Analysis of Gravitational Wave Signals on Heterogeneous Architectures

Maciej Cytowski

Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw

Abstract

Heterogeneous architectures and programming techniques will be very important in the development of exascale HPC applications. Adapting heterogeneous programming techniques to scientific programming is not always straightforward. Here we present an in-depth analysis of an astrophysical application used for performing an all-sky coherent search for periodic signals of gravitational waves in narrowband detector data. The application was first ported to the PowerXCell8i architecture and then on the basis of achieved performance it was again redesigned and programmed in a heterogeneous model. Moreover presented heterogeneous techniques could be easily adopted for other scientific computational problems involving FFT computations.

1 Introduction

Nowadays using specialized hardware architectures or accelerators for specific computational problems is very common. For large scientific codes it usually means that special programming techniques have to be applied to offload some of the computationally intensive parts of the application on given hardware. Such techniques are usually called heterogeneous computing.

In this work we present an in-depth analysis of an astrophysical application used for performing an all-sky coherent search for periodic signals of gravitational waves in narrowband detector data. The application was ported to a prototype hybrid platform based on the IBM PowerXCell8i architecture. The resulting implementation can be compiled and used as a standalone x86-64 application, standalone Cell application or hybrid x86-64/Cell application. The IBM Cell processor was designed to bridge the gap between general purpose processors and specialized computer architectures like GPUs. The architecture was already extensively described i.e. in [14], [18] and [19]. Supercomputer architectures like Roadrunner [5] or Nautilus [9] utilize the IBM PowerXCell8i processor as an accelerator for calculations running on x86-64 cores. One of the programming techniques available for such heterogeneous architectures is the IBM DaCS library which has proven to be useful in several scientific applications developed for the Roadrunner and Nautilus supercomputers. The described application is one of the codes implemented with the use of DaCS for execution on Nautilus.

The application itself and its reference setup is briefly discussed in Chapter 2. A detailed description of the programming techniques used for implementation of both the PowerXCell8i and hybrid versions of the code is presented in Chapter 3. In this chapter we also show how the custom designed and implemented data conversion mechanism improves the overall performance of the hybrid application. In the last chapter we discuss the results and formulate the conclusions and perspective of code development on described computer architectures.

2 Compute problem

2.1 Scientific background

A gravitational wave is a physical phenomenon which arises from Einstein's theory of general relativity and is defined as a fluctuation in the curvature of spacetime which propagates as a wave, traveling outward from the source. Gravitational waves are radiated by objects whose motion involves acceleration. This class of objects include binary star systems composed of white dwarfs, neutron stars, or black holes. Compared to standard methods used for observing the universe, like visible light or radio telescope observations, gravitational waves have two important unique properties. First of all gravitational waves can be emitted by a binary system of uncharged black holes without presence of any type of matter nearby. Secondly gravitational waves can pass through any intervening matter without being scattered significantly. Both of these features allow researchers to explore astronomical phenomena which have never before been observed by humans. The observations of gravitational waves are usually done by ground-based interferometers. Big research projects like LIGO [2] or VIRGO [7] usually involve operationally running interferometers and produce a significant amount of observational data. This data is subject to further analysis by computer codes developed by research groups involved in those projects. The main computational tasks to be performed on those observational data sets are usually described by algorithms for searching of periodicity or quasiperiodicity.

2.2 Compute algorithm

In this work we present an in-depth analysis of an astronomical application used for performing an all-sky coherent search for periodic signals of gravitational waves in narrowband data from a detector. The search is based on the maximum likelihood statistics called the F-statistics as proposed by P.Jaranowski, A. Królak and B.F.Schultz [17]. The computer code developed by the Polgraw group [4] was used in an operational manner for analysis of observational data from the NAUTILUS [9] and VIRGO [7] detectors. The algorithm implemented in the code was designed for gravitational wave signals generated by rotating neutron stars. The mathematical description of the algorithm was given in previous works of the code authors ([17],[15],[16],[8]). Here we will just shortly describe the most important parts of the code. The simplified flowchart of the code is presented in Fig. 1.

