HPCLatAm 2013, pp. 158-166 (full paper)
Session: CPU and Multicore Architectures and Applications
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Trading Off Performance for Power-Energy in Dense Linear Algebra Operations

Peter Benner[1], Pablo Ezzatti[2], Enrique S. Quintana-Ortí[3], and Alfredo Remón[1]

[1] Max Planck Institute for Dynamics of Complex Technical Systems, D-39106 Magdeburg, Germany. {benner,remon}@mpi-magdeburg.mpg.de
[2] Instituto de Computación, Universidad de la República, 11.300-Montevideo, Uruguay. 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 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.

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 [6]. While the performance-power trade-off of HPC platforms has also enjoyed considerable advances in the past few years [4] —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 [7, 11, 12]. Even if the current progress pace of the performance-power ratio can be maintained (a factor of about 5× in the last 5 years [4]), 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 [1, 2]. 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) [16]. While this operation is key for the solution of important matrix equations arising in control theory via the matrix sign function [8, 19], 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 the applications of this particular operation. 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

Gauss-Jordan elimination is an appealing method for matrix inversion, with a computational cost and numerical properties analogous to those of traditional approaches based, e.g., on the LU factorization [16], but superior performance on a variety of architectures, from clusters [18] to general-purpose multicore processors and GPUs [8].

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 [9, 15]. A description of the unblocked version of GJE, called from inside the blocked routine, can be found in [18]; 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-performance evaluation can be extended to many other dense linear algebra kernels such as the

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 0×0

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.

solution of linear systems via the LU and Cholesky factorizations, and least-squares computations using the QR factorization [14], among others.

Additionally, explicit matrix inversion appears during the computation of the sign function of a matrix A using the simple Newton iteration [19]

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

and this particular matrix function plays an important role in the solution of spectral division problems [14, 10] as well as control theory applications (e.g., model reduction and optimal control) [17], and is the bottleneck computation in many lattice quantum chromodynamics computations [13].

3 Parallelization 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 the block size b is chosen so that $b \ll n$, 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 this algorithm primarily relies on using a highly tuned implementation of the matrix product and, under certain conditions, the reduction of the serial bottleneck that the factorization of $\hat{A}$ represents applying, e.g., a look-ahead strategy [20].

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 [21] or OpenBLAS [5]. Therefore, for the implementation of GJE in the multicore 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 product kernel `sgemm` in a recent version of Intel MKL.

Let us consider next the hybrid SECO development kit [3], which combines a quad-core NVIDIA Tegra3/ARM Cortex A9 processor and an NVIDIA Quadro 1000M GPU with 96 cores. As briefly described next, 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 A9 processor and the Quadro GPU.

In a previous work we introduced a CPU-GPU implementation of the GJE algorithm [8], and demonstrated the benefits of mapping each operation to the most convenient device: multicore processor or manycore accelerator. Here we apply a similar approach to obtain a tuned implementation of the GJE algorithm for the SECO platform, performing the matrix-matrix products on the manycore GPU, the factorization of $\hat{A}$ on the low-power CPU, and tuning the block size for these particular architectures and each problem size. Furthermore, we include look-ahead in the implementation, to overlap the factorization of the $(k+1)$-th panel with part of the updates performed at iteration k and keep a low communication overhead. Finally, the parallelism intrinsic to the matrix-matrix products that appear in algorithm GJE_BLK, and in the vector operations in algorithm GJE_UNB, are exploited employing multi-threaded implementations of the corresponding kernels from libraries CUBLAS and reference BLAS (using OpenMP), respectively.

4 Experimental Evaluation

In this section, we evaluate the matrix inversion routines on three target platforms: a state-of-the-art server equipped with two multicore Intel Xeon ("Nehalem") processors, an Intel Atom-based laptop, and an 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 routine 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 4.0) and the legacy implementation of BLAS[4] parallelized with OpenMP. (We note however, that the amount of computation that is performed in the cores of the 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.

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

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

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

	Xeon						Atom						SECO					
n	Time	GFLOPS	P_{avg}	P_{max}	E_{tot}	E_{net}	Time	GFLOPS	P_{avg}	P_{max}	E_{tot}	E_{net}	Time	GFLOPS	P_{avg}	P_{max}	E_{tot}	E_{net}
256	0.002	16.8	175.5	175.5	0.4	0.21	0.51	0.1	15.2	15.2	7.8	1.83	0.18	0.2	19.1	19.2	3.1	1.42
512	0.008	33.6	152.2	152.5	1.2	0.68	1.47	0.2	15.5	15.6	22.8	5.73	0.38	0.7	20.0	20.3	7.5	3.34
1,024	0.04	53.7	185.2	185.2	7.4	4.72	4.03	0.5	16.1	16.3	64.9	18.13	0.88	2.4	20.8	21.7	18.3	8.44
2,048	0.29	59.2	183.9	184.7	54.8	33.90	15.41	1.1	15.8	15.9	243.6	64.72	2.29	7.5	25.5	27.0	58.4	32.74
3,072	0.86	67.4	189.0	190.0	162.5	104.92	33.97	1.7	15.8	16.1	536.8	142.67	4.66	12.4	27.9	30.6	130.2	77.82
4,096	1.82	75.5	190.0	190.9	346.9	223.86	80.94	1.7	15.7	15.9	1,270.8	331.85	7.50	18.3	27.6	30.9	207.0	123.00
5,120	3.46	77.6	188.2	189.5	652.7	419.35	154.58	1.7	15.7	16.0	2,427.0	663.77	11.26	23.8	30.1	36.5	338.8	212.81
6,144	5.58	83.1	193.1	193.8	1,077.5	703.64	266.20	1.7	15.7	16.0	4,179.4	1,091.42	19.40	23.9	31.1	40.2	603.6	386.06
7,168	8.56	86.1	193.2	194.4	1,654.8	1,080.27	420.66	1.8	15.6	16.0	6,562.4	1,682.64	23.66	31.1	33.4	39.5	789.2	525.25
8,192	12.42	88.5	190.2	190.4	2,363.2	1,530.14	631.43	1.8	15.9	16.3	10,039.8	2,715.14	32.99	33.3	35.1	41.3	1,159.3	788.46

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).

