Radioastronomy Image Synthesis on the Cell/B.E.*

Ana Lucia Varbanescu[1,4], Alexander S. van Amesfoort[1], Tim Cornwell[2], Andrew Mattingly[3], Bruce G. Elmegreen[4], Rob van Nieuwpoort[5], Ger van Diepen[5], and Henk Sips[1]

[1] Delft University of Technology, The Netherlands
[2] Australia Telescope National Facility
[3] IBM, ST Leonards, NSW Australia
[4] IBM Research, T.J.Watson Research Center, Yorktown Hts, NY, USA
[5] ASTRON, Dwingeloo, The Netherlands

Abstract. Now that large radiotelescopes like SKA, LOFAR, or ASKAP, become available in different parts of the world, radioastronomers foresee a vast increase in the amount of data to gather, store and process. To keep the processing time bounded, parallelization and execution on (massively) parallel machines are required for the commonly-used radioastronomy software kernels. In this paper, we analyze data gridding and degridding, a very time-consuming kernel of radioastronomy image synthesis. To tackle its its dynamic behavior, we devise and implement a parallelization strategy for the Cell/B.E. multi-core processor, offering a cost-efficient alternative compared to classical supercomputers. Our experiments show that the application running on one Cell/B.E. is more than 20 times faster than the original application running on a commodity machine. Based on scalability experiments, we estimate the hardware requirements for a realistic radio-telescope. We conclude that our parallelization solution exposes an efficient way to deal with dynamic data-intensive applications on heterogeneous multi-core processors.

1 Introduction

High performance computing (HPC) applications can benefit a lot from the emerging multi-core platforms. However, (legacy) sequential code for HPC applications is not re-usable for these architectures, as they require multiple layers of parallelism to be properly exploited to achieve peak performance [1].

On the other hand many large-scale HPC areas, like radioastronomy, have reached a point where computational power and the efficient ways to use it are becoming critical. For example, radioastronomy projects like LOFAR [2], ASKAP [3], or SKA [4] provide highly accurate astronomical measurements by collecting huge streams of radio synthesis data, which are further processed

* This research is partially supported by the AstroSTREAM project, funded by NWO/STARE, and the SCALP project, funded by STW/Progress.

into sky images. For radioastronomy applications, processing power and storage space need to be used with extreme efficiency: whatever is not computed in time, may get stored; whatever does not fit in the storage space gets lost. Therefore, the choice for a suitable parallel hardware platform is essential, and it may come at the price of increased programming effort. For example, on a heterogeneous multi-core platform like the Cell/B.E., three different parallelism classes - multi-processor, multi-core, and core-level - require specific optimization techniques. Most of the "classical" parallel languages (like MPI or OpenMP) fail to adapt automatically to the different hardware scale. The few specific multi-core programming models like RapidMind [5] or Sequoia [6] do not yet provide the full parallelism range that an application should exploit on a Cell-like architecture.

In this work, we present one of the first successful attempts to parallelize radioastronomy kernels on a heterogeneous multi-core platform. Specifically, we focus on the parallelization of *the gridding and degridding* operations on the Cell/B.E.. Both kernels are implemented using convolutional resampling, a dynamic data-intensive radioastronomy kernel which, as a basic building block in many of the data processing phases, dominates the workload of radio synthesis imaging [3]. Optimizing it has a direct impact on the performance and hardware requirements of any of the large radiotelescopes mentioned above. In response to these requirements, our solution leads to a speed-up factor of more than 20 when comparing the Cell/B.E. solution with the reference (sequential) code running on a commodity machine. Based on these results, we provide a scalability analysis for the present solution and show how our solution can be extended to reach the scale of a real application like SKA.

Using our experience with the convolutional resampling, we show a generic scalable solution for parallelizing dynamic data-intensive applications on Cell-like architectures. We claim that efficient execution on heterogeneous multi-cores *requires* identifying and isolating the dynamic behaviour from the parallel computation. Further, using a master-workers model, the parallel processing is assigned to the "worker" cores, while the dynamic, irregular behaviour is assigned to the "master" core. The advantages are threefold: (1) the worker processing can be optimized with generic in-core techniques, (2) data locality decisions can be taken on-the-fly by a master process with better knowledge of the entire application, and (3) overall application scalability is easier to analyze and improve.

The remainder of this paper is organized as follows. Section 2 briefly presents our application, together with a radioastronomy primer. The target platform and the potential parallelization strategies are presented in Section 3. Section 4 discusses our experiments and results, focusing on the increased performance of the parallel version. Scalability analysis is covered in Section 5. We briefly survey existing related work in Section 6. We conclude that although the performance results are good, further research should address more application specific optimizations and different platforms, as presented in Section 7.

