Parallel 2d-Wavelet Transform on the Cell/B.E. for Fast Calculation of Coulomb Potentials

A. Schiller, G. Sutmann

published in

From Computational Biophysics to Systems Biology (CBSB08),
Proceedings of the NIC Workshop 2008,
Ulrich H. E. Hansmann, Jan H. Meinke, Sandipan Mohanty,
Walter Nadler, Olav Zimmermann (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **40**, ISBN 978-3-9810843-6-8, pp. 385-388, 2008.

Parallel 2d-Wavelet Transform on the Cell/B.E. for Fast Calculation of Coulomb Potentials

Annika Schiller and Godehard Sutmann

Institute for Advanced Simulation, Jülich Supercomputing Centre,
Research Centre Jülich, 52425 Jülich, Germany
E-mail: {*a.schiller, g.sutmann*}*@fz-juelich.de*

1 Introduction

Long range interactions between particles often play an important role in biomolecular simulations in order to describe the structure and dynamics of particles correctly. The calculation of this type of interaction often limits the time and length scale of a simulation, since it scales as $\mathcal{O}(N^2)$, where N is the number of particles in the system. In order to overcome this limitation, different types of fast algorithms of order $\mathcal{O}(N \log N)$ or $\mathcal{O}(N)$ were developed (for an overview see e.g. Ref. 1, 2). One of this type of algorithms is based on a Wavelet transform technique[3]. A computationally intensive part consists in preparing the 2d-Wavelet transform of inverse distances between fixed grid points in space, onto which particle properties are transferred. Since the grid is static, this computational intense part has only to be performed once. Due to memory requirements and performance, it is desirable to perform these calculations on a scalable computer architecture. To this end the Cell Broadband Engine (Cell/B.E.) heterogeneous multicore processor was chosen to explore its capabilities and high potential in performance. The processor characteristics include multiple heterogeneous execution units, SIMD processing engines, fast local store and a software managed cache. Applications can achieve a performance which is close to the theoretical peak performance if specific features are respected.

In the present work an implementation of the fast 2d-Wavelet transform, realized via a triple matrix multiplication, was developed using the native programming API of the IBM Software Development Kit (CellSDK[4]). In this implementation the architectural requirements of the Cell processor are taken into account and Cell specific optimizations are applied where practical. Therewith, the difficulties and problems in porting code to Cell/B.E. and using sparse linear algebra operations on the Cell processor are assessed. Finally the results of the native implementation are discussed and compared to the results of the same algorithm implemented via the Cell Superscalar framework (CellSs[5]), a high-level portable programming model.

2 Method

The method of calculating Coulomb potentials with the help of Wavelets was described in Ref. 3. Here, we concentrate on the description on how to calculate the triple-matrix-multiply

$$\tilde{\mathsf{A}}_t \quad = \mathbb{T}\{\mathsf{W}_l \, \mathsf{A} \, \mathsf{W}_l^T ; \, t\} \tag{1}$$

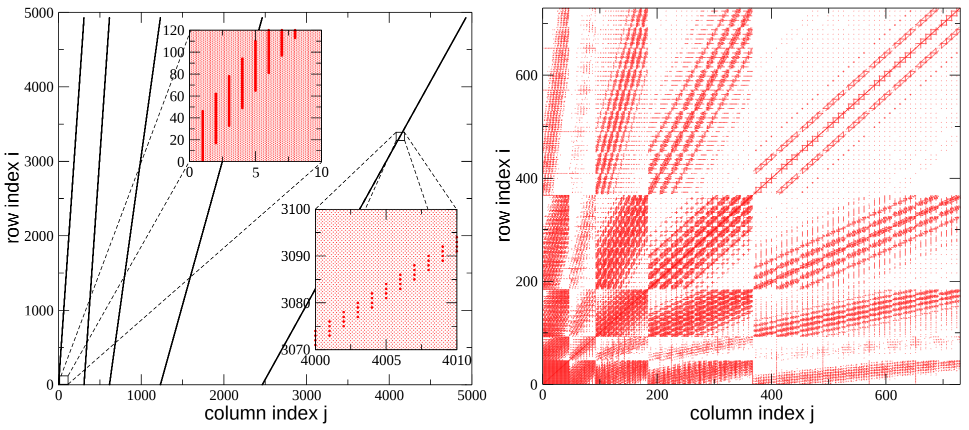

Figure 1. Structure of matrices $\mathbf{W}_l$ (left) for a Daubechies-4 Wavelet-basis (details are shown in the insets) and $\tilde{\mathbf{A}}_t$ (right) with threshold value $t = 0.01$ and compression rate $\chi = 81\%$.