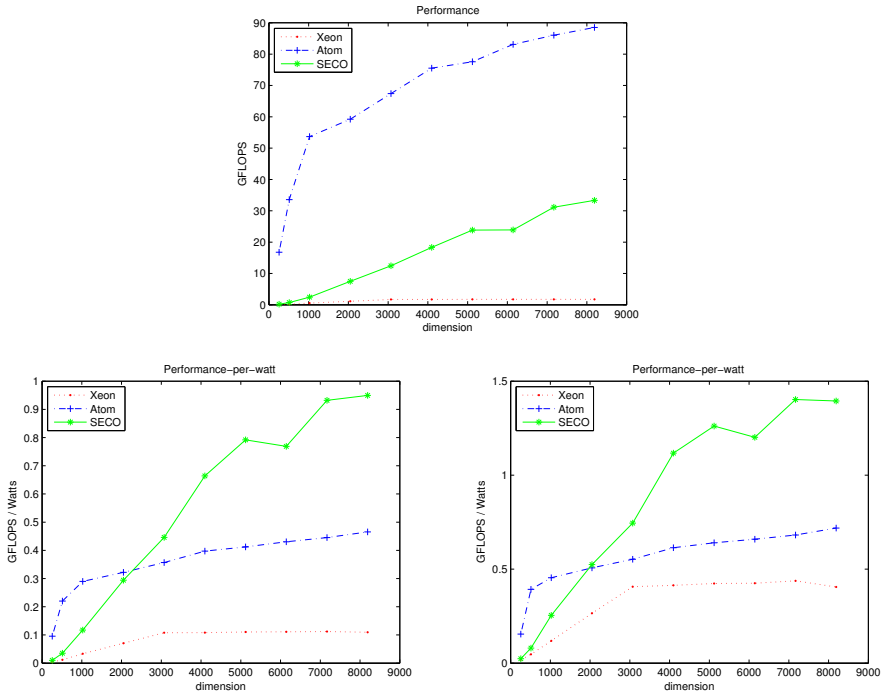

Fig. 2. Performance (top plot); and total and net performance-per-watt (bottom left and right plots, respectively).

Table 2 collects 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 Figure 2. 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 requires 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.

5 Concluding Remarks and Future Directions

We have investigated the trade-off between performance and power-energy for two low-power architectures, comparing those to a conventional general-purpose multicore processor. Our experimental evaluation using blocked routines for GJE matrix inversion shows that, for dense linear algebra operations which are rich in matrix-matrix products (or any other Level-3 BLAS kernel), 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.

Acknowledgments

The researcher from UJI was supported by the CICYT project TIN2008–06570-C04-01 and FEDER, and the EU Project FP7 318793 "EXA2GREEN".

References

1. CRESTA: collaborative research into Exascale systemware, tools and applications. `http://cresta-project.eu`.
2. The Mont Blanc project. `http://montblanc-project.eu`.
3. The CUDA development kit from SECO, 2013. `http://www.nvidia.com/object/seco-dev-kit.html`.
4. The Green500 list, 2013. Available at `http://www.green500.org`.
5. Open BLAS, 2013. `http://xianyi.github.io/OpenBLAS/`.
6. The top500 list, 2013. Available at `http://www.top500.org`.
7. Steve Ashby *et al.* The opportunities and challenges of Exascale computing. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, November 2010.
8. Peter Benner, Pablo Ezzatti, Enrique S. Quintana-Ortí, and Alfredo Remón. Matrix inversion on CPU-GPU platforms with applications in control theory. *Concurrency and Computation: Practice & Experience*, 25(8):1170–1182, 2013.
9. 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.
10. R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Linear Algebra Appl.*, 85:267–279, 1987.

11. J. Dongarra and *et al*. The international ExaScale software project roadmap. *Int. J. of High Performance Computing & Applications*, 25(1):3–60, 2011.
12. M. Duranton and *et al*. The HiPEAC vision for advanced computing in horizon 2020, 2013.
13. A. Frommer, T. Lippert, B. Medeke, and K. Schilling, editors. *Numerical Challenges in Lattice Quantum Chromodynamics*, volume 15 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin/Heidelberg, 2000.
14. G.H. Golub and C.F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.
15. 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.
16. N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
17. P.H. Petkov, N.D. Christov, and M.M. Konstantinov. *Computational Methods for Linear Control Systems*. Prentice-Hall, Hertfordshire, UK, 1991.
18. 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.
19. J.D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32:677–687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).
20. P. Strazdins. A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization. Technical Report TR-CS-98-07, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia, 1998.
21. Texas Advanced Computing Center, `http://www.tacc.utexas.edu/tacc-software/gotoblas2`.