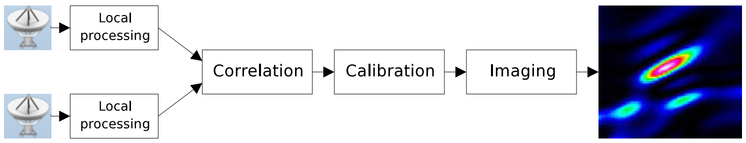

Fig. 1. The software pipeline from antennas to sky images

2 The Gridding and Degridding Kernels

In this section we briefly analyze our application, from its radioastronomy background to its computation and data access patterns.

2.1 Radioastronomy - A Primer

The history of radio astronomy has been one of solving engineering problems to construct radio telescopes of continually increasing angular resolution (i.e., telescopes able to distinguish the finest level of detail in the sky). Very good resolutions require very large antennas. An alternative to building large single-dish radiotelescopes is radio interferometry. This solution enables arrays of connected radiotelescopes (with various antennas types and placement geometries) to collect more signals and, using the aperture synthesis technique [**?**] to significantly increase the angular resolution of the "combined" telescope. By interfering the signals from different antennas, this technique creates a combined telescope the size of the antennas furthest apart in the array.

The simplified path of the signal from the antenna to a sky image is presented in Figure 1. The signals coming from two antennas forming a *baseline*[1] have to be correlated before they are combined. A correlator reads these (sampled) signals and generates the corresponding set of *complex visibilities*, V, depending on the baseline b, the frequency f, and the sample time t. Increasing the number of baselines in the array (i.e., varying the antennas numbers and/or placement) increases the quality of the generated sky image. The total number of baselines B in an array of A antennas is $B = A \cdot (A+1)/2$, and it is a significant performance parameter of the radiotelescope.

2.2 Building the Sky Image

An image of the sky is a reconstruction of the sky brightness using the measured visibilities. For coplanar baselines, (e.g., for a *narrow field of view*), the visibility function and the sky brightness are a 2D Fourier pair. Thus, in practice, image reconstruction uses the discrete Fourier transform on the observed (sampled) visibilities. For the more general case of non-coplanar baselines (i.e., *wide field*

[1] The projected separation between any two individual antennas in the array as seen from the radio source is called *a baseline* and it is described by a set of 3D spatial coordinates, (u, v, w).

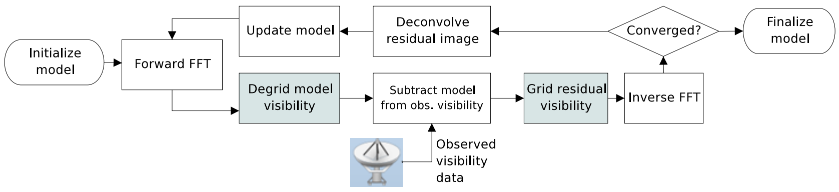

Fig. 2. A diagram of the typical deconvolution process in which a model is iteratively refined by multiple passes. The shaded blocks (gridding and degridding) are both performed by convolutional resampling.

of view), we use the W-projection algorithm [8], which computes one FFT for each projection of the baseline b on P_w parallel planes (usually, betwen 10 and 30), and combines the results.

The practical process of building a sky image has two phases: imaging and deconvolution. The imaging phase generates a *dirty image* directly from the measured visibilities, using FFT. The deconvolution "cleans" the dirty image into a *sky model*. Further, this sky model can be iteratively enhanced by repeating the process using new measured visibilities. A snapshot of this process is presented in Figure 2. Before any FFT operations, data has to be placed in a regularly spaced grid. The operation used to interpolate the original visibility data to a regular grid is called *gridding*. *Degridding* is the "reverse" operation, that projects the regular grid points back to the original tracks; degridding is required when a computed grid is used to refine an existing model.

2.3 Application Analysis

The visibility data is gathered at regular time intervals from each baseline in the system. For a single sample, gridding and degridding are performed by convolution with a function designed to have good properties in the image domain. In practice, all the convolution coefficients are pre-calculated and stored in a large matrix, **C**, and the gridding of the $V(u,v,w)_t)$ visibilities into **G**, the $2^g \times 2^g$ regular grid is implemented by convolution with sub-blocks from **C**. Such a sub-block, SK_M, having $M = m \times m$ elements, is called a *support kernel*. Typical values for M are between 15×15 and 129×129, depending on the required accuracy level. Similarly, degridding uses the same support kernels to transform the data from the regular grid back into the visibility domain, generating a new set of $V'(u,v,w)_t$.