where the matrix $\mathbf{W}_l$ is the resulting Wavelet transform matrix on level l, which is sparse and contains both the Wavelet and scaling coefficients. The structure of this matrix is displayed in Fig.1. On the other hand the matrix $\mathbf{A} \in \mathbb{R}^{N_g \times N_g}$ is dense, as it contains the inverse distances between equi-spaced grid points, where N_g is the number of grid points in the system. Since $N_g \propto N$, the dimension of the matrix may become rather large, which requires an economic memory management. Finally, $\tilde{\mathbf{A}}_t$ is a thresholded Wavelet transform of $\mathbf{A}$, which results from the threshold operation $\mathbb{T}\{.\,;t\}$, where all absolute values below a given value t are set to zero.

In order to apply the method also to larger problem sizes, the Wavelet matrix $\mathbf{W}_l$ is partitioned and transferred in parts to the SPU. Information about the matrix is stored in Compressed Sparse Row (CSR) format[6], for which four arrays are introduced: (i) the elements of the matrix, `j_elem`; (ii) column index of n-th element, `j_index`; (iii) offset value, `nnn_off`, in order to address the first element of row i; (iv) number of elements of row i, `nj_w`.

The partitioning of $\mathbf{W}_l$ does not guarantee that every block of data contains complete rows. Transferring incomplete rows to the SPU would, however, increase the complexity of the computation algorithm. A possible method to avoid splitting of rows consists in padding. Therefore, in order to align the array `j_elem` a second array has to be allocated where rows of $\mathbf{W}_l$ are grouped into 16 kB blocks. If one row is split by a block border the complete row is put into the next block and the current block is filled by padding. The handling of array `j_index` proceeds analogously. The rows of $\mathbf{W}_l$ do not always contain the same number of non-zeros. Therefore, an array is introduced which stores the number of rows in each block. Using this block information the corresponding elements of arrays `nnn_off` and `nj_w` can be loaded. Note, that these arrays are not aligned. Therefore it is necessary to expand the needed part to an aligned extract which contains all the data actually required.

In the present implementation, the matrix $\tilde{\mathbf{A}}_t$ is calculated column by column in two

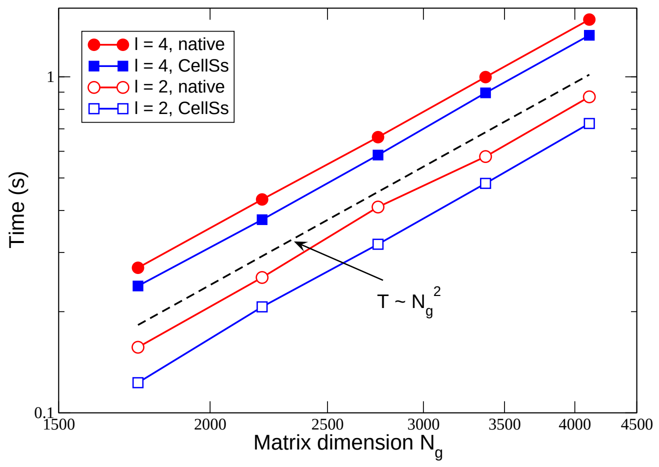

Figure 2. Timings for a native implementation of the algorithm and for the CellSs implementation for Wavelet transforms of different level l. CellSs outperforms the native implementation, since features like double-buffering are not considered in the native version.

steps as follows[7] (for different implementations cf. Ref. 8)

$$\tilde{\mathsf{A}} = \mathsf{W}_l \, \mathsf{A} \, \mathsf{W}_l^T \quad \Rightarrow \quad \tilde{\boldsymbol{a}}_q = \mathsf{W}_l \times (\mathsf{A} \times \boldsymbol{w}_q^T) \tag{2}$$

In this equation $\tilde{\boldsymbol{a}}_q$ is the q-th column of the result matrix $\tilde{\mathsf{A}}$ and $\boldsymbol{w}_q^T$ is the q-th column of the transposed Wavelet matrix W_l^T. In the first step the intermediate result $\boldsymbol{b}_q = \mathsf{A} \times \boldsymbol{w}_q^T$ is calculated. In the second step the q-th column of $\tilde{\mathsf{A}}$ is calculated via $\tilde{\boldsymbol{a}}_q = \mathsf{W}_l \times \boldsymbol{b}_q$. These steps have to be repeated for each of the N columns. In the implementation of this algorithm the intermediate result $\boldsymbol{b}_q$ is buffered in a temporary array on the SPU. To decrease DMA transfers this intermediate array is not transferred back to the PPU.

