

Implementation of an Effective Non-Bonded Interactions Kernel for Biomolecular Simulations on the Cell Processor

Horacio Pérez-Sánchez, Wolfgang Wenzel

Forschungszentrum Karlsruhe, Institut für Nanotechnologie
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
horacio.sanchez@int.fzk.de

Abstract: In biomolecular simulations intensive computations are spent in non-bonded interactions kernels, i.e., electrostatic interactions. Therefore this part can be considered as a bottleneck, and its optimization permits biomolecular simulation methods to deal with more complex systems or to simulate longer time scales. Using novel computational architectures, i.e., the Cell processor, and programming it adequately in parallel, can considerably improve the performance of biomolecular simulation methods. Programming the Cell processor is difficult, but we show a strategy, using the metacompiler Cellsuperscalar. We obtain sustained speedups of around 150 times.

1 Introduction

Molecular simulation methods are regarded as a helpful tool in the study of biological systems. Among the most important molecular simulation techniques we find Molecular Dynamics (MD), Quantum Mechanical Methods (QM), Brownian Dynamics (BD), etc. In most of them, the biological system is represented in terms of interacting parts or particles, at different levels of detail, i.e., atoms in MD, beads in BD, etc. In the vast majority of all simulation methodologies, the interaction between particles is calculated at different values of time, i.e., as we study the evolution in time of the system. For the calculation of the interaction energies, usually classical potentials are used. These potentials are usually divided in bonded and non-bonded terms. The non-bonded terms describe interactions between all the elements of the system. Classical non-bonded potentials are the electrostatic and the Lennard-Jones potentials.

Non-bonded potentials are represented with the expression;

$$V = \sum_{i,j} \left(\frac{q_i q_j}{r_{ij}} + \frac{R_{ij}}{r_{ij}^{12}} - \frac{A_{ij}}{r_{ij}^6} \right)$$

On the equation we observe that Coulomb and Van der Waals potentials are expressed in terms of double summations between all the elements or particles of the system.

The practical implications of this mathematical form conclude in expensive calculations. As the system grows, or as we represent it with more particles, the computational effort required to compute the potential will grow with $O(n^2)$ and therefore it becomes much more difficult to compute the evolution of the system over time.

Calculation of non-bonded interactions is a common task to different simulation methods. Therefore it is important to have a good and efficient implementation that can be easily adapted to different simulation methods. Solving this limitation allows us to perform longer simulations or to simulate complex systems. One approach in order to solve this problem is the use of novel parallel computational architectures instead of classical serial computers. We can take advantage of the constant development and improvement nowadays in this field of computer hardware if we use them efficiently. Between the last advancements on this field we can find graphical processing units (GPU's) and multicore processors [Cj01, Ct07], or the Cell Broadband Engine (CBE) [Dg07, Gm06]. Currently, the CBE has a lower computer-power/cost ratio than the other systems [Gm07] so we will therefore use the CBE as a platform for the optimization of our non-bonded interactions kernel. We show in this work how an efficient implementation can be carried out, and we suggest how another programmers can develop their simulation codes. The performance of the CBE can be only unleashed when we program it in parallel. For this purpose, the programmer has to put a lot of effort. We also want to manifest the difficulties in the programming of the CBE and how to circumvent them.

The performance of the CBE has been investigated in the cases of matrix operations [Gm07] that are indirectly related with non-bonded interaction kernels. There have been attempts on porting kernels of biomolecular simulation methods to the CBE with modest performance like Gromacs [Os07], with a speedup around 2x, the kernel for the Lennard Jones potential, obtaining in some cases speedups of around 5x [MAV07], and an implementation of both electrostatics and van der Waals kernels by De Fabritiis showing a speedup of around 20x [Dg07]. In this work, we concentrate on the implementation of the electrostatics kernel. Among the main results of our work, we obtain speedups of around 150 with respect to the PPE for the electrostatic kernel, higher than in the previous work already mentioned [Os07, MAV07, Dg07]. We also show a programming strategy, which can be adapted to many methods for biomolecular simulation.

2 Cell processor

The Cell Broadband Engine or CBE, developed jointly by Sony, Toshiba and IBM, combines a general-purpose 64-bit multithreaded PowerPC architecture core (PPE) with 8 coprocessing elements (SPEs), as we can see in Figure 1, which emphasizes peak computational throughput. The architecture of this processor emphasizes greatly efficiency per watt ratio, which is also an important factor when we compare with another processors with similar computational power but usually with much higher energy consumption.