The essential application data structures are summarized in Table 2.3. Data is collected from A antennas (i.e., $B = A \cdot (A+1)/2$ baselines); in one observation session, each baseline is sampled at regular intervals, providing $N_{samples}$ for each one of the chosen N_{freq} frequency channels. For example, at a sampling rate of 1 sample/s, $N_{samples} = 28800$ for an 8 hours observation; N_{freq} can vary between tens and thousands of channels.

The computation patterns for gridding and degridding are presented in Listing 1.1. Note that the effective computation is the same: one complex

Listing 1.1. The core of the convolutional resampling kernel. f1 and f2 are two different functions.

```
forall (i=0..N_freq; j=0..N_samples-1) // for all samples
   compute g_index=f1((u,v,w)[j], freq[i]);
   compute c_index=f2((u,v,w)[j], freq[i]);
   for (x=0; x<M; x++)   //sweep the convolution kernel
     if(gridding)   G[g_index+x] += C[c_index+x]*V[i,j];
     if(degridding) V'[i,j] += G[g_index+x]*C[c_index+x];
```

Table 1. The main data structures of the convolutional resampling and their characteristics

Name	Symbol	Cardinality	Type	Access pattern
Coordinates/baseline	u, v, w	$N_{samples}$	`Real`	Linear
Visibility data	V	$N_{samples} \cdot N_{freq}$	`Complex`	Linear
Convolution matrix	$\mathbf{C}$	$M \cdot os^2 \cdot P_w$	`Complex`	Irregular
Support kernel	SK_M	$M = m \times m$	`Complex`	Linear
Grid	$\mathbf{G}$	512 x 512	`Complex`	Irregular
Grid subregion	SG_M	$M = m \times m$	`Complex`	Linear

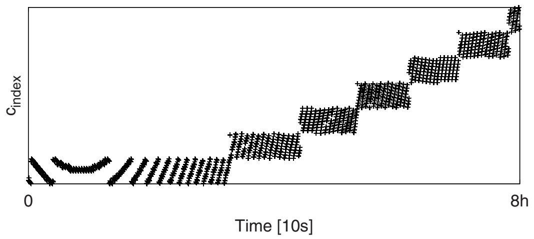

Fig. 3. Irregular access patterns in the **C** matrix for measurements collected by 1 baseline in 8h. The more distant the points are in the Y dimension, the poorer the data locality is.

multiply-add (MADD) operation, i.e., 4 MADD floating point operations. The large execution time of both kernels is caused by (1) the large iteration space ($N_{freq} \times N_{samples} \times M$), and (2) the irregular accesses in both **C** and **G**. Figure 2.3 gives an example of how irregular these accesses are by plotting all the `c_index` values computed for measurements taken by one baseline in an 8-hour session. A similar graph can be drawn for `g_index`. These irregular accesses in both **C** and **G** lead to a data-dependent behaviour of the application, which in turn results in poor data locality and requires a non-trivial data layout for parallelization.

3 Parallelization on the Cell/B.E.

In this section we discuss the parallelization solutions used to efficiently implement the gridding and degridding kernels on the Cell/B.E. platform.

3.1 Cell/B.E. Overview

The Cell Broadband Engine (Cell/B.E.) is a heterogeneous multi-core processor, initially designed by Sony, IBM and Toshiba for the Playstation 3 (PS3) game console. Due to its peak performance levels [**?**], the processor became quickly a popular target for high performance computing applications.

Cell/B.E. has nine cores: one Power Processing Element (PPE), acting as a coordinator for the eight Synergistic Processing Elements (SPEs), which share the main computational load. All cores, the main memory, and the external I/O are connected by a high-bandwidth Element Interconnection Bus (EIB). The theoretical maximum data bandwidth of the EIB is 204.8 GB/s. The PPE contains the Power Processing Unit (PPU) - a two-way multithreaded core, based on the Power Architecture -, separated L1 caches (32KB for data and 32KB for instructions), and 512KB of L2 Cache. The PPE runs the operating system and coordinates the SPEs. An SPE contains a RISC-core (the SPU), a 256KB Local Storage (LS), and a Memory Flow Controller (MFC). The LS is used as local memory for both code and data and is managed *entirely* by the application. All SPU instructions are 128-bit SIMD instructions, and all 128 SPU registers are 128-bit wide. The theoretical peak performance of one SPE is 25.6 single precision GFlops.

