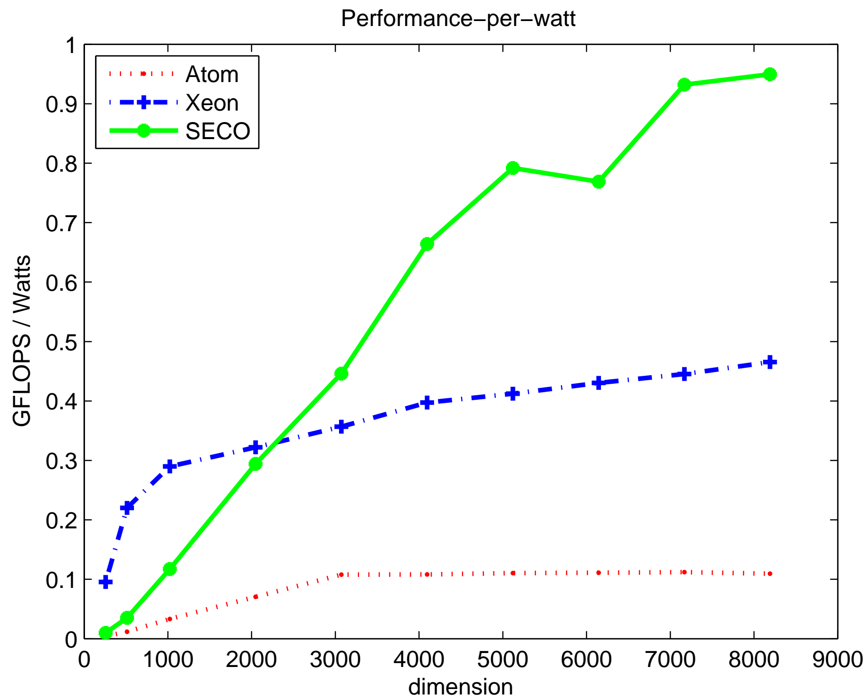

Figure 5: Total performance-per-watt in the target platforms

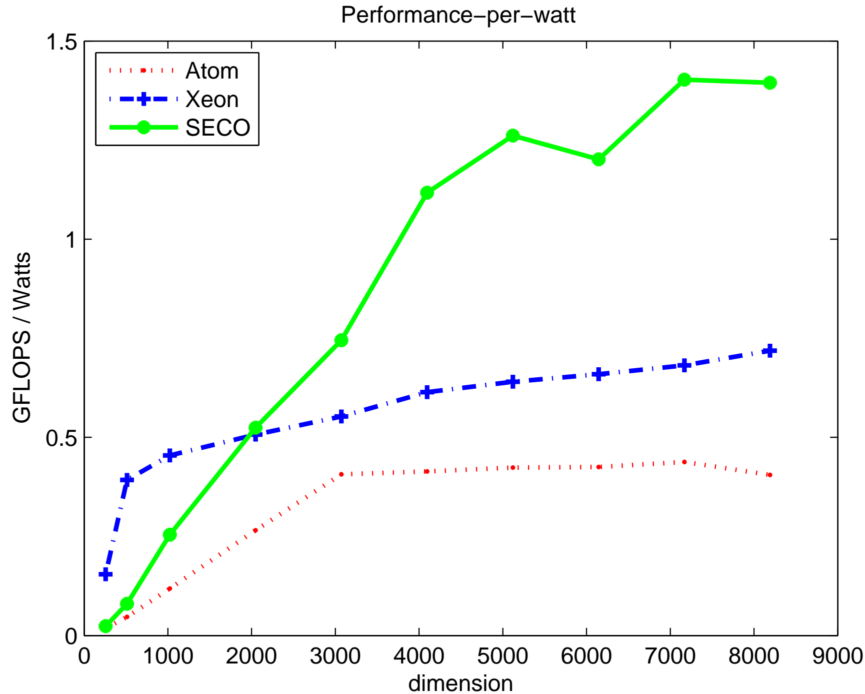

Figure 6: Net performance-per-watt in the target platforms

4.3.2 Matrix sign function

We next evaluate the performance and power-energy consumption to solve the matrix sign function (see Section 2.1) using the three target platforms, i.e. Xeon, Atom and SECO. Concretely, Table 5 presents the runtime to obtain the matrix sign function (in seconds) for four different problem dimensions, 256, 2,048, 5,120 and 8,192. Additionally, we discriminate the execution time related to the matrix inversions including the associated percentage in the overall process time. The time required for different problem dimensions can be easily infered from the results showed in the previous subsection and the data in Table 5. As the input matrices were randomly generated, and do not correspond to a real problem, the number of steps of the algorithm to reach the solution was fixed to 20 iterations.

