Increasing the efficiency of the DaCS programming model for heterogeneous systems

Maciej Cytowski Marek Niezgódka

Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw

Abstract

Efficient programming of hybrid systems is usually done with the use of new programming models. It creates a unique opportunity to increase the performance of scientific applications and is also especially interesting in the context of future exascale applications development where extreme number of MPI processes tend to be a limitation. Future scientific codes will make use of hierarchical parallel programming models with message passing techniques used between nodes and optimized computational kernels used within multicore, multithreaded or accelerator nodes. In this article we consider the x86 and PowerXCell8i heterogeneous environment introduced in the High Performance Computing (HPC) sites like Roadrunner [6] or Nautilus [5]. Programming techniques for this environment are usually based on the IBM Data Communication and Synchronization library (DaCS). We describe our effort to increase the hybrid efficiency of the DaCS library and show how it affects the performance of scientific computations based on FFT kernels. The results are very promising especially for computational models that involve large three dimensional Fourier transformations.

1 Introduction

Novel computer systems from desktops to world's biggest supercomputers are very often based on new architectures or hardware accelerators. Efficient computations on such systems can be achieved with the use of new programming models. The performance of scientific applications can be increased by the functional decomposition of computations and offloading chosen computational kernels on accelerators. However the final performance of those applications depend on their feasibility to the architectures in use and on the performance of the programming tools. Therefore developers have to examine the efficiency of both applications and programming environments in order to produce the fastest solutions for specific scientific problems. In this work we present a performance benchmark of a novel heterogeneous programming model together with its specific scientific application - Fast Fourier Transform (FFT) computations. The outcomes and measurements presented in this work are important not only for FFT computations but for many other computational algorithms and scientific disciplines. The work is accompanied with libraries and example codes.

In this paper we are looking at the heterogeneous programming techniques

for high performance systems based on the PowerXCell8i architecture. The PowerXCell8i was released in 2008 as an enhanced Cell Broadband Engine processor with improved double-precision floating point performance. It is a multi-core chip composed of one Power Processor Unit (PPU) and eight Synergistic Processing Units (SPU). The architecture itself was already extensively described e.g. in [11], [14], [15] and [6]. The PowerXCell8i was designed to bridge the gap between general purpose processors and specialized computer architectures like GPUs. Applications can be compiled and executed in a standard Linux environment on the PPU. Furthermore specific computational kernels can be implemented and executed on the SPUs playing a role of hardware accelerators. Most of the applications achieve poor performance on the PPU since it is not designed for computations. The computations that are not optimized for execution on the SPUs are very often 3 to 5 times slower when compared with their performance on modern x86 compute cores. One of the techniques used to overcome those issues is to use a heterogeneous environment. Therefore HPC architectures like Roadrunner [6] or Nautilus [5] utilize the PowerXCell8i chip as an accelerator for calculations running on x86 cores. Both systems are composed of the IBM LS21 and IBM QS22 blades but they differ in a type of interconnect. In the case of the Roadrunner system [6] the interconnect is based on the PCIe x8 and HyperTransport technology. In the case of the Nautilus system [5] nodes are connected with the DDR Infiniband 4x network. Both systems are important milestones in the development of future HPC systems. Roadrunner is well known to be the world's first TOP500 Linpack sustained 1.0 petaflops system (November 2008). On the other hand Nautilus has been ranked on the 1'st place of the Green500 list twice (November 2008 and June 2009).

Most important programming technique available for such heterogeneous architectures is the IBM Data Communication and Synchronization library (DaCS) used in several scientific codes already developed for the Roadrunner and Nautilus supercomputers ([16], [8], [12]). One of the main advantage of DaCS is that it creates a very interesting hierarchical programming model together with the message passing techniques like the MPI library. This is very important in the context of future exascale applications development where extreme number of MPI processes tend to be a limitation.

We decided to take a closer look at DaCS performance on the Roadrunner like systems. Especially we decided to measure the data transfer rate since it is one of the fundamental factor for application optimization on the accelerator based systems. We describe DaCS functionality and our benchmark results in Chapter 2. The effort we have made to increase DaCS performance is extensively covered in Chapter 3. Finally in Chapter 4 we discuss one of the potential usage scenarios of our implementation, the FFT computations. We describe how to use DaCS for offloading the FFTW library computations on the PowerXCell8i processor.