The code begins with reading the observational data sets and setting appropriate parameters for the searching algorithm. After that an outer loop across the sky begins. The very first step in this loop is an ampiltude and phase

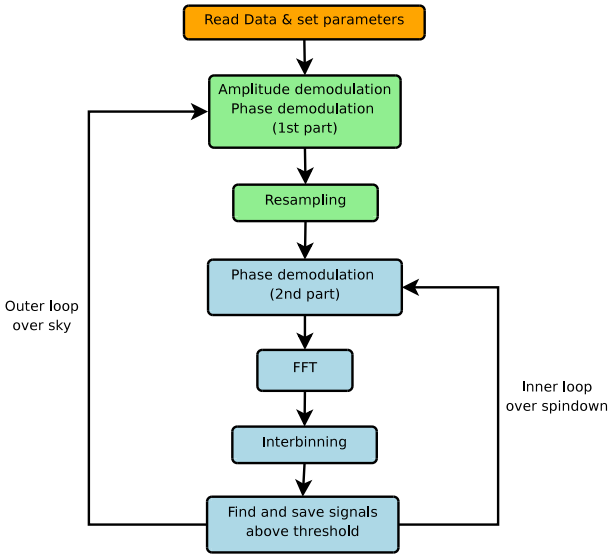

Figure 1: Flowchart of the application.

demodulation. The following step, the so-called resampling of the signal, is performed in two stages: a Fourier interpolation and a spline interpolation. The most computationally expensive part of the whole code is a loop over spindown which can have a length between 0 and 1000 depending on the signal currently analyzed. This loop consists of four main steps:

1. **Phase demodulation (2nd part)** - double precision sin/cos computations, double precision complex multiplications
2. **FFT computations** - double precision 1 dimensional complex forward transforms of size $N = 524288$
3. **Interbinning step** - interpolation algorithm, double precision complex subtractions and divisions, double precision real square root computations
4. **Finding and saving signals** - nonlinear optimization for finding a maximum, saving the resulted signals with values below threshold

3 Methodology

The very first attempt to accelerate the execution of the described application was to port some of its functional parts to the Synergistic Processing Units (SPUs) of the PowerXCell8i processor. In order to select appropriate parts of the application the programmer usually has to perform the following tasks:

- compile, run and measure the performance of the application on a general purpose architecture (i.e. x86-64)
- compile, run and measure the performance of the application on the PPU of the PowerXCell8i processor,

- identify the most computationally intensive parts of the application,
- check the suitability of the selected parts for execution on SPUs.

Unfortunately the usual result of the first two tasks listed above is that the application's performance is much higher on the single core of the x86-64 architecture than on the PPU, which is related to the fact that the Power Processing Unit of the Cell processor was not designed and optimized for computations. This was also valid for the described application. Executions on the PPU were approximately 3 times slower than on a corresponding single core of the x86-64 chip. This observation is of crucial importance to the overall performance of the application on the Cell processor even if some of its parts were already optimized for executions on SPUs. One of the ways to overcome this issue is to use a heterogeneous programming model where only the well optimized parts of the application are executed on the Cell processor whereas the application itself is running on an x86-64 core. In this chapter we present the performance results of the application ported to the Cell processor. Since not all of the parts/algorithms are well suited for execution on the SPUs we decided to use a heterogeneous environment to increase the overall performance. This is briefly described in the second section of this chapter.

3.1 SPU implementation

We have identified 3 functional parts of the application that were especially well suited for execution on the SPU architecture: the 2nd part of phase demodulation, FFT computations and the interbinning step. However to achieve a certain level of granularity for computations on the SPUs we needed to redesign the whole program. We decided to make use of the available RAM memory (8GB for each IBM QS22 Cell blade) and perform all 3 functional parts in seperate loops over spindown. The new resulting flowchart of the program is presented in Fig. 2.

This small change turned out to be very important for the final performance of the application on the Cell chip. Here we will describe the effort we have made to optimize the code for this architecture in detail.