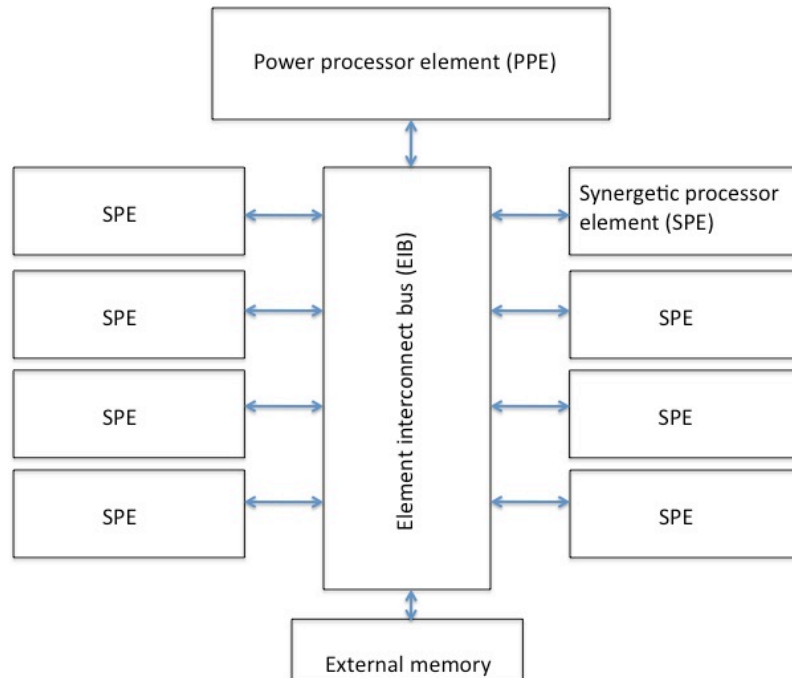


Figure 1: Representation of the CBE. High bandwidth Element Interconnection Bus orchestrates all data transfers between main processor (PPE) and the eight synergetic processor (SPEs)

The cell processor excels at several types of scientific computation [Ws07, Gm06] but is widely regarded as a challenging environment for software development [Gm07]. Main difficulties in programming the CBE are the reduced local storage available for each SPE, which is 256K. Another problem is the management of tasks between the different SPEs and the data transfer with the PPE. In our calculations we will use a IBM Bladecenter QS20, which features two Cell processors at 3.2 GHz, and we will work in single precision floating point.

3 Cell superscalar

Program development for the Cell processor is very complex due to the non-coherence of the main and local memories. For that purpose, the cell-superscalar (CSS) metacompiler has been developed by experts from the Barcelona Supercomputing Center [Pj07], CSS comprises a source-to source C and compiler and a runtime library. It allows the programmers to write sequential applications and the framework is able to exploit the existing concurrency and to use the different components of the Cell processor by means of an automatic parallelization at execution time. Therefore CSS allows a flexible and high-level programming model for the Cell processor.

4 Implementation on the CBE

We describe the strategy we use to implement, in an effective way, the calculation of the coulomb potential on the CBE. The first important requirement is that, given the architecture of the CBE, code and data should fit the 256K of local storage of the SPE. Second requirement is that in order for CSS to perform loop unrolling, tasks should be independent between them. A last important requirement for CSS, is that the tasks must have certain granularity, i.e., should run on the SPE for at least 100 ns or more in order to override the effects of data sending from PPE to SPEs.

```
for(j=0;j<NREC;j++){
  for(i=0;i<NLIG;i++){
    difx=rec[j][0]-lig[i][0];
    dify=rec[j][1]-lig[i][1];
    difz=rec[j][2]-lig[i][2];
    mod2x=difx*difx;
    mod2y=dify*dify;
    mod2z=difz*difz;
    temp_sqrt=sqrt(mod2x+mod2y+mod2z);
    temp_div=1/temp_sqrt;
    S=S+ql[i]*temp_div*qr[j];
  }
}
```

Figure 2: Excerpt of the sequential code used on the PPE for the calculation of the electrostatic interactions

In our approach a random collection of charges and positions that represent a fictitious system is generated on the PPE. Afterwards a electrostatic calculations code, which runs on the PPE, is implemented. We obtain results for different values of the size or particles number of the system. An excerpt of the PPE code can be seen on Figure 2.