The Cell/B.E. cores combine functionality to execute a large spectrum of applications, ranging from scientific kernels [1,10] to image processing applications [11]. The basic Cell/B.E. programming is based on a multi-threading model: the PPE spawns threads that execute asynchronously on the SPEs, until interaction and/or synchronization is required. The SPEs can communicate with the PPE using simple mechanisms like signals and mailboxes for small amounts of data, or DMA transfers via the main memory for larger data.

3.2 Parallelization on the Cell/B.E.

Because both our kernels are data-intensive, the parallelization follows an SPMD model, where all SPEs run the same computation (i.e., the convolution itself) on different sets of data. For an efficient solution, we need to design a balanced data and task distribution, to implement it, and to address the eventual Cell-specific problems that may occur.

Data Distribution. The simplest option for data distribution is to share all data evenly among the SPEs: each core receives a piece of V, $\mathbf{C}$, and $\mathbf{G}$, and computes its share. This solution could provide linear speed-up when increasing the number of participating cores if a contiguous set of data from V would map uniformly in either $\mathbf{C}$ or $\mathbf{G}$. Of course, this is not the case for the irregular access patterns of convolutional resampling. Choosing $\mathbf{G}$ or $\mathbf{C}$ for symmetrical distribution (i.e., each SPE is working *only* with its own subregion, and fetching the other data as needed) leads to overall load-imbalance and extensive communication overhead, as seen in [12]. Finally, we can symmetrically distribute the visibilities $V(u, v, w)_t$. If the data is available offline (i.e., stored in files), a block-based distribution is sufficient for good load-balancing. If the data is streaming in the application, SPE utilization and load balancing are improved by using a cyclic, round-robin-like distribution: each sample fetched by the PPE is distributed to the next SPE in line. The SPE receives the data, fetches $(u, v, w)_t$, computes c_{index_t} and g_{index_t} locally, and then uses DMA to bring the necessary SK_M and SG_M in LS (assuming all these fit!). Although relatively simple, this solution requires excessive SPE-PPE communication, not suitable for the Cell/B.E., where the SPEs are severely underutilized and the overall speed-up (on 16SPEs) is only about a factor of 3.

Implementation. In the end, we opt for a dynamic data distribution, implemented using the master-workers paradigm: the PPE distributes the visibility data samples *on-the-fly*, and stores the share for each SPE in a dedicated queue. In this scenario, the PPE computes the `c_{index}` and `g_{index}`, and distributes adjacent values in the same queue. The solution increases per-SPE data locality, at the expense of the extra main memory consumption; the potential load-imbalance (too many work piling up in the same queue) is controlled by limiting the queue size and allowing more queues to focus on the same subregions.

The SPE performs a simple loop: it polls its queue to check if there is work to do; as soon as there is, the SPE fetches the SK_M and SG_M via DMA, computes the new values for SG_M, and does the DMA-out transfer of the new SG_M. To avoid too expensive synchronization mechanisms, we allocate one grid copy for each SPE. The final result, calculated by the PPE, is a simple addition of these individual grids.

We further optimize this solution as follows: if consecutive SPE queue elements have the same `c_{index}` and `g_{index}`, the data samples are summed. Once the sequence is over, the convolution is no longer executed for each V_i, but only once, for the entire $\sum(V_i)$. In the case only `c_{index}` is the same, we can still spare one DMA transfer for the SK_M data. Similarly, if `g_{index}` is the same, the number of DMA out transfers can be decreased.

Cell-specific Issues. Once the top level parallelization is implemented, we verify the memory footprint of the SPE code: the complete SK_M and SG_M, as well as the local copy of the queue should fit, together with the code, in 256KB of memory. For large values of M, this is not possible. Thus, a slightly more complicated scheme is implemented: for each data sample in the queue, SK_M is fetched entirely in a sequential series of DMA transfers, while SG_M is fetched, updated, and written back line-by-line. Finally, although we only need one SG_M line for the actual computation, we store three such lines - the one being processed, the one ready to be transferred out to the main-memory, and the one being read in for the next computation, thus enabling a simple opportunity for computation/communication overlap Finally, we optimize the core computation of each SPE by partial SIMD-ization and loop unrolling, as well as DMA double buffering [13].

4 Experiments and Results

In this section we present our experiments on two Cell/B.E. platforms and we analyze their results (more detailed experiments and more in-depth explanations are described in [12]).

We run our experiments on two platforms: (1) a PlayStation3 gaming console, and (2) a QS20 Cell blade, a platform for high-performance computing. PS3 has one Cell/B.E. processor, running at 3.2GHz, with six out of the eight SPEs fully available for programming; the QS20 blade has two Cell/B.E. processors, providing (almost) uniform access to 16 SPEs.

Our input data is a collection of samples from a real astronomical measurement. The data is collected from 45 antennas (990 baselines), over a period of 8 hours, with a sampling rate of a one element per 10s. However, all results in this section refer to experiments performed for one baseline only. We discuss the multi-baseline application and its scalability in Section 5.

4.1 Overall Application Performance

The first set of experiments shows the overall performance improvement of the parallelized convolutional resampling. The input data set is a collection of 2880 samples produced by one baseline. The support kernel size varies between $M = 33 \times 33$ and $M = 129 \times 129$ elements. The metric we use is the execution time per operation (i.e., total execution time divided by the number of elementary operations, e.g., the number of grid additions) for both the gridding and degridding kernels. Note that for full utilization and scalability, the values for this metric should be constant (see the Pentium D behavior). We compare the reference code, running on a single core of a 3.4GHz Pentium D machine[2] with the our parallelized version running on different configurations on the two available Cell/B.E. platforms. Figure 4 presents these execution time results, emphasizing the best performance on each Cell platforms and the SPE configuration that generated it.

Note that Cell/B.E. outperforms the sequential machine. Further, the larger the kernel, the better the Cell performance becomes. However, note that for kernels as small as $M = 17 \times 17$, the PentiumD results are somewhat comparable with the Cell ones; also, for $M = 129 \times 129$, where both the PentiumD core and the PS3 hit a memory bottleneck. The case of PS3 is much worse because the total available platform memory is very low. Besides the good speed-up factor (over 20 for $M = 101 \times 101$), these results also signal a significant core underutilization - see the SPEs numbers for each peak performance. This behavior is caused by the small input set: one baseline with 2880×16 data samples does not provide enough computation for a full Cell/B.E.

[2] The PentiumD processor is used here as an instance of a general purpose processor; the choice for this machine was only due to availability, and we use the execution time on PentiumD only as a measure of the performance the reference sequential code, not as a measure of the potential performance of the processor.

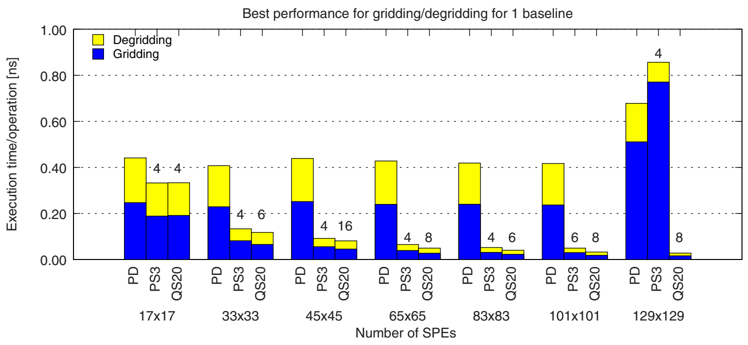

Fig. 4. The overall application performance for the gridding/degridding with different support kernel sizes, running on a PentiumD using 1 core, on the PS3, and on the QS20. The labels inside the graph specify with how many SPEs was each performance peak obtained.

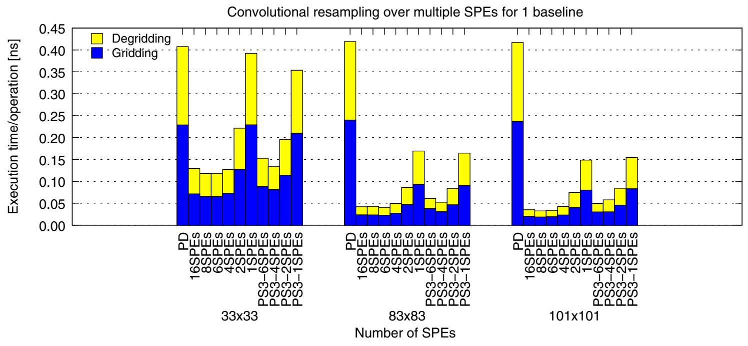

Fig. 5. The performance for the gridding/degridding with different support kernel sizes, running on 6 different hardware configurations

4.2 SPE Utilization

Due to the application low computation-to-communication ratio, we expect the SPEs to be under-utilized. Thus, we show more details on SPE utilization: we measure the execution time for different problem sizes running on different numbers of SPEs (1, 2, 4, 6, 8, and 16). We present the results of these experiments in Figure 5.

