Comparison of CellSs and native programming with a Jacobi solver and triple-matrix-multiply on Cell/B.E.

Liang Yang[1,3], Annika Schiller[1], Godehard Sutmann[1], Brian J. N. Wylie[1]
Ralph Altenfeld[1], and Felix Wolf[1,2]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany
[2] Department of Computer Science, RWTH Aachen University, Germany
[3] Xi'an Microelectronics Technology Institute, China
yungicn@gmail.com
{a.schiller,b.wylie,g.sutmann,r.altenfeld,f.wolf}@fz-juelich.de

Abstract. The Cell Superscalar framework (CellSs), from Barcelona Supercomputing Centre, offers a high-level portable programming model to port, parallelise and tune applications on Cell Broadband Engine. Via implementation of a Jacobi solver and a triple-matrix-multiply (TMM) kernel from a wavelet-based evaluation of Coulomb potentials in molecular systems, using the native programming API of the IBM Software Development Kit (CellSDK) and the latest version of CellSs, ease of programming and resulting performance are assessed.
CellSs is found to be a convenient and effective vehicle for achieving performant parallelisations. The relative simplicity of the CellSs programming model and its efficient automatic implementation of task scheduling and internal data management were particularly helpful during the development of a novel TMM algorithm with more efficient memory usage, which was necessary to realise the application within the limited SPE memory of Cell/B.E..

Key words: Programming models & tool environments; application parallelisation & optimisation; performance analysis & evaluation.

1 Introduction

The high potential performance of the Cell Broadband Engine (Cell/B.E.) derives from a broad spectrum of capabilities. Processor characteristics include multiple heterogeneous execution units, SIMD processing engines, fast local store and a software managed cache. Applications can achieve nearly-maximum theoretical performance if specific features are respected [1]. Exploiting the full potential of Cell/B.E., however, is challenging for programmers who are trying to port their applications. It is necessary to initiate parallel tasks by calling special functions supplied in the IBM Software Development Kit (CellSDK) SPE runtime management library and data transfers to and from the limited local memory of the SPEs have to be managed explicitly via DMA function calls. This

means that to port an application to Cell/B.E. a custom parallel program has to be written which can only run on that processor. To facilitate the porting of native code to Cell/B.E., Barcelona Supercomputing Centre is developing CellSs, which proposes a portable programming model for multi-core processors [2].

In this paper a Jacobi solver and the TMM kernel from a wavelet-based evaluation of Coulomb potentials in molecular systems are ported to Cell/B.E. via CellSs, and the results compared to implementations created using the CellSDK native programming API.

2 Programming with CellSs

The Cell Superscalar framework (CellSs) is based on a source to source compiler and runtime library. By inserting simple annotations (pragmas) into the application code, CellSs generates code for PPE and SPEs, which is then compiled by the native compiler. CellSs requires that the application is composed of coarse grain functions and that these functions do not have collateral effects (i.e., only local variables and parameters are accessed). With CellSs, the annotation before a coarse grain function (task) does not indicate a parallel region like OpenMP, but simply indicates the direction of the parameters of this function (input/inout/output). The CellSs runtime system builds a data dependency graph by collecting the information about these parameters and schedules independent tasks to different SPEs concurrently. All data transfers required for computations on the SPEs are also handled automatically. Nevertheless, the programmer should calculate carefully if the data will fit in the SPEs' local memory and also consider data alignment requirements. Furthermore, CellSs includes an execution trace generation capability so that the programmer can examine how long is taken for DMA transfers, SPE task computations, waiting on data, user code on the PPE, etc., to find application imbalances and inefficiencies.

3 3D Jacobi solver

The Jacobi solver computes the solution for systems of linear equations, which are used in a wide range of scientific applications. For a 3-dimensional problem, the Jacobi solver basically updates each data point based on its six neighbors. Equation 1 shows the update scheme considered in this work.

$$A^{n+1}_{i,j,k} = \alpha(f_{i,j,k}+A^{n}_{i-1,j,k}+A^{n}_{i+1,j,k}+A^{n}_{i,j-1,k}+A^{n}_{i,j+1,k}+A^{n}_{i,j,k-1}+A^{n}_{i,j,k+1}) \quad (1)$$

Due to the limited size of the SPE's local memory, the whole matrix must be partitioned into smaller blocks. When the data of one block is sent to an SPE, the six neighbour planes are grouped together and sent with it. Several schemes have been investigated, varying how the source operator data is blocked and how ghost planes are updated. The best speedup was found when five SPEs are used concurrently, and CellSs produced code which significantly outperformed the comparable native implementations with CellSDK, as shown on the left in Figure 1.