In the next stage we run the same code in one SPE. Data transfers from PPE to the SPEs can be implemented in CSS in two different ways, with special DMA routines, or just passing the data as arguments to the functions. We will adopt the last method. We must also be sure that tasks are independent, so loop unrolling can be automatically performed by CSS. In the last stage we also optimize the code for the SPE using vectorization and an approximate function for the calculation of the inverse square root.

```

for(i=0;i<NREC;i++){

    temp_Rjx = spu_splats(Rjx[i]);
    temp_Rjy = spu_splats(Rjy[i]);
    temp_Rjz = spu_splats(Rjz[i]);
    temp_qr  = spu_splats(qr[i]);

    difx=spu_sub(Rix_v[j],temp_Rjx);
    dify=spu_sub(Riy_v[j],temp_Rjy);
    difz=spu_sub(Riz_v[j],temp_Rjz);

    prodx=spu_mul(difx,difx);
    prody=spu_mul(dify,dify);
    prodz=spu_mul(difz,difz);
    mod2=spu_add(spu_add(prodx,prody),prodz);
    inv_dist=spu_rsqrte(mod2);
    q_inv_dist=spu_mul(inv_dist,temp_qr);
    sum_inv_dist=spu_add(sum_inv_dist,q_inv_dist);

}

```

Figure 3: Excerpt of the vectorized code used on the SPE for the calculation of the electrostatic interactions

An excerpt of the SPE code can be seen on Figure 3. Once compiled and running, we will use the profiling tool Paraver [Cj01] to analyze the behavior and performance of the application, i.e., how the calculations are distributed between the different processing units.

5 Results and discussion

As mentioned before, raw programming in C language of the Cell processor is a daunting task, but CSS can generate code for the CBE in a very easy fashion. We consider important that the generated code compiles and runs effectively, and that it can be generated from a human readable and easy original source code. When the original code, consisting of 290 lines of source code is analyzed and compiled with CSS, it generates the different codes for the PPE and the SPEs, and we can count around 3000 lines of source code.

A normal C code can run on the SPE, just without modifications, once we use CSS to send to program the CBE. But in order to take advantage of the CBE, and as we want to get the maximal performance, specific optimizations at the code level must be performed. We show different levels of optimization that we have used for the generation of the code, and how they contribute to a gaining in performance of the overall program.

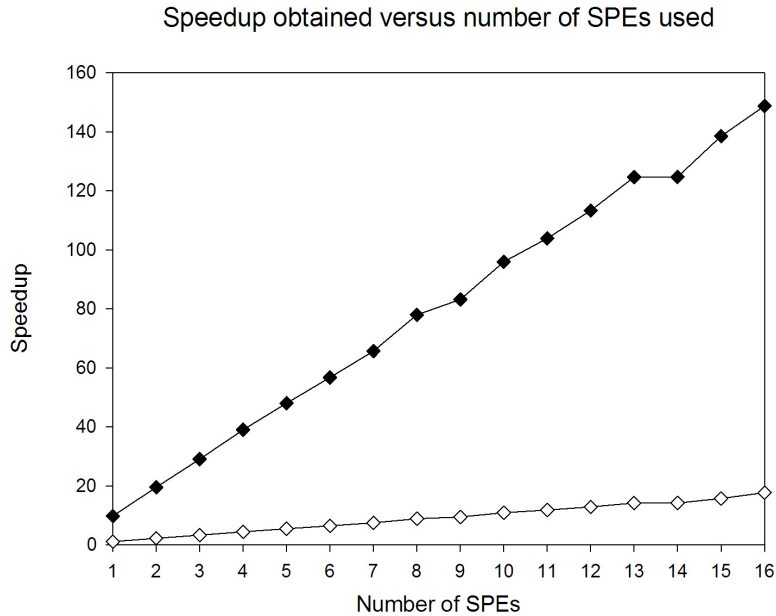


Figure 4: Representation of the speedup obtained versus the number of SPEs used on the calculation for a system with 128 particles. The speedup is calculated relative to the execution of the code on the PPE. White and black diamonds represent results for non optimized and optimized code on the SPEs

It is important for a method to show whether it can scale correctly with the number of parallel computing units of the system. We can observe in Figure 4 the results obtained when we use the same code on the PPE and SPE (white diamonds) and when the code is optimized, i.e., vectorized and approximate mathematical functions used. We can see that our implementation scales well with the number of SPEs in both optimized and not optimized cases.