We have implemented a parallel version of the phase demodulation and interbinning step on the SPUs with the use of the libspe2 library [12]. We have used a double buffering scheme for DMA transfers between Local Store of the SPUs and main memory. We have then compared the resulting performance with the use of 8 SPUs (one PowerXCell8i chip).

The FFT step was initially implemented with the use of the Fastest Fourier Transform in the West (FFTW) library [13]. We have decided to use the same library in our implementation since FFTW was already ported to the Cell processor by a group of programmers at IBM Austin Research Laboratories.

A performance comparison between a single core AMD Opteron 2216 processor and a single PowerXCell8i chip for the described functional parts is presented in Fig. 3. It should be also mentioned that all computations involved in the algorithm are based on double precision arithmetic. Therefore the maximum performance rate that could be achieved on one PowerXCell8i chip is 102.4 GFlops. The single core of the corresponding AMD Opteron 2216 processor has a maximum performance rate of 9.6 GFlops.

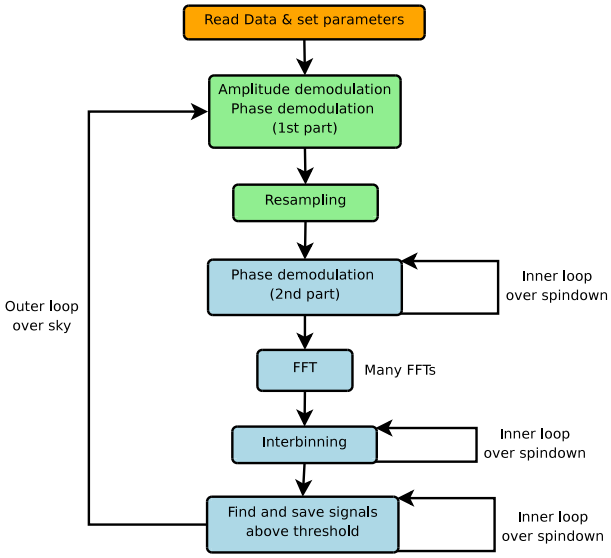

Figure 2: Flowchart of the redesigned application.

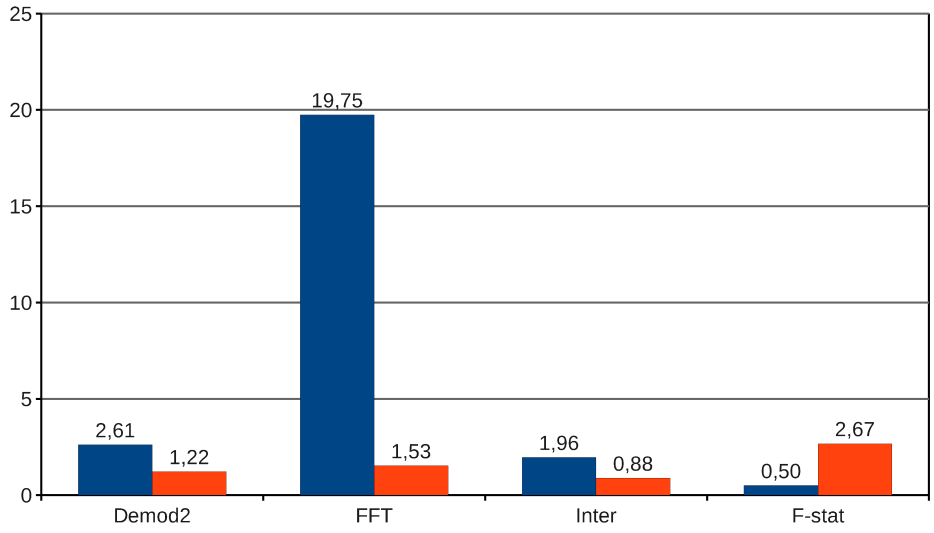

Figure 3: Performance of the described functional parts of the algorithm on the Cell architecture (blue - single core x86-64, red - PowerXCell8i)

We were able to speed up few parts of the application with the use of multiple SPUs. However not all steps of the implemented algorithm could be ported and optimized on the Cell architecture. One example is the so-called "Finding signals" step based on the maximum likelihood statistics. Moreover as we mentioned before computational performance of the PPU core is very poor, thus usually the fragments of the code that are not accelerated on SPUs slow down the overall performance. In particular the "Finding signals" step takes in aver-

age 2.67 sec. on the Cell and 0.5 sec. on a single core of the x86-64 architecture chip. The final result we obtained was approximately **3.24 speedup** of the 8 SPU version compared to a single core x86-64 version.