The two main aspects which are to be considered for this application are the amount of data and the number of floating point operations on the SPU. Since for realistic applications $N_g \approx 32^3$, the size of A easily exceeds the memory capacities of the SPU (256 kB). However, considering the inverse distance matrix in detail, reveals that full storage of elements contains a lot of redundency. If $N_g = n^3$ there are $N_g(N_g - 1)$ distances between grid points, of which only $n(n+1)(n+2)/6 - 1$ are different. Therefore, only non-redundant data are calculated on the PPU and transferred to the SPU, where an addressing of matrix elements is performed in order to map a three-dimensional grid onto a two-dimensional matrix. That means that it is enough to store $\mathcal{O}(N_g)$ values for the kernel of A instead of $\mathcal{O}(N_g^2)$ because of redundant entries. The Wavelet matrix W_l is blocked and loaded blockwise onto the SPUs, where $\tilde{\mathsf{A}}$ is then calculated in two steps column by column and then thresholded. The intermediate result is buffered on the SPU and not transferred back to the PPU.

3 Results

A native programming approach was compared to one, where CellSs was used. Fig. 2 shows the result of this comparison for Wavelet transforms of different sizes N_g and levels l for a Haar-Wavelet transform.

As a first result it is found that the scaling of the algorithm is $\mathcal{O}(N_g^2)$, as it could be expected for a sparse-dense-sparse matrix multiply. This result, however, shows that (i) the algorithm is efficiently implemented and (ii) with increasing problem size the PPU-SPU-communication is not a limiting factor. This leads to optimism for increasing problem sizes.

As a second result it is seen that the CellSs based implementation outperforms the native implementation in all cases by about 10-20%. This prooves that CellSs is not only a very nice tool to ease Cell programming, but that it is also able to produce high performance code. The reason for the difference of the two approaches may be found in neglecting SIMD vectorization and features like double buffering in the native implementation. Since the complexity of the code already was large, these features were postponed in first instance. The result shows that CellSs leads to very good performance while reducing the programming effort at the same time.

References

1. P. Gibbon and G. Sutmann. Long range interactions in many-particle simulation. In J. Grotendorst, D. Marx, and A. Muramatsu, editors, *Quantum simulations of many-body systems: from theory to algorithms*, volume 10, pages 467–506, Jülich, 2001. John von Neumann Institute for Computing.
2. M. Griebel, S. Knapek, and G. Zumbusch. *Numerical Simulation in Molecular Dynamics*. Springer, Berlin, 2007.
3. G. Sutmann and S. Wädow. A Fast Wavelet Based Evaluation of Coulomb Potentials in Molecular Systems. In U.H.E. Hansmann, editor, *From Computational Biophysics to Systems Biology 2006*, volume 34, pages 185–189, Jülich, 2006. John von Neumann Institute for Computing.
4. IBM, *Cell Broadband Engine: Programming Tutorial*, March 2007, Version 2.1.
5. `http://www-03.ibm.com/technology/cell/swlib.html`.
6. Y. Saad. SPARSKIT: A Basic Tool for Sparse Matrix Computation. Technical Report CSRD TR 1029, University of Illinois, 1990.
7. F. Marino, V. Piuri, and E.E. Swartzlander. A parallel implementation of the 2d discrete Wavelet transform without interprocessor communications. *IEEE Trans. Sign. Proc.*, 47:3179–3184, 1999.
8. L. Yang, A. Schiller, G. Sutmann, B.J.N. Wylie, R. Altenfeld, and F. Wolf. Comparison of CellSs and native programming with a Jacobi solver and triple-matrix-multiply on Cell/B.E. Proc. of the *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing* Para 2008 (submitted).