Figure 5 shows that for large kernels, i.e., 83x83 elements, the application has good scalability with the number of used SPEs for one Cell/B.E. The best results are indeed obtained for 8 SPEs. The 15% increase in the execution time on 16 SPEs is due to the architecture organization: the 16SPEs are part of two different Cell/B.E. processors, and the "remote" memory accesses from one processor to the other take slightly longer. Thus, for one baseline, one Cell/B.E. (8 SPEs) is sufficient. If a performance penalty of about 10% can be tolerated by the application, using only 6 out of 8 SPEs/Cell provides the hardware cost

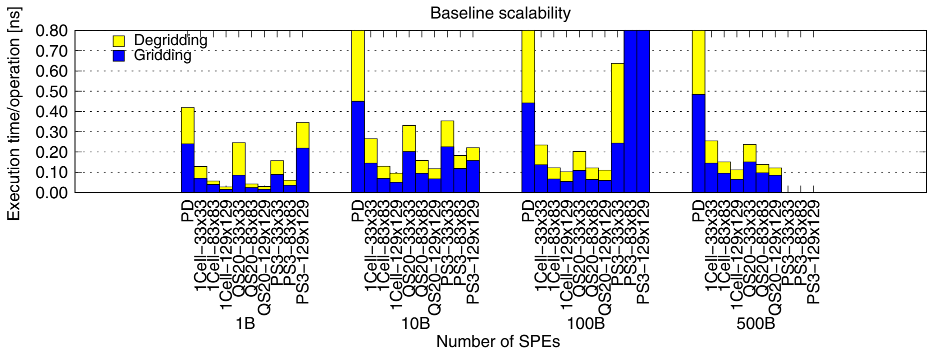

Fig. 6. Baseline scalability: analyzing data from 1, 10, 100, 500 baselines for 4 hardware configurations (PD=1core, PS3=6SPEs, 1Cell=8SPEs, QS20=16SPEs and three different support kernel sizes

advantage (replaces one QS20 with two cheaper PlayStation3). Finally, for smaller support kernels, the SPEs are significantly under-utilized. Ideally, the optimal SPE configuration for a given problem size should be computed by an automated performance predictor. No such tool is available yet for the Cell/B.E.

5 Scalability Analysis

In this section we test the scalability potential of our application, and we evaluate how far off we are from the real system scale.

5.1 Multi-baseline Parallelization

In Section 4, we have discussed the performance results obtained when using the PS3 or QS20 machines to perform the gridding/degridding for a single baseline. To test our solution's scalability, we repeat the experiments for sets of 10, 100 and 500 baselines. To simulate a streaming-like environment, we have used real baseline data, only shuffled such that all data arrives in correct time order, but data coming in the same time interval from different baselines and on different frequency channels has no guaranteed order. We present our results in Figure 6.

Note that the execution time for a single baseline (1B) is better than for a 10 baselines set. This happens because the data locality is decreased by the shuffling. However, as we increase to 100+ baselines, the **G** coverage tends to be more uniform, and the performance is increasing. Also note the differences between the platforms: the sequential version, running on the a single core of a PentiumD, pays some performance penalty only for very large kernel sizes, and only for the gridding operation - probably a cache effect. The PS3 performance drops badly *also* for larger numbers of baselines (about 5-7 times for 100 baselines, and complete crash on 500 baselines), as the processor runs out of main memory. The QS20 - in both 8 and 16 SPEs scales for all test cases.

5.2 The Scale of the Real Application

Radio-telescopes are typically used to gather data during one 8 or 12 hours period (depending on how long the source is above the horizon), and process it later. Data is gathered in a streaming fashion, with a given data rate (e.g, a sample every 10s). For telescopes like SKA, processing is also required to be on-line streaming: gather data, process, move on.

Our goal is to verify whether a collection of X Cell processors can deal with online data processing constraints and, if so, to show how can X be estimated and minimized. Equation 1 shows the upper bound to be enforced on the computation time, T_{sample}^{Cell}, such that data can be processed at a rate of 1 sample/T_{int} [3].

$$\frac{B \times N_{freq}}{T_{int}} \leq \frac{X}{T_{sample}^{multi-Cell}} \tag{1}$$

So, for each data sample (one per baseline per frequency channel) delivered to the X processors in the time T_{int}, there has to be at least one grid addition, which is the time $T_{sample}^{multi-Cell}$. For example, in the case of $B = 1000 baselines$, $N_{freq} =$ 50000 frequency channels, and $T_{int} = 10s$, $X = 5.000.000 \times (T_{sample}^{multi-Cell}/1s)$, or the addition time has to be below $200ns$ for a single Cell to be able to handle the data *computation* online.