2 Data Communication and Synchronization library

2.1 Overview

The DaCS [1] library was designed to support development of applications for the heterogeneous systems based on the PowerXCell8i and x86 architectures. It contains two main components: the application programming interface (API) and the runtime environment. The DaCS API provides an architecturally neutral layer for application developers. It serves as a resource and process manager for applications that use different computing devices. With the use of specific DaCS functions we can execute 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. However developers can choose between few other programming concepts for the Cell processor and the DaCS model is for sure not the most productive and efficient one. The DaCS library is much more interesting as a tool for creating hybrid applications that use two different processor architectures. In such a model DaCS can support the execution, data transfers, synchronization and error handling of processes on three different architectural levels (i.e. the x86, PPU and SPU levels). Additionally programmer can decide to use DaCS with any other Cell programming language on the PPU level. The PPU process can execute the SPU kernels provided by optimized libraries or created originally by developers with the use of programming tools like the Libspe2 [13], Cell SuperScalar [9] or OpenMP [7].

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 library is used within those nodes. Such programming model was used for applications development on the Roadrunner and Nautilus systems ([16], [8], [12]).

2.2 Performance benchmarking

A common feature of heterogeneous systems is the bottleneck introduced by the data transfer crossing the accelerator boundary. The computational granularity of the optimized compute kernels must be carefully measured and compared with the data transfers performance in order to make a decision on offloading specific calculations on the accelerator.

The performance measurements presented within this work were prepared on two systems: the Roadrunner-like system located in IBM Laboratories in Rochester and the Nautilus system located at ICM in Warsaw. The best data transfers rate was achieved on the Roadrunner-like system with the use of the PCI x8 interconnect. The measurements on the Nautilus system were performed with the use of the Gigabit Ethernet and Infiniband interconnect. Unfortunately the DaCS library does not support RDMA over Infiniband mechanism which results in a very poor data transfer's rate. In this work we will mainly concen-

trate on the results achieved on the Roadrunner-like system.

We have prepared a performance benchmark for the DaCS library and evaluated it on available heterogeneous systems. It is a simple ping-pong program similar to the one used for benchmarking of MPI point-to-point communication. The sending host process sends a message of a given data size to the accelerator process and waits for a reply. The average time of such communication is measured for different message sizes. The data transfers are initiated with the use of `dacs_put` and `dacs_get` functions.

Developing applications on the heterogeneous systems based on the x86 and PowerXCell8i architectures introduces an additional byte-swapping step related to different endianness of the host and accelerator nodes. The DaCS library addresses this issue by providing a byte swapping mechanism that can be switched on and off by setting corresponding parameters of the DaCS data transfers functions.

The data transfers implemented in our benchmark program were used for transferring of the double precision floating point numbers and were executed with byte swapping mechanism turned on and off for comparison. In this way the influence of the DaCS byte swapping step on the overall performance of the data transfers was measured. We will refer to those two different setups by using two shortcuts for simplicity: BS and NBS will stand for byte swapping and no byte swapping version respectively.

The performance results obtained on the Gigabit Ethernet for medium and large data transfers (between 2^{18} and 2^{30} bytes) were reaching 70 MB/s for NBS version. The average difference between BS and NBS versions for those data transfers is around 9 MB/s. This difference is of minor importance for computations. The performance results obtained on the Infiniband network were reaching the level of 100 MB/s. This weak performance is related to the lack of support for the Infiniband over RDMA transfers in the DaCS library. Much more interesting results were those obtained on the Roadrunner-like system where data transfers are handled by dedicated PCI based interconnect. Although the maximum performance for the NBS version reached more than 1090 MB/s, the BS version was much slower. The difference for large data transfers exceeds 800 MB/s. The results of the benchmark on PCI interconnect are depicted on Figure 2.

3 Optimized byte swapping

The benchmark results described in the previous chapter present an undesirable dependence of the computational performance of DaCS applications on the performance of byte swapping step. The byte swapping mechanism implemented in DaCS seems to be very unoptimal. Unfortunately almost every application implemented on the described heterogeneous system have to make use of it.

Byte swapping is a very simple operation and can be implemented with the use of permutation of bytes which transforms the bit representation of a given number from one endianness format to the other. Our implementation on the PowerXCell8i processor is based on two very important observations. First of all for large data sizes byte swapping can be easily parallelized in a data parallel mode. Secondly byte swapping can be performed with the use of SIMD vector operations. For large data sizes we decided to make use of the available

Table 1: Time measurement of byte swapping optimized kernels.

Kernel version	2^{14} **bytes**	2^{20} **bytes**	2^{30} **bytes**
1 threaded PPU	9 usec	1672 usec	1677745 usec
4 SPU threads	11 usec	1520 usec	99159 usec
AMD Opteron	29 usec	1696 usec	1716653 usec

vector SPUs. For smaller data sizes we propose a SIMD-ized optimal implementation on PPU. The final PowerXCell8i byte swapping library (PXCBS) is a mix of both described implementations based on their performance measured for different data sizes.

