Implementation and Performance Evaluation of Polyphase Filter Banks on the Cell Broadband Engine Architecture

by

Brandon Kyle Hamilton

Submitted to the Department of Electrical Engineering
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Electrical and Computer Engineering

at the

University of Cape Town

October 2007

Supervisor: Professor Michael R. Inggs

Abstract

This thesis investigates the performance of the Cell Broadband Engine (Cell BE) Architecture for executing a Polyphase Filter Bank algorithm, used for efficient signal channelisation. The heterogeneous multi-core architecture of the Cell BE is introduced and important considerations for developing programs for the Cell BE are discussed. The Polyphase Filter Bank DFT channeliser is discussed in depth, and the process of mapping this algorithm onto the Cell BE processor architecture is shown. An evaluation of the performance of the Polyphase Filter Bank algorithm on the Cell BE processor is presented with results obtained from the Sony Playstation 3 and the IBM Full System Simulator. The effect of the parameters of the algorithm on the performance is investigated. The Cell BE processor is shown to be efficient for some algorithms, and the factors that have both positive and negative impact on its performance are presented. The current implementation of the Cell BE processor has some important limitations for use in scientific computation.

Contents

List of Figures

List of Algorithms

Acknowledgments

I would like to express gratitude to the following people:

Dr Alan Langman for always making time for discussions and the great effort that he put into helping me sort out all of the problems I had during the course of this project , as well as the constant encouragement and support.

Prof. M.R. Inggs for wisdom and supervision.

Peter McMahon for his advice and answers to any question that I had, as well as comments on earlier drafts of this report.

Andrew Woods for helping me understand some concepts that were important for this project and more.

Vanessa Marques de Freitas for all the constant support and inspiration.

Chapter 1

Introduction

This thesis investigates the performance benefits that can be achieved by the use of the Cell Broadband Engine (Cell BE) Architecture for signal processing applications, specifically in the area of radio astronomy. This introduction will provide a background to the content of the thesis, as well as lay out the objectives in some detail. Finally, an outline for the contents of the thesis will be presented.

1.1 Background

1.1.1 Radio Astronomy

Interferometry is used in radio astronomy to enable radio wave signals received simultaneously by multiple telescopes in an array to be combined. This allows high resolution radio band signals to be obtained due to the large effective size of the synthetic aperture formed by the multiple telescopes. The received signals are passed through a correlator to compensate for the delay, phase and amplitude variations from the different sources, after which they are added to obtain a high resolution signal of complex visibilities.

Correlators are typically classed as either XF or FX type, with the XF type correlator traditionally favoured for its lower complexity and lower data

rate. More recently, the availability of inexpensive high performance processors have enabled the FX type correlator to become a more popular choice, and successful results have been achieved with pure software implementations such as the DiFX software correlator [11] developed to run on the Swinburne supercomputer. By first channelising the signal into frequency bands, and then performing a cross-multiplication of each channel, the FX correlator reduces the number of cross-multiplications that are needed in the XF type correlator [2]. In a correlator design, an initial coarse channelisation stage could be applied, which splits the data into a number of smaller bandwidth channels. The total bandwidth is then reduced as only the channels with frequencies of interest are channelised further into many fine bandwidth channels for analysis.

The channelisation stage of the FX correlator has been performed by making used of the Fast Fourier Transform (FFT) by Chikada et. al [7] in their Digital FFT Spectro-Correlator, as well as by Romeny in the Very Long Baseline Array (VLBA) [32]. The higher computational cost associated with the FFT has been viewed as a disadvantage of the FX correlator, but Bunton [2] has shown that a polyphase DFT filter bank is an efficient solution for performing the channelisation of data in the initial stage of the FX correlator. By using the polyphase filter bank implementation instead of the standard DFT, the effect of narrowband components that cause a degradation in signal-to-noise ratio during correlation is minimized [2]. Oversampling the data in each channel will also reduce the degradation further [4], as well as allow the channels produced during the initial course channelisation to be further channelised to to obtain a higher frequency resolution [3]. The polyphase filter bank algorithms are thus an important component of an efficient correlator implementation, and have been identified as ideal for implementation on the Cell BE due to the highly parallel nature of the algorithms.

1.1.2 The Cell BE Processor

The Cell Broadband Engine Architecture has been designed and developed by a consortium formed by Sony, Toshiba and IBