Frequency Scaling and Energy Efficiency regarding the Gauss-Jordan Elimination Scheme on OpenPower 8

Martin Köhler[1] Jens Saak[2]

The Gauss-Jordan Elimination scheme is an alternative to the LU decomposition for solving linear systems or computing the inverse of a matrix. We develop a multi-GPU aware implementation of this algorithm on an OpenPOWER 8 system with application to the Matrix Sign Function. Thereby, we analyze the influence to the CPU clock frequency scaling on the overall energy consumption. The results show possible energy saving of 14.2% without a noteworthy increase of the runtime.

1 Introduction

Beside solving a general linear system $Ax = b$ using the LU decomposition there are a few applications, like Newton's Method for computing the Matrix Sign Function [9, 4, 3, 2] or the Polar Decomposition [7], that require the explicit inverse A^{-1}. In this case, either the three step scheme implemented in LAPACK [1] or the Gauss-Jordan Elimination [11] can be used to obtain A^{-1}. The LAPACK approach first computes the LU decomposition of the matrix A, then inverts the upper triangular matrix U ad finally solves $LA^{-1} = U^{-1}$. This procedure takes $2m^3$ flops if m is the order of the matrix. Furthermore, this approach causes that three routines need to be regarded during the optimization of the implementation. Moreover, the two steps working on triangular matrices are complicated to parallelize by their nature. On the other hand, we have the Gauss-Jordan Elimination computing the inverse A^{-1} by rearranging the three step LAPACK scheme [11]. The resulting algorithm is free of any operations dealing with triangular matrices and mainly consists of general matrix-matrix products. This makes the algorithm preferable on massively parallel architectures, like multi-core or accelerator based systems. Furthermore, one can show that the Gauss-Jordan Elimination reduces the number of memory accesses [10] and by using general matrix-matrix multiplies the data locality for the single operations of the algorithm is improved.

In our contribution, we focus on the efficient implementation of the Gauss-Jordan matrix inversion on the OpenPOWER 8 platform. Besides two 10-core IBM POWER8 CPUs the test system is equipped with two Nvidia Tesla P100 accelerators with NVLink interconnect and 256 GB DDR4 memory. The system can be seen as predecessor of the compute nodes in the upcoming super computer “Summit” at Oak Ridge National Laboratory[3] that will use the IBM POWER9 platform together with the next generation of Nvidia's accelerators named “Volta”. The most important differences to previous GPU accelerated systems and the named POWER8 system are:

[1]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, D-39106 Magdeburg ,
koehlerm@mpi-magdeburg.mpg.de

[2]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, D-39106 Magdeburg ,
saak@mpi-magdeburg.mpg.de

[3]`https://www.olcf.ornl.gov/summit/` – Accessed March 20th, 2017

- The usage of NVLink as interconnect between CPU and GPU. This increases the transfer rate between their memories by a factor of 2 to 3 in comparison to the latest PCI-Express bus. In this way, data transfers between the CPU and the GPU are cheaper (with respect to runtime) than on older systems.
- The ratio of the peak performances between both CPUs and GPUs is a factor of 20. This is a increase by a factor of 5 if we compare it to an older system, like the 16 core Intel Xeon Haswell with two Nvidia K20 accelerators, where we have done our previous work on [10].
- While keeping the energy consumption for the GPUs in the same order of magnitude as for the old K20 GPUs the energy consumption of the POWER8 CPUs is much higher in idle state, as well as in full operation mode, compared to the Intel Haswell Xeon CPUs with a similar peak performance.

The last point makes the difference from the energy point of view. For this reason we want to focus on reducing the power consumption of the CPUs by changing their clock frequency and/or changing the CPU frequency governors that control the automatic adjustment of the CPU clock frequency.

The contribution is organized as follows. First we recall the Gauss-Jordan Elimination approach and its efficient implementation. Later on we use the Matrix Sign Function as an application for our matrix inversion code. Finally, we show the influence of changing the CPU's clock frequency to the time-to-solution, the energy-to-solution and the Energy-Delay-Product (EDP) [5, 8]. Thereby, the EDP can be used to decide whether it is worth to save the energy or to save runtime from an economic point of view.