3.1 Key optimization steps

Here we describe the consecutive optimization steps of byte swapping for the PowerXCell8i architecture.

SPU implementation

For large data sizes we can parallelize byte swapping simply by dividing the data into blocks. For double precision floating point numbers we choose a block of size 1024 since the maximal transfer size for the DMA operations is 16kB. We use a double buffering scheme to overlap computations and communication. Moreover we exploit the SIMD operations by using the `spu_shuffle` instruction. We've measured the performance achieved with the use of 1,2,4,8 and 16 concurrent SPU threads. Although the final implementation presents very good performance for large data sizes it is not optimal for smaller sizes due to the overhead introduced by creation and maintaining parallel SPU threads during execution.

PPU implementation

For smaller data sizes we decided to implement byte swapping on the PPU. The main optimization technique used here is SIMD-ization. Here we use `vec_perm` Altivec operation instead of `spu_shuffle`. Performance results are summarized in Table 1. Since the PPU is a two-way multithreaded core we decided to create a dual threaded version of byte swapping with the use of POSIX threads. We've measured the performance achieved with the use of 1 and 2 concurrent PPU threads.

Performance measurements

The final version of optimized byte swapping is a mix of developed kernels used interchangeably dependent on the data size. The decision which kernel should be executed is statically implemented in the code based on two comparisons. First of all we compare the performance of those kernels on small, medium and large data sizes (see Table 1). In this comparison we measure only the time needed to perform single byte swapping step.

We have also performed measurements of the full DaCS data transfer process that includes data movement and byte swapping. This was done with the use of the previously described DaCS ping-pong test. The results are presented in Figure 1 for varying data sizes and show that each of the implemented kernels should be considered for usage in the final solution.

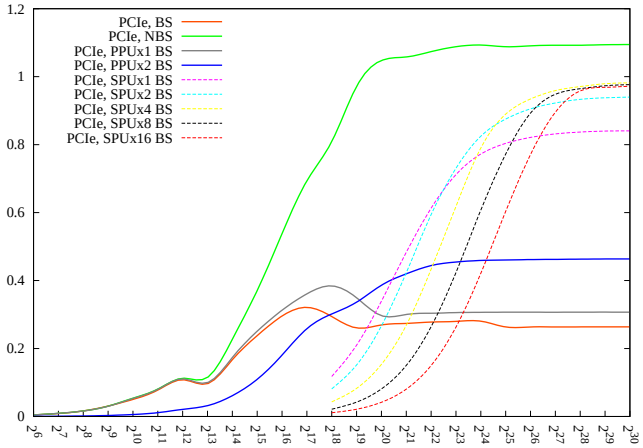

Figure 1: Measurements of data transfers performance for different version of optimized byte swapping (SPU and PPU kernels). Data size in bytes is depicted on the x-axis. Performance measured in GB/s is depicted on the y-axis.

3.2 Result and usage details

Based on the results presented in Table 1 and Figure 1 we have prepared a final implementation which make use of the optimized kernels in a following way:

- the single threaded PPU version is used for sizes smaller than 2^{19} bytes,
- the dual threaded PPU version is used for sizes between 2^{19} and 2^{20} bytes,
- the single SPU version is used for sizes between 2^{20} and 2^{22} bytes,
- the two SPUs version is used for sizes between 2^{22} and 2^{24} bytes,
- the four SPUs version is used for sizes bigger than 2^{24} bytes.

The overall performance of our implementation is presented in Figure 2.

Note that the shape of presented curve is similar to the original BS version for small and medium sizes (growth, local minimas and maximas). However the performance for large sizes achieves a stable level of ~980 MB/s.

The PowerXCell8i optimized byte swapping (PXCBS) is available for download and licensed on the basis of GPL. It is a lightweight library that enables byte swapping for 32- and 64-bit data. The library functions are accessible from PowerXCell8i accelerator code. Basic usage is simple and straightforward. In order to transfer a double precision floating point table T of size N programmer needs to perform the DaCS data transfers with byte swapping turned off and call the `bswap64_pxc(N,T)` function. The program needs to be linked with provided PXCBS library.