Table 5: Time (in sec.) and percentage of matrix inversion runtime to calculate the matrix sign function

Platform	n	Time	% of inversion
Xeon	256	0.05	98.0
	2,048	5.20	97.6
	5,120	69.08	98.6
	8,192	249.14	99.3
Atom	256	1.30	95.9
	2,048	311.46	98.9
	5,120	3,111.57	99.3
	8,192	13,158.92	96.0
SECO	256	3.62	99.4
	2,048	47.35	96.7
	5,120	234.92	95.9
	8,192	685.24	96.3

The time evaluation summarized in Table 5 shows the close relation between the computational complexity of the matrix sign function algorithm and the matrix inversion kernel. Thus, the time required by the computation of the matrix inverses represents at least the 95% of the computational time of the overall process. This allows to easily compute a rough approximation of the energy consumption from the data obtained during the matrix inversion kernels evaluation. In particular, Table 6 summarizes the total runtime (in seconds) and the energy consumption (in Joules) for the computation of the matrix sign function on the three platforms and four different dimension cases (256, 2,048, 5,120 and 8,192).

Table 6: Time (in sec.) and total energy (Etot, in Joules) to calculate the matrix sign function

Platform	n	Time	E_{tot}
Xeon	256	0.05	8.8
	2,048	5.20	960.4
	5,120	69.08	13,090.7
	8,192	249.14	47,436.2
Atom	256	1.30	20.1
	2,048	311.46	4,921.1
	5,120	3,111.57	48,851.6
	8,192	13,158.92	209,226.8
SECO	256	3.62	69.1
	2,048	47.35	1,207.2
	5,120	234.92	7,071.0
	8,192	685.24	24,051.9

The highest energy consumption is required by Atom. Despite its low average power consumption, the large computational time leads to the worst results in terms of energy for this platform. Thus, the energy consumed by the Xeon is 4× lower for the largest problem tackled. On the other hand the lowest energy consumption is obtained for SECO, which requires 2× and 8× less energy than Xeon and Atom respectively. This is explained by the favorable performance-power ratio of the SECO platform.

5 Concluding Remarks and Future Directions

We have investigated the trade-off between performance and power-energy of three architectures using as a benchmark the matrix sign function, a useful mathematical tool in some numerical methods. In particular, the evaluation includes two low-power architectures and a conventional general-purpose multicore processor.

Our experimental evaluation is divided in two stages. First, the main computational kernel in the sign function algorithm, the general matrix inversion, is evaluated. Then, the complete sign function algorithm is assessed.

The use of blocked routines for GJE matrix inversion shows that, for dense linear algebra operations that are rich in matrix-matrix products, the *race-to-idle* strategy (i.e, execute the task as fast as possible, even if there is a high power dissipation associated with that) is crucial to attain both high throughput and performance-per-watt rates on general-purpose processor architectures, favoring power-hungry complex designs like the Intel Xeon processor over the Intel Atom counterpart. However, the experimentation also shows that a hybrid architecture that combines a low-power multicore processor and a limited GPU can offer competitive performance compared with that of the Intel Xeon platform, while being clearly superior from the perspective of energy efficiency.

The results obtained during the evaluation of the matrix sign function reinforce the conclussions extracted from the previous analisys.

Future research lines resulting from this experience will include:

- Evaluate in detail the power-energy consumption of each stage of the GJE method for general matrix inversion.
- Analyze the impact of other optimization techniques on memory-bounded dense linear algebra operations, and the adoption of dynamic frequency-voltage scaling (DVFS) and dynamic concurrency throttling (DCT) for certain stages of the algorithm.

- Extend our study to other high performance platforms, e.g. Kepler GPUs connected to high end multi-core CPU and extend the evaluation to other low-power processors with a large number of cores, such as ARM Cortex A15 processors.