3.2 Hybrid implementation

The observation that some of the parts of the application have much better performance on the x86-64 architecture encouraged us to prepare a heterogeneous version of the code where we use the PowerXCell8i processor as an accelerator to compute only the functional parts of the application that were optimized for execution on multiple SPUs. For implementing such a scheme we have chosen to use a hybrid library developed by IBM [1].

The Data Communication and Synchronization (DaCS) [1] library and runtime was designed to support the development of applications for heterogeneous systems based on the PowerXCell8i and x86-64 architectures. The DaCS API provides an architecturally neutral layer for application developers. It serves as resource and process manager for applications that use different computing devices. With the use of specific DaCS functions we can execute different remote processes and initiate data transfers or synchronization between them.
One of the main concepts of DaCS is a hierarchical topology which enables application developers to choose between a variety of hybrid configurations. First of all it can be used for programming applications for the Cell processor by exploiting its specific hybrid design. In such a model developers use DaCS to create and execute processes on the PPU and SPUs and to initiate data transfers or synchronization between those processes. It should be stated that developers can choose between a few other programming concepts for Cell processor and that the DaCS model is for sure not the most productive and efficient one. However the DaCS library is much more interesting as a tool for creating hybrid applications that use two different processor architectures, in this case: AMD Opteron and PowerXCell8i. In such a model DaCS can support the execution, data transfers, synchronization and error handling of processes on three different architectural levels: x86-64 cores, PPUs and SPUs. Additionally the programmer can decide to use DaCS with any available Cell programming model on the level of PPU process. PPU process can execute SPU kernels implemented within optimized libraries or created originally by developers with the use of programming models like libspe2 [12], Cell SuperScalar [11] or OpenMP [6].
The DaCS library has a much wider impact on high performance computing since it was designed to support highly parallel applications where the MPI library is used between heterogeneous nodes and the DaCS model is used within those nodes. It is presented schematically on Figure 4. Such programming model was used for application development on the Roadrunner and Nautilus supercomputers ([10],[3]).

The resulting heterogeneous scheme of the application is presented in Fig. 5. The application is executed on the x86-64 architecture. The initialization of the DaCS library is performed in the very beginning of the code together with the allocation of specific memory regions reserved for synchronized data transfers between hybrid processes. At this time the corresponding Cell process is executed on the PPU via the DaCS library and hangs its execution waiting for proper signals. The application parts performed on the Cell processor are

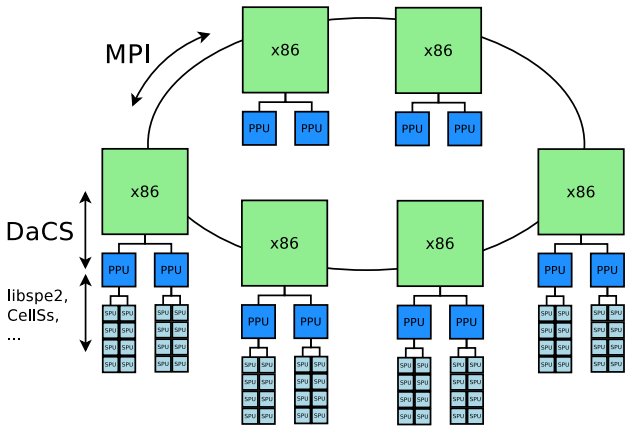

Figure 4: Scheme of multi-level DaCS programming model for heterogeneous architectures.

only those that presented good performance and were optimized for execution on SPUs (the demodulation step, FFTs and the interbinning step). It should be stated here that such an implementation introduces memory transfers between both processes. The size of such data transfers is reaching 1 GB per each outer loop step and thus the performance rate of the interconnect is of crucial importance for the overall performance of the application.

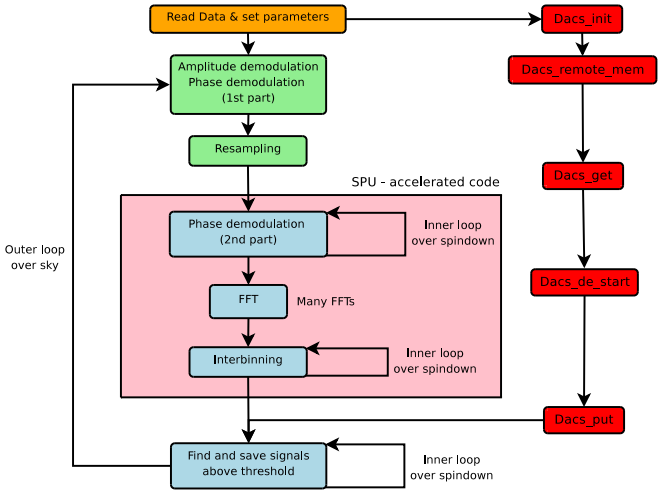

Figure 5: Flowchart of the Hybrid DaCS version of the application.

We have used two heterogeneous systems for testing purposes. Both of them were composed of one IBM LS21 blade and two IBM QS22 blades and they differ in the type of interconnect used for data transfers between those blades. The basic development system installed at ICM uses Gigabit Ethernet. The other system is a node of IBM's triblade cluster (RoadRunner-like prototype system) located at Rochester, USA and uses PCI for data transfers. Like we have assumed the performance of data transfers used for moving data from

x86-64 to the Cell architecture turned out to be very important for the overall speedup of the application. On Gigabit Ethernet the maximum speedup was approx. **1.5**. First measurements made on PCI showed that the performance gain from the hybrid approach is rather small reaching a maximum speedup of **3.56** compared to the previously mentioned **3.24** result in the non-hybrid Cell version of the application. Thus we decided to take a closer look at the performance of the DaCS data transfers on PCI. Those data transfers always involve DMA operations followed by byte swapping applied to the binary data being sent (different endianness of processing devices). In the DaCS library you can decide to turn the byte swapping functionality on and off. We have measured that the byte swapping operation limits the performance of the data transfers over PCI to approx. 280 MB/s. Transfers that don't involve this step can reach performance of more than 1100 MB/s. Following this observation we have decided to implement an optimized version of byte swapping on the Cell processor itself which resulted in much better data transfer performance reaching up to 900 MB/s for big data sizes. Such a high performance level was achieved thanks to a number of Cell programming techniques, mainly: parallel processing on SPUs and SIMD vector processing. Our custom implementation of the byte swapping step is universal and could be used as library by other applications. The following Fig. 6 shows the performance gain of data transfers for different data sizes.

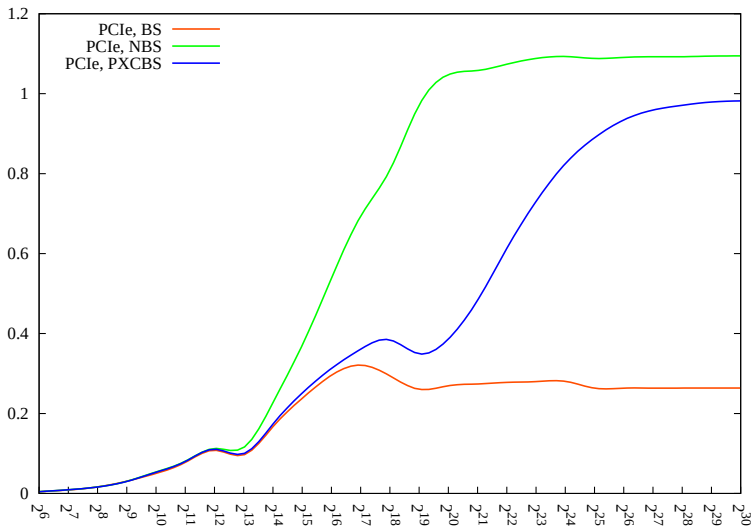

Figure 6: DaCS data transfer performance with optimized byte swapping library (blue) compared to non-byte swapping (green) and DaCS byte swapping (red) versions. Data size in bytes is depicted on the x-axis. Performance measured in GB/s is depicted on the y-axis.