To further bound $T_{sample}^{multi-Cell}$, we note the following: (1) even if the computation speed may be able to keep-up with streaming requirements for 1000 baselines, the I/O capacity of Cell/B.E. will not: it is impossible to stream data into one single Cell/B.E. at a rate of $(1000 \times 8)B/s$ [14]; (2) streaming at a low rate may decrease the computation performance due to core underutilization; and (3) the application scalability will be different for more Cell/B.E. processors, especially when "packaged" in different machines, thus $T_{sample}^{Cell} \leq T_{sample}^{multi-Cell}$. A more general model, including these new constraints and presented in Equation 2.

$$\frac{B \times N_{freq}}{T_int} \leq \frac{X}{\max(T_{sample}^{Cell}, T_{streaming}^{multi-Cell}, T_{IO})} \tag{2}$$

Thus, we conclude that our solution for the parallel implementation of the gridding/degridding kernels scales well with the number of baselines. However, although we seem to be able to deal with more than 500 baselines on a single Cell/B.E. processor, slow I/O and streaming operations may as well impose the use of several Cell/B.E. .

[3] B is the number of baselines, N_freq is the number of frequency channels, T_{int} is the time interval for the correlation integral.

6 Related Work

In this section we show how our work can be related with the fields of parallel radio astronomy algorithms and HPC on multi-cores. So far, these two fields have been completely disjoint. Our approach is a first step to high-performance radio astronomy kernels on a high-performance heterogeneous multi-core system.

Astronomers mainly use shared memory and MPI to parallelize their applications. To use MPI on Cell/B.E., a very lightweight implementation is needed and tasks must be stripped down to t in the remaining space of the local store. The only MPI-based programming model for Cell is the MPI microtask model [15], but their prototype is not available for evaluation. OpenMP [16] is not an option as it relies on shared-memory.

Applications like RAxML [17] and Sweep3D [10], have also shown the challenges and results of efficient parallelization of HPC applications on the Cell/B.E. Although we used some of their techniques to optimize the SPE code performance, the higher-level parallelization was too application specific to be reused in our case. Furthermore, typical ports on the Cell, like MarCell [18], or real-time ray tracing are [11] are very computation intensive, so they do not exhibit the unpredictable data access patterns and the low number of compute operations per data byte we have seen in this application.

Currently, efforts are underway to implement other parts of the radio-astronomy software pipeline, such as the correlation and calibration algorithms on Cell and GPUs. Correlation may be very compute-intensive, but it is much easier to parallelize - it has already been implemented very efficiently on Cell and FPGAs [19]. Apart from the general-purpose GPU frameworks like CUDA [20] and RapidMind [5], we are following with interest the work in progress on real-time imaging and calibration [21], which deals with similar applications.

7 Conclusions and Future Work

HPC applications are hard to port efficiently on multi-core processors, due to the multiple levels of parallelism that need to be properly addressed. In this paper, we have presented a significant HPC radioastronomy kernel and we have shown how to efficiently tackle its parallelization on the Cell/B.E. processor. Our approach is based on a locality-enhancing parallelization, which isolates and assigns the lower-level compute intensive blocks to the worker cores, and dedicates master cores to execute the irregular control- and data-flow. Due to its high-level view of the problem, our approach can be easily extended to parallelize dynamic data-intensive applications on heterogeneous multi-cores. We have applied this strategy to the gridding and degridding kernels, one of the very first successful attempts to port radioastronomy kernels on a heterogeneous multi-core processor. The experimental results included in the paper show a 20 times performance improvement of the Cell/B.E. solution over the original sequential code running on a commodity machine, as well as very good scalability. However, due to additional limitations, like the I/O and memory rate, we may still need hundreds of Cell/B.E. to process the LOFAR or SKA data streams on-line.

In the near future, we aim to implement a new series of aggressive, data-dependent optimizations on the current implementation. Further, we aim to test the performance and scalability of the gridding/degridding kernels in a multi-Cell environment. Finally, as data-intensive irregular-access applications like the gridding/degridding kernels are notorious stress-cases for parallel architectures, we plan to make a thorough comparison between parallelization approaches, programming effort and performance pay-off for several multi-core platforms running gridding and degridding.

Acknowledgements. We would like to thank Michael Perrone, Gordon Braudaway, Fabrizzio Petrini and Daniele Scarpazza for their valuable support and ideas during the development of this application. We would also like to thank Jennifer Turner (IBM) for always finding an extra time-slot for us on the QS20 blade.