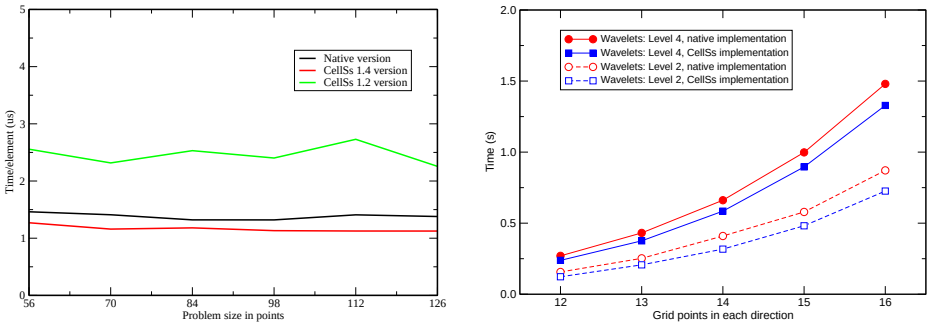

Fig. 1. Execution time performance comparison between native CellSDK and CellSs implementations of a Jacobi solver (left) and triple-matrix-multiply (right).

4 Triple-matrix-multiply

This application is one part in the kernel of a program which is used to perform a wavelet based evaluation of Coulomb potentials in molecular systems [3]. The following triple-matrix-multiply (TMM) has to be calculated.

$$\tilde{A} \;= \mathcal{W}\, A\, \mathcal{W}^T \;\Rightarrow\; \tilde{A}_{ij} = \sum_{k,l} \mathcal{W}_{ik} \cdot A_{kl} \cdot \mathcal{W}_{jl} \tag{2}$$

Wavelet matrix $\mathcal{W}$ is sparse and has a special banded structure, therefore it is stored via Compressed Sparse Row (CSR) format in this application. The inverse-distance matrix A is dense and symmetric with dimension $N \times N$, where $N = np \cdot np \cdot np$ is the number of grid points and np the number of grid points in one dimension. Elements of matrix A can be calculated as follows.

$$A_{ij} = \begin{cases} \frac{1}{|r_i - r_j|} = \frac{1}{a \cdot \sqrt{(ix-jx)^2+(iy-jy)^2+(iz-jz)^2}} & , \; i \neq j \\ 0 & , \; i = j \end{cases} \tag{3}$$

where

$$ix = i \;\%\; np \qquad iy = (i \;\%(np \cdot np))/np \qquad iz = i/(np \cdot np) \tag{4}$$

In equation 3, a is a constant, i and j are the indices of the elements in A, and ix, iy, iz are the corresponding coordinates of matrix index i in the three-dimensional $np \cdot np \cdot np$ grid. They can be calculated via equation 4.

Three different algorithms were implemented, which consider different architectural requirements. The first approach considers the storage requirement for matrix A which can become very large. Since the memory of the PPE is limited to 2GB and SPE local store is limited to 256kB, the amount of data is an important aspect. To minimise storage space and DMA transfers, the elements of matrix A can be calculated on the SPE when they are needed (using equation 3). This approach is efficient in its memory usage but computationally inefficient because each element of A is calculated multiple times.

To avoid redundant re-calculation of elements of matrix A, a second approach was to calculate the complete matrix A only once on the PPE and load it line by line onto the SPEs, however, this approach needs so much memory that it cannot be used for larger problem sizes.

To decrease memory usage and redundant computation, a novel approach which is efficient on both aspects was developed, which exploits the fact that the three-dimensional grid of the simulated system has to be mapped onto the two-dimensional matrix A. There is therefore one row in the inverse-distance matrix A for each point in the three-dimensional grid which corresponds to the relationship between this point and other points in the grid. In other words, matrix A contains a lot of redundant information, so that it is enough to store only $O(N)$ values for the kernel of A instead of $O(N^2)$ values for the whole matrix A. The correct distance can be calculated by mapping the two-dimensional addressing of A back onto the three-dimensional addressing of the system's grid as shown in equation 4.

The third approach turned out to be the most efficient. In the right of Figure 1, the time for TMM calculated using the third approach for two problem sizes is compared between the native and CellSs implementations, showing that the CellSs code is consistently 10–20% faster than the native CellSDK version.

5 Conclusions

A Jacobi solver and triple-matrix-multiply were implemented with different algorithms to evaluate parallelisation with CellSs. Although current limitations of CellSs (v1.4) present certain difficulties (particularly with respect to support for applications written in C++ or Fortran), it provides programmers a flexible programming model with an adaptive parallelism level that provided acceptable performance and portable code, while considerably simplifying Cell/B.E. programming. In particular, CellSs automatically implements efficient double-buffering of data transfers, which are complicated to implement natively with the CellSDK. A novel algorithm was thereby developed for triple-matrix-multiply, which is efficient in both storage requirements and computation, ultimately making the algorithm published in [3] realisable on Cell/B.E.. Although significant effort was invested in the current implementations, it is still introductory work without vectorisation and pipelining optimisations. With new versions of Cell/B.E. and CellSs, further performance improvements are also expected.

References

1. David A. Bader, Virat Agarwal and Kamesh Madduri, *On the Design and Analysis of Irregular Algorithms on the Cell Processor: A Case Study of List Ranking*, 21st Int'l Parallel and Distributed Processing Symp. (Long Beach, CA), March 2007
2. Pieter Bellens, Josep M. Perez, Rosa M. Badia and Jesus Labarta, *CellSs: A Programming Model for the Cell BE Architecture*, Proc. SC06, Nov. 2006
3. Godehard Sutmann, Silke Waedow, *A Fast Wavelet Based Evaluation of Coulomb Potentials in Molecular Systems*, Biophysics to Systems Biology 2006, pp. 185–188