The application described in this work achieved a speedup of **4.5** with the use of our optimized byte swapping technique and this performance level is our final result. Moreover in our opinion it is the highest performance level that could be achieved on the corresponding computer architectures.

4 Results and Conclusions

We have successfully implemented a heterogeneous version of the presented application on RoadRunner-like prototype systems. The application described in this work achieved the highest speedup of **4.5** with the use of a custom optimized byte swapping technique and IBM DaCS programming model. Methods used for the implementation can also be adopted and used in many different scientific applications, especially those involving large FFT computations. The main disadvantage in programming applications for heterogeneous systems is usually related to the necessity of creating few programs dedicated for different computational devices. Therefore our future work on the presented topic will be addressing the development of a heterogeneous parser library for Fourier computations. Such a tool could be used with minor code modifications within scientific codes that make use of the FFTW library. All the important heterogeneous programming issues like data transfers and byte swapping can be hidden behind the library interface. The overall performance of such a heterogeneous tool will be based on the DaCS model and most importantly on heterogeneous techniques developed and used in this work.

Acknowledgements

We would like to thank dr Peter Hofstee (IBM Systems and Technology Group, Austin), Minassie Tewoldebrhan (IBM Rochester) and Maciej Remiszewski (IBM Central and Eastern Europe) for making the IBM triblades available remotely for testing. We have also used the computing resources of the Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw within research grant G33-19.

References

[1] Data Communication and Synchronization for Cell BE Programmer's Guide and API Reference IBM SC33-8408-01, Publication Number: v3.1.

[2] Ligo project *http://www.ligo.caltech.edu/*.

[3] Nautilus supercomputer site *http://cell.icm.edu.pl/index.php/nautilus*.

[4] Polgraw group *http://www.astro.uni.torun.pl/∼kb/allsky/virgo/polgraw.htm*.

[5] Roadrunner supercomputer site *http://www.lanl.gov/roadrunner/*.

[6] OpenMP Application Program Interface, Version 3.0, 2008.

[7] F. Acernese et al. Virgo status. *Class. Quantum Grav.*, 25, 2008.

[8] P. Astone et al. Data analysis of gravitational-wave signals from spinning neutron stars. IV. An all-sky search. *Phys. Rev. D*, 65, 2002.

[9] P. Astone et al. All-sky search of NAUTILUS data. *Class. Quantum Grav.*, 25, 2008.

[10] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho. Entering the Petaflop Era: The Architecture and Performance of Roadrunner. *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*.

[11] BSC. Cell Superscalar (CellSs) User's Manual, Version 2.1. 2008.

[12] I. B. M. Corporation. Programming tutorial. *Technical document SC33-8410-00. Software Development Kit for Multicore Acceleration Version 3.1.*

[13] M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[14] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *IEEE Micro*, 26(22):10–24, 2006.

[15] P. Jaranowski and A. Królak. Data analysis of gravitational-wave signals from spinning neutron stars. II. Accuracy of estimation of parameters. *Phys. Rev. D*, 59, 1999.

[16] P. Jaranowski and A. Królak. Data analysis of gravitational-wave signals from spinning neutron stars. III. Detection statistics and computational requirements. *Phys. Rev. D*, 61, 2000.

[17] P. Jaranowski, A. Królak, and B. Schultz. Data analysis of gravitational-wave signals from spinning neutron stars: The signal and its detection. *Phys. Rev. D*, 58, 1998.

[18] J. Kahle, M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM Journal of Research and Development*, 49(4):589–604, 2005.

[19] M.Kistler, M.Perrone, and F.Petrini. Cell Multiprocessor Communication Network: Build for Speed. *IEEE Micro*, 26(3):10–23, 2006.