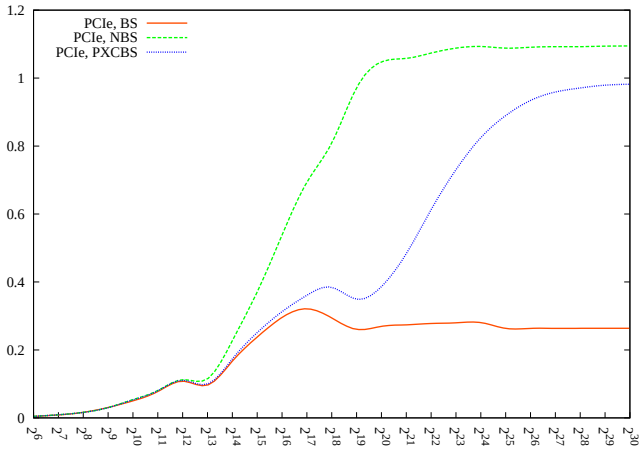

Figure 2: DaCS data transfers performance with optimized byte swapping library compared to previous versions (ping-pong benchmark). Data size in bytes is depicted on the x-axis. Performance measured in GB/s is depicted on the y-axis.

4 Usage scenario: FFTW library

The usage scenario presented in this chapter addresses applications that involve FFT computations. One of the most popular tools used for Fourier transform computations in scientific codes is the FFTW library [10]. The FFTW library was ported and optimized for execution on the PowerXCell8i architecture by IBM Austin Research Laboratories [2]. The performance of the optimized FFTW library on the PowerXCell8i architecture was reported for many different benchmark settings in [3] and [4]. Applications that use FFTW can be compiled and executed on the PowerXCell8i architecture. There are some important caveats that need to be taken into account. First of all the data must be stored in contiguous arrays aligned at 16-byte boundary. Secondly as noted by the developers the `FFTW_ESTIMATE` mode may produce unoptimal plans and the user is encouraged to use `FFTW_MEASURE` instead. However the latter is much more time consuming on the PowerXCell8i processor and for many applications `FFTW_ESTIMATE` is still a better choice. All performance measurements presented in this article were obtained with the use of `FFTW_ESTIMATE` mode.

We present the results of performance analysis of the FFTW library on heterogeneous systems based on the PowerXCell8i architecture. The benchmark problem was designed to evaluate the possible advantages of using such heterogenous approach. We compare the reference x86 performance (AMD Opteron 2216, 2.4 GHz) with corresponding hybrid implementation. Time measurements include the data transfers, byte swapping steps and FFT computations.

4.1 Computational model

We have developed a set of simple heterogeneous programs that serve as a benchmark suite for the FFTW library. The main purpose was to show how

the PowerXCell8i optimized library could be used to accelerate scientific applications on heterogeneous architectures. In a very first step of the program the decision on type, size and direction of the transform is made within application running on AMD Opteron core. These informations are then sent to the accelerator process. In the next step accelerator process allocates tables for transform and prepares the FFT plan with the use of the FFTW library interface. At the same time transformation datas are being prepared on the host process and are sent to the accelerator process when ready. The FFT computations on the accelerator process are preceded and followed by the byte swapping steps. A reverse data transfer is carried out and the FFT plan created for computations is destroyed. The performance comparison presented here is made between reference x86 program and two heterogeneous programs: the one that uses the PXCBS library and the one that uses the DaCS built-in byte swapping mechanism.

4.2 Performance measurements

The performance measurements for 1D, 2D and 3D FFT transforms are presented in Table 2. All performed FFTs are double-precision complex forward transforms. In the case of heterogeneous programs we present the communication time and walltime measured during execution. The heterogeneous programs based on the DaCS library built-in byte swapping mechanism do not achieve a significantly better performance than their x86 equivalents. However the use of optimized byte swapping mechanism gives much better overall performance. The best speedup measured was 4.3x, 4.8x and 7.4x for 1D, 2D and 3D cases respectively.

Table 2: Performance comparison of FFT transforms on heterogeneous architecture.