In Figure 5 we can observe a general tendency of performance improvement as the granularity of the system increases, and the same tendency can be appreciated when the system size grows. A factor that affects performance on the CBE is the granularity of the calculations, in this case we mean number of times the calculations are repeated. Sending data from PPE to the SPEs needs some time, and if the time that the calculation on the SPEs takes less time that the data transfer using DMA, then we will obtain a very low performance, and the PPE will be much faster. In the same fashion the size of the system affects the performance, as bigger systems need to spend more time on the SPEs for the calculation. There is also a limit of the system size that can be used, due to the limited local storage of the SPE. In our calculations this limit is around 8000 particles. But this limit can be surpassed if one modifies the specifications of CSS and uses simple buffering instead of double buffering, and if one designs the code taking into account this memory limitation.

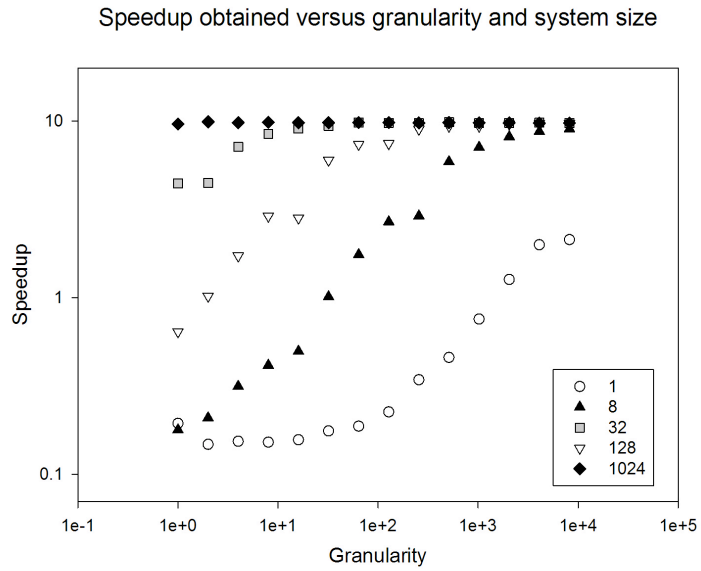


Figure 5: Representation of the speedup obtained for a single SPE versus the granularity of the calculation. The results are obtained for different values of the system size

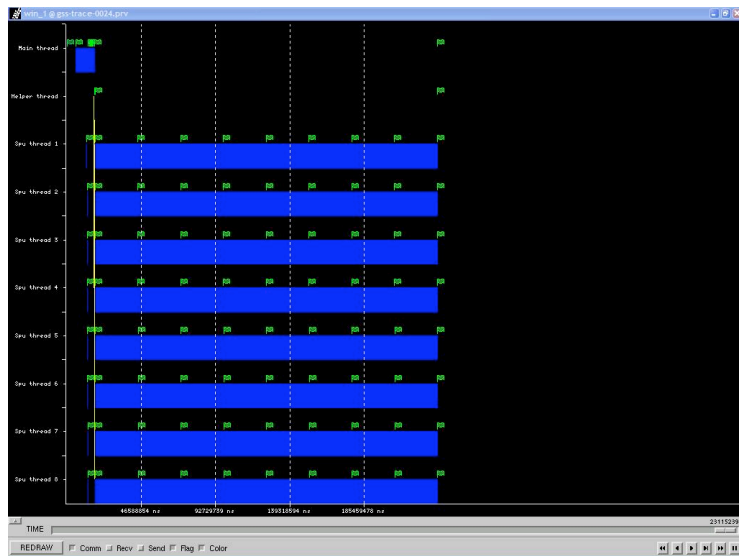


Figure 6: Representation of the tasks distribution and processor usage versus time obtained with Paraver. The first line corresponds to the main thread (PPE), the second one to the helper thread, and the next eight, to the activity on the different SPEs

In Figure 6 we can see a representation obtained with Paraver [Cj01] of the activity in a optimized situation. In this representation, we can see a part of the total simulation. At the beginning the tasks are prepared on the PPE, and afterwards all information is sent to the SPEs, and the program runs at the same time in all of them. In a no effective parallelization we would just see activity in only one of the SPEs and not on the others.