References

1. Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, K.: The Potential of the Cell Processor for Scientific Computing. In: ACM Computing Frontiers 2006, Italy (May 2006)
2. van der Schaaf, K., Broekema, C., van Diepen, G., van Meijeren, E.: The lofar central processing facility architecture. Experimental Astronomy, special issue on SKA 17, 43–58 (2004)
3. Cornwell, T.J.: SKA and EVLA computing costs for wide field imaging. Experimental Astronomy 17, 329–343 (2004)
4. Schilizzi, R.T., Alexander, P., Cordes, J.M., Dewdney, P.E., Ekers, R.D., Faulkner, A.J., Gaensler, B.M., Hall, P.J., Jonas, J.L.:, Kellermann, K.I.: Preliminary specifications for the square kilometre array. Technical Report v2.4 (November 2007), `www.skatelescope.org`
5. McCool, M.: Signal processing and general-purpose computing on GPUs. IEEE Signal Processing Magazine, 109–114 (May 2007)
6. Fatahalian, K., Knight, T.J., Houston, M., Erez, M., Horn, D.R., Leem, L., Park, J.Y., Ren, M., Aiken, A., Dally, W.J., Hanrahan, P.: Sequoia: Programming the memory hierarchy. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (November 2006)
7. Thompson, A., Moran, J., Swenson, G.: Interferometry and synthesis in radio astronomy. Wiley, New York (2001)
8. Cornwell, T., Golap, K., Bhatnagar, S.: W projection: A new algorithm for wide field imaging with radio synthesis arrays. In: Astronomical Data Analysis Software and Systems XIV ASP Conference Series, vol. 347, p. 86–95 (2004)
9. Gschwind, M.: The Cell Broadband Engine: Exploiting multiple levels of parallelism in a chip multiprocessor. International Journal of Parallel Programming 35(3), 233–262 (2007)
10. Petrini, F., Fernàndez, J., Kistler, M., Fossum, G., Varbanescu, A.L., Perrone, M.: Multicore Surprises: Lessons Learned from Optimizing Sweep3D on the Cell Broadband Engine. In: IPDPS 2007. IEEE/ACM (March 2007)

11. Benthin, C., Wald, I., Scherbaum, M., Friedrich, H.: Ray tracing on the Cell processor. In: IEEE Symposium on Interactive Ray Tracing 2006, pp. 15–23 (September 2006)
12. Varbanescu, A.L., van Amesfoort, A., Cornwell, T., Elmegreen, B.G., van Nieuwpoort, R., van Diepen, G., Sips, H.: The performance of gridding/degridding on the Cell/B.E. Technical report, Delft University of Technology (January 2008)
13. IBM: Cell Broadband Engine Programming Tutorial. 2.0 edn. (December 2006)
14. Hofstee, P.: Power efficient processor architecture and the cell processor. In: HPCA 2005, pp. 258–262. IEEE Computer Society Press, Los Alamitos (2005)
15. Ohara, M., Inoue, H., Sohda, Y., Komatsu, H., Nakatani, T.: MPI microtask for programming th Cell Broadband Engine processor. IBM Systems Journal 45(1), 85–102 (2006)
16. O'Brien, K., Sura, Z., Chen, T., Zhang, T.: Supporting openmp on the cell. In: International Workshop on OpenMP (2007)
17. Blagojevic, F., Stamatakis, A., Antonopoulos, C., Nikolopoulos, D.S.: RAxML-CELL: Parallel phylogenetic tree construction on the cell broadband engine. In: IPDPS 2007, Long Beach, CA. IEEE/ACM (March 2007)
18. Liu, L.K., Liu, Q., Natsev, A.P., Ross, K.A., Smith, J.R., Varbanescu, A.L.: Digital Media Indexing on the Cell Processor. In: ICME 2007, N/A (July 2007)
19. de Souza, L., Bunton, J.D., Campbell-Wilson, D., Cappallo, R.J., Kincaid, B.: A radio astronomy correlator optimized for the Xilinx Virtex-4 SX FPGA. In: International Conference on Field Programmable Logic and Applications (2007)
20. ***: nVidia CUDA - Compute Unified Device Architecture Programming Guide. nVidia (2007)
21. Wayth, R., Dale, K., Greenhill, L., Mitchell, D., Ord, S., Pfister, H.: Real-time calibration and imaging for the MWA (poster). In: AstroGPU 2007 (November 2007)