2 Gauss-Jordan Elimination

The Gauss-Jordan Elimination scheme can be interpreted as an alternative representation of the LU decomposition with reordered operations [11]. Therefore, we will only recall the basics to obtain a blocked algorithm here. First, we consider the Gauss-Transform $G_i \in \mathbb{R}^{m\times m}$:

$$G_i = \begin{bmatrix} 1 & & & -\frac{a_{1i}}{a_{ii}} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & -\frac{a_{(i-1)i}}{a_{ii}} & & & \\ & & & \frac{1}{a_{ii}} & & & \\ & & & -\frac{a_{(i+1)i}}{a_{ii}} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -\frac{a_{mi}}{a_{ii}} & & & 1 \end{bmatrix}, \qquad (1)$$

which introduces zeros in the i-th column of a matrix G_iA and sets the i-th row in this column to one. By applying G_i to a permuted matrix P_iA, where P_i exchanges row i with row k, $k := \operatorname{argmax}_{k\geq i} |a_{ki}|$, one can use pivoting as in the LU decomposition [6]. We use $\widetilde{G_i} := G_iP_i$ as pivoted Gauss-Transform (PGT). The application of m Gauss-Transforms or pivoted Gauss-Transforms from the left to a matrix $A \in \mathbb{R}^{m\times m}$ yields its inverse:

$$\underbrace{\widetilde{G_m}\cdots\widetilde{G_1}}_{A^{-1}}A = I. \qquad (2)$$

Using the fact that $\widetilde{G_i}$ eliminates the i-th column, except of the 1 on the diagonal, one can store the i-th column of $\widetilde{G_i}$ in the i-th column of A, after it is applied, to obtain an in-place algorithm. The block algorithm (with a block size of N_B) is obtained by the following considerations. Without loss of generality, we neglect the pivoting matrix P_i. The application of a Gauss-Transform G_i can be written as a rank-1 update with an additional operation:

$$A \leftarrow A - \frac{1}{a_{ii}}\left(a_{1i}, \cdots, a_{(i-1)i}, 0, a_{(i+1)i}, \ldots, a_{mi}\right)^T A_{i,\cdot}$$
$$A_{i,\cdot} := \frac{1}{a_{ii}} A_{i,\cdot}. \tag{3}$$

By partitioning the matrix A into

$$A \leftarrow \left[\begin{array}{c|c|c} A_{11} & A_{12} & A_{13} \\ \hline A_{21} & A_{22} & A_{23} \\ \hline A_{31} & A_{32} & A_{33} \end{array}\right], \tag{4}$$

where A_{22} is of dimension $N_B \times N_B$ we obtain the block formulation of the rank-1 update (3) as:

$$A \leftarrow \left[\begin{array}{c|c|c} A_{11} & 0 & A_{13} \\ \hline 0 & 0 & 0 \\ \hline A_{31} & 0 & A_{33} \end{array}\right] + \underbrace{\left[\begin{array}{c} -A_{12}A_{22}^{-1} \\ \hline A_{22}^{-1} \\ \hline -A_{32}A_{22}^{-1} \end{array}\right]}_{H} \left[A_{21} \quad I_{N_B} \quad A_{23}\right]. \tag{5}$$

Thereby, the matrix H_k can be regarded as the (partial) inverse of the block column $B := \left[A_{12}^T \quad A_{22}^T \quad A_{32}^T\right]^T$ and can be computed by applying Gauss-Transforms to B as well.

In order to avoid a direct fallback from the rank-N_B updates from (5) to the rank-1 updates from (3) in the computation of H we use the same strategy as in LAPACK since version 3.6.0. There the locality improved LU decomposition was introduced [12] to avoid the direct level-2 BLAS fallback. The key idea is to apply the blocked algorithm again to H but with a block size of $\frac{N_B}{2}$ recursively until the block size is reduced to 1 and the final work consists only in updating a single column. As in the case of the LU decomposition this increases the data locality of the operations and allows to use more level-3 BLAS operations than if one uses the rank-1 formulation immediately.

Taking the GPUs into account, we can easily create a hybrid CPU-GPU version of our algorithm. The rank-N_B update is well suited for the GPU because, on the one hand, the general matrix-matrix product is one of the best optimize routines for the GPUs and, on the other hand, using the block cyclic distribution scheme this can be easily parallelized across multiple GPUs. Asynchronous operations and lookahead are also easy to implement by splitting the update with H and A_{23} into two parts. The first part affects the leading N_B columns of A_{23} and results in the input data for the computation of the next matrix H. Afterwards the GPUs can handle the remaining part of A_{23} while the CPU prepares the next matrix H.

Newton's Method for the Matrix Sign Function The Matrix Sign Function $X := \operatorname{sign}(A)$, e.g. [9], is the generalization of the sign of a scalar number to the matrix case.