Acknowledgments

The researcher from UJI was supported by the CICYT project TIN2011-23283 of the Ministerio de Economía y Competitividad, FEDER, and the EU Project FP7 318793 "EXA2GREEN". P. Ezzatti acknowledges support from Agencia Nacional de Investigación e Innovación (ANII) and Programa de Desarrollo de las Ciencias Básicas (PEDECIBA), Uruguay. The authors gratefully acknowledge Juan Pablo Silva and Germán León for their technical support with SECO hardware platform.

References

[1] "The top500 list," 2013, available at http://www.top500.org.

[2] "The Green500 list," 2013, available at http://www.green500.org.

[3] S. Ashby *et al*, "The opportunities and challenges of Exascale computing," Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, November 2010. [Online]. Available: http://science.energy.gov/~/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf

[4] J. Dongarra and *et al*, "The international ExaScale software project roadmap," *Int. J. of High Performance Computing & Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[5] M. Duranton and *et al*, "The HiPEAC vision for advanced computing in horizon 2020," 2013.

[6] "CRESTA: collaborative research into Exascale systemware, tools and applications," http://cresta-project.eu.

[7] "The Mont Blanc project," http://montblanc-project.eu.

[8] N. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[9] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, "Matrix inversion on CPU-GPU platforms with applications in control theory," *Concurrency and Computation: Practice & Experience*, vol. 25, no. 8, pp. 1170–1182, 2013. [Online]. Available: http://dx.doi.org/10.1002/cpe.2933

[10] J. Roberts, "Linear model reduction and solution of the algebraic Riccati equation by use of the sign function," *Internat. J. Control*, vol. 32, pp. 677–687, 1980, (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).

[11] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore: The Johns Hopkins University Press, 1996.

[12] E. Quintana-Ortí, G. Quintana-Ortí, X. Sun, and R. van de Geijn, "A note on parallel matrix inversion," *SIAM J. Sci. Comput.*, vol. 22, pp. 1762–1771, 2001.

[13] 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.*, vol. 31, no. 1, pp. 1–26, March 2005. [Online]. Available: http://doi.acm.org/10.1145/1055531.1055532

[14] 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.*, vol. 27, no. 4, pp. 422–455, Dec. 2001. [Online]. Available: http://doi.acm.org/10.1145/504210.504213

[15] P. Petkov, N. Christov, and M. Konstantinov, *Computational Methods for Linear Control Systems.* Hertfordshire, UK: Prentice-Hall, 1991.

[16] A. Frommer, T. Lippert, B. Medeke, and K. Schilling, Eds., *Numerical Challenges in Lattice Quantum Chromodynamics*, ser. Lecture Notes in Computational Science and Engineering. Berlin/Heidelberg: Springer-Verlag, 2000, vol. 15.

[17] R. Byers, “Solving the algebraic Riccati equation with the matrix sign function,” *Linear Algebra Appl.*, vol. 85, pp. 267–279, 1987.

[18] *Oberwolfach model reduction benchmark collection*, IMTEK, `http://www.imtek.de/simulation/benchmark/`.

[19] P. Benner and E. Quintana-Ortí, “Solving stable generalized Lyapunov equations with the matrix sign function,” *Numer. Algorithms*, vol. 20, no. 1, pp. 75–100, 1999.

[20] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, “Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function,” in *Euro-Par 2009, Parallel Processing - Workshops*, ser. Lecture Notes in Computer Science, H.-X. Lin, M. Alexander, M. Forsell, A. Knupfer, R. Prodan, L. Sousa, and A. Streit, Eds. Springer-Verlag, 2009, no. 6043, pp. 132–139.

[21] P. Strazdins, “A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization,” Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia, Tech. Rep. TR-CS-98-07, 1998.

[22] Texas Advanced Computing Center, http://www.tacc.utexas.edu/tacc-software/gotoblas2.

[23] “Open BLAS,” Lab. of Parallel Software and Computational Science, ISCAS, 2013, http://xianyi.github.io/OpenBLAS/.

[24] “The CUDA development kit from SECO,” NVIDIA, 2013, http://www.nvidia.com/object/seco-dev-kit.html.