		DaCS PCI			**DaCS PCI + PXCBS**		
1D FFT size	**x86**	**Comm.**	**Walltime**	**Speedup**	**Comm.**	**Walltime**	**Speedup**
131072	0.025s	0.016s	0.018s	1.38x	0.009s	0.011s	2.27x
262144	0.076s	0.032s	0.035s	2.17x	0.014s	0.018s	4.22x
524288	0.102s	0.064s	0.067s	1.52x	0.024s	0.030s	3.4x
1048576	0.210s	0.127s	0.141s	1.48x	0.043s	0.057s	3.68x
2097152	0.446s	0.254s	0.288s	1.54x	0.079s	0.113s	3.94x
4194304	0.924s	0.503s	0.704s	1.31x	0.146s	0.347s	2.66x
8388608	1.838s	1.007s	1.153s	1.59x	0.282s	0.425s	4.32x
		DaCS PCI			**DaCS PCI + PXCBS**		
2D FFT size	**x86**	**Comm.**	**Walltime**	**Speedup**	**Comm.**	**Walltime**	**Speedup**
256x256	0.009s	0.009s	0.009s	1.0x	0.006s	0.006s	1.5x
512x512	0.073s	0.033s	0.047s	1.55x	0.015s	0.029s	2.51x
1024x1024	0.345s	0.128s	0.169s	2.04x	0.044s	0.086s	4.01x
2048x2048	1.674s	0.512s	0.701s	2.38x	0.156s	0.346s	4.83x
		DaCS PCI			**DaCS PCI + PXCBS**		
3D FFT size	**x86**	**Comm.**	**Walltime**	**Speedup**	**Comm.**	**Walltime**	**Speedup**
64x64x46	0.021s	0.033s	0.036s	0.58x	0.016s	0.018s	1.16x
128x128x28	0.583s	0.261s	0.279s	2.08x	0.088s	0.105s	5.55x
256x256x256	6.062s	2.083s	2.246s	2.69x	0.643s	0.812s	7.46x

5 Summary

In this work we have presented the optimized byte swapping mechanism that replaces its unoptimal equivalent in the DaCS library and can be used for implementation of heterogeneous applications that involve movement of large data between host and accelerator. The performance rate of our solution measured with the use of DaCS ping-pong test is up to 3.7x more efficient in terms of MB/s. This result was achieved by exploiting parallel and SIMD processing features of the PowerXCell8i chip. The PXCBS library can be used together with the IBM DaCS library to support heterogeneous computations. Usually it significantly increases the overall performance and in our opinion it should be always considered as a tool for byte swapping on PowerXCell8i processor.

We have also described how the proposed solution can be directly and efficiently used for accelerated FFT computations based on the FFTW library. One of the key results here is the one reported for 3D Fast Fourier transforms. For large transform sizes like 256x256x256 we've achieved approximately 7.4x speedup over reference x86 implementation. The same hybrid FFT computations based on the DaCS library built-in byte swapping mechanism achieved only 2.6x speedup.
The presented FFT performance result based on optimized byte swapping mechanism can have a significant impact on the performance of many algorithms that involve Fourier transforms: convolution and correlation computations, spectral windowing, power spectra computations, periodicity searching algorithms, linear prediction, wavelet transforms and many more.

All the results presented in this work can be easily reproduced with the use of freely available tools and benchmarks available at: `http://www.icm.edu.pl/~sheed/dacs_performance`.

Acknowledgments

We would like to thank Peter Hofstee (IBM Systems and Technology Group, Austin), Minassie Tewoldebrhan (IBM Rochester) and Maciej Remiszewski (IBM Deep Computing, Central and Eastern Europe) for making IBM triblades available remotely for tests. This research was carried out with the support of the "HPC Infrastructure for Grand Challenges of Science and Engineering" Project, co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme. Part of the calculations were performed in the Interdisciplinary Centre for Mathematical and Computational Modelling (ICM) at University of Warsaw under the computational 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] FFTW on Cell `http://www.fftw.org/cell`.

[3] FFTW on the Cell Processor benchmarks. `http://www.fftw.org/cell/cellblade/`.

[4] JCCC FFTW benchmark suite `http://cell.icm.edu.pl/index.php/FFTW_on_Cell`.

[5] Nautilus supercomputer site `http://cell.icm.edu.pl/index.php/Nautilus`.

[6] Roadrunner supercomputer site `http://www.lanl.gov/roadrunner/`.

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

[8] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation . *Phys. Plasmas*, 15, 2008.

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

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

[11] 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.

[12] S. Habib, A. Pope, Z. Lukić, D. Daniel, P. Fasel, N. Desai, K. Heitmann, C.-H. Hsu, L. Ankeny, G. Mark, S. Bhattacharya, and J. Ahrens. Hybrid petacomputing meets cosmology: The Roadrunner Universe project. *Journal of Physics: Conference Series*, 180(1):012019, 2009.

[13] International Business Machines Corporation. Programming Tutorial. *Technical document SC33-8410-00. Software Development Kit for Multicore Acceleration Version 3.1.*

[14] 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.

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

[16] S. Swaminarayan, K. Kadau, T. C. Germann, and G. C. Fossum. 369 Tflop/s molecular dynamics simulations on the Roadrunner general-purpose heterogeneous supercomputer. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 64:1–64:10, Piscataway, NJ, USA, 2008. IEEE Press.