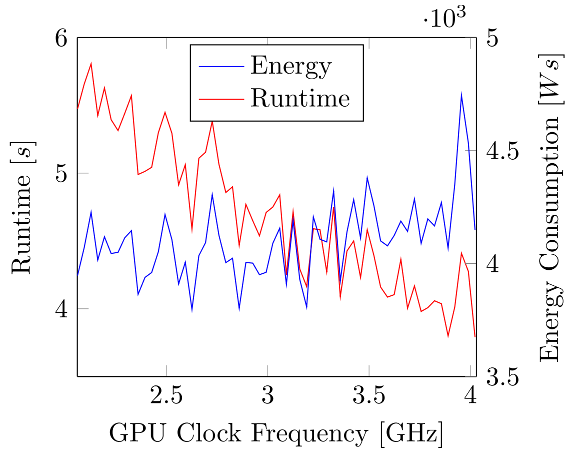

Figure 1: $m = 20\,480$

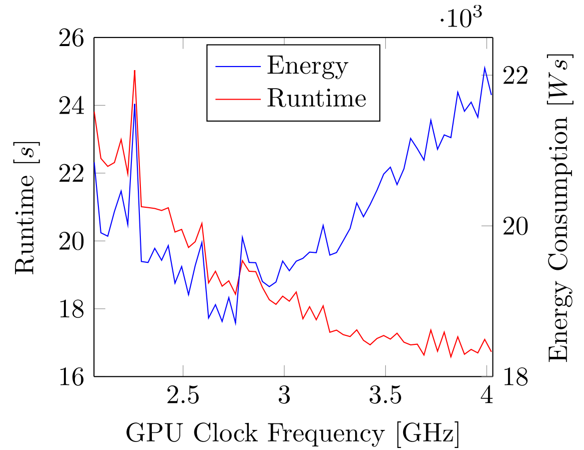

Figure 2: $m = 40\,960$

One way to compute it is to use one of its defining properties, $X^2 = I$, and apply the Newton iteration with the initial value A. This yields the following iteration:

$$X_{k+1} := \frac{1}{2}\left(X_k + X_k^{-1}\right) \quad X_0 = A. \tag{6}$$

On convergence $\text{sign}(A) = X_\infty$ holds. In practical implementations a scaling factor c_k is introduced to accelerate the convergence [3]. For ease of presentation we do not regard this here.

Beside the inversion of the matrix X_k we only need a matrix valued scale and a matrix valued add operation. Having in mind that this operation is bandwidth bound we refer to the high bandwidths of the system here. Reaching a practical GPU–memory bandwidth of 500 GB/s one can still scale a matrix filling up the device memory 31.25 times per second. With a memory bandwidth of 230GB/s bandwidth bound operations can also be performed on the CPUs.

If pivoting is enabled during the inversion of X_k, the Gauss-Jordan-Elimination scheme computes the inverse $\widetilde{X_k}^{-1}$ of PX_k, where P consists of all permutations used during the pivoting. Therefore, we have to add a column permutation to $\widetilde{X_k}^{-1}$ to obtain $X_k^{-1} := \widetilde{X_k}^{-1} P^T$. As long as only one GPU is used this can easily be performed on the GPU. If the matrix is distributed across several GPUs this becomes a communication intensive procedure. Due to the limited bandwidth between two GPUs (40GB/s) the irregular movement of the columns will slow down the whole procedure. In this case we move the whole matrix X_k^{-1} to the host memory again and use a parallel permutation algorithm there. Finally, we distribute the matrix to devices again an create an on device copy of the matrix to have X_{k+1} already available for the next iteration. Changing the devices' data layout to a cyclic block row representation we can easily permute the columns but the problem of the distributed data moves to the row permutation inside the Gauss-Jordan Elimination, which causes the same problems there.

3 Results

We run all experiments on the OpenPOWER 8 system (IBM POWER System 822LC) running CentOS 7.3 (with a custom build Linux 4.8 kernel) mentioned in the Introduction. The software ecosystem consists of Nvidia CUDA 8.0, IBM XLC 13.1.5, IBM XLF 15.1.5,

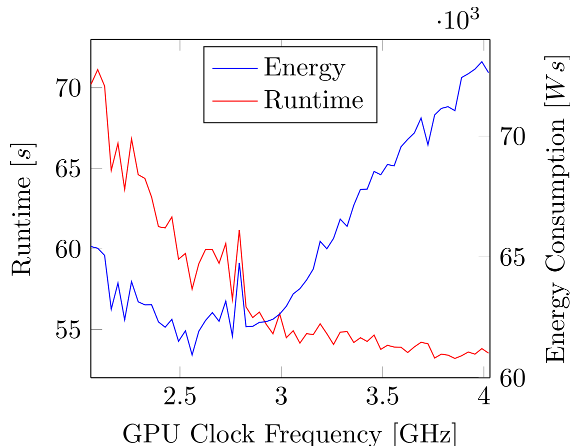

Figure 3: $m = 61\,440$

Dimension	20 480	40 960	61 440
EDP(1)	3.890	3.225	2.959
EDP(2)	3.890	3.225	2.959
EDP(3)	3.890	3.690	3.092

Table 1: CPU Clock Frequency (in $[GHz]$) minimizing the EDP(w) with weights $w = 1, 2, 3$.

and IBM ESSL 5.5 as host BLAS/LAPACK library. The CPUs clock frequency can be adjusted in a range from 2.061GHz to 4.023GHz by steps of $\approx 33MHz$, where for high clock frequencies above $3.823GHz$ the frequencies are reduced automatically due to power supply and thermal issues. Nevertheless, we force the CPUs to reach these frequencies by using the *userspace* performance governor of the *cpufreq* mechanism of the Linux kernel.