6 Conclusions and future work

We have implemented the calculation of the coulomb interactions, using a strategy to simplify writing code for the CBE. We have shown its application for a simple case, and we focused on this one because it is representative of most molecular non-bonded potentials, and it is easy to extend the conclusions obtained here to another potentials.

Programming the CBE can be very difficult. Using the metacompiler CSS it is much easier to program the CBE than doing it directly with standard C compilers. Anyway, porting a complex simulation methodology can be very time consuming. We propose to design it from scratch and to follow the idea proposed here. The general techniques involved in porting a program to the CBE consist mainly on preparing independent tasks for each SPE. They should also have certain granularity to override the effects of data transfers from PPE to SPEs. The code should also be prepared for automatic loop unrolling so autoparallelization is detectable by CSS, and also for vectorization, using the different built-in intrinsics of the SPEs. We showed how the parallelization and the vectorization of the code on the SPEs affect dramatically the performance of our implementation. We compared execution times using the PPE versus using all the SPEs with the parallelized and vectorized code. We observe a speedup of around 150 when using 16 SPEs.

There are still many aspects of the program that can be improved. Between the most relevant, we are investigating the use of simple buffering instead of double and a better vectorization scheme. At the moment we are also implementing a methodology for docking simulations. The program includes several computation kernels programmed in a similar fashion that the one shown here.

New versions of the CBE will be released, which will be faster and with a higher number of SPEs. Our implementation will be able to run in the new CBE versions without important modifications. Learning to program the CBE now, will be useful for the use of future computational systems.

Acknowledgments This research was supported by the Deutsche Forschungsgemeinschaft, the Barcelona Supercomputing Center and a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme (INSILICODRUGDISCOVER).

References

- [Cj01] Caubet, J., et al., A dynamic tracing mechanism for performance analysis of OpenMP applications. Lecture Notes in Computer Science, 2001: p. 53-67.
- [Ct07] Chen, T., et al., Cell Broadband Engine Architecture and its first implementation-A performance view. *Ibm Journal of Research and Development*, 2007. 51(5): p. 559-572.
- [Dg07] De Fabritiis, G., Performance of the Cell processor for biomolecular simulations. *Computer Physics Communications*, 2007. 176(11-12): p. 660-664.
- [Gm06] Gschwind, M., et al., Synergistic processing in cell's multicore architecture. *Ieee Micro*, 2006. 26(2): p. 10-24.
- [Gm07] Gschwind, M., The Cell Broadband Engine: Exploiting multiple levels of parallelism in a chip multiprocessor. *International Journal of Parallel Programming*, 2007. 35(3): p. 233-262.
- [MAV07] Meredith, J., S. Alam, and J. Vetter. Analysis of a computational biology simulation technique on emerging processing architectures. *IEEE International Parallel and Distributed Processing Symposium*, 2007.
- [Na08] Nakano, A., et al., De novo ultrascale atomistic simulations on high-end parallel supercomputers. *International Journal of High Performance Computing Applications*, 2008. 22(1): p. 113-128.
- [Os07] Olivier, S., et al. Porting the GROMACS molecular dynamics code to the cell processor. *IEEE International Parallel and Distributed Processing Symposium*, 2007.
- [Pd06] Pham, D., et al., Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *IEEE Journal of Solid-State Circuits*, 2006. 41(1): p. 179-196.
- [Pj07] Perez, J., et al., CellSs: Making it easier to program the Cell Broadband Engine processor. *Ibm Journal of Research and Development*, 2007. 51(5): p. 593-603.
- [THM02] Thompson, C., S. Hahn, and M. Oskin. Using modern graphics architectures for general-purpose computing: a framework and analysis. 2002: IEEE Computer Society Press Los Alamitos, CA, USA.
- [Ws06] Williams, S., et al. The potential of the cell processor for scientific computing. 2006: ACM New York, NY, USA.
- [Ws07] Williams, S., et al., Scientific computing kernels on the Cell processor. *International Journal of Parallel Programming*, 2007. 35(3): p. 263-298.