The main goal of the experiments is to check whether it is worth to spent more energy by enforcing a high CPU clock frequency, or where optimal points between an increase of the runtime and the saved energy are. The optimality is checked with respect to the Energy-Delay-Product (EDP) [5, 8] defined by:

$$\text{EDP}(w) = E \cdot T^w, \tag{7}$$

where E is the energy-to-solution, T is the time-to-solution, and w a weight factor to penalize the time. The optimal block size N_B for the Gauss-Jordan Elimination was determined in previous experiments. Here, we restrict to the matrix inversion since this is the most challenging operation during the computation of the Matrix Sign Function. We use random matrices A of dimension 20 480, 40 960, 61 440.

Figures 1 to 3 show the runtime and the energy consumption of the Gauss-Jordan Elimination. For the smallest case ($m = 20\,480$) we observe that we have an approximately linear decrease of the runtime coupled with a slowly increasing energy consumption. In this case, one can still choose nearly maximum CPU clock frequency and still obtain an economically optimal execution. This coincides with the suggestion of the EDP from Table 1 to chose a frequency next to the maximum. For larger problems we see that beginning with a clock frequency of $\approx 2.9\,GHz$ we have a steep increase of the energy consumption while only obtaining a small speed up in the runtime. On the other hand, regarding the largest example ($m = 61\,440$) the increase of the clock frequency from $2.959\,GHz$ to $4.023\,GHz$ costs 14.2% more energy while only accelerating the process by 2.2%, while switching from the clock frequency from $2.016\,GHz$ to $2.959\,GHz$ we only need 5% more energy and accelerate the algorithm by 26.9%. The EDP from Table 1 suggest exactly this clock frequency for $w = 1$ and $w = 2$. Even if we increase the impact of the runtime to $w = 3$ the EDP only suggests an increase of the clock frequency by 4 steps to $3.092\,GHz$. Finally, we see that for an increasing problem dimension the influence of the pure CPU power decreases and even for higher weights of the runtime the EDP suggests a moderate clock frequency in order to obtain an economically acceptable solution.

4 Conclusions

We have shown that for the Gauss-Jordan Elimination on the OpenPOWER 8 platform one can save a remarkable amount of energy by choosing a proper clock frequency for the CPUs without causing a noteworthy increase of the runtime. This extended abstract only covers the case of fixed CPU clock frequencies but the hardware (supported by the drivers from the Linux kernel) supports automatic adjustment with respect to several policies. These so called *cpufreq* governors will also be taken into account in the final contribution as well as the overall process of the Newton iteration (6).

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, third ed., 1999.

[2] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, *Matrix inversion on CPU-GPU platforms with applications in control theory*, Concurrency and Computing: Practice and Experience, 25 (2013), pp. 1170–1182.

[3] R. Byers, C. He, and V. Mehrmann, *The matrix sign function method and the computation of invariant subspaces*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 615–632.

[4] E. D. Denman and A. N. Beavers, *The matrix sign function and computations in systems*, Appl. Math. Comput., 2 (1976), pp. 63–94.

[5] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, *Analyzing the energy-time trade-off in high-performance computing applications*, IEEE Trans. Parallel Distrib. Syst., 18 (2007), pp. 835–848.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, fourth ed., 2013.

[7] N. J. Higham, *Computing the polar decomposition—with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160–1174.

[8] M. Horowitz, T. Indermaur, and R. Gonzalez, *Low-power digital design*, Proceedings of 1994 IEEE Symposium on Low Power Electronics, (1994), pp. 8–11.

[9] C. Kenney and A. J. Laub, *The matrix sign function*, IEEE Trans. Automat. Control, 40 (1995), pp. 1330–1348.

[10] M. Köhler, C. Penke, J. Saak, and P. Ezzatti, *Energy-aware solution of linear systems with many right hand sides*, Comput. Sci. Res. Dev., (2016). accepted for publication.

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

[12] S. Toledo, *Locality of reference in LU decomposition with partial pivoting*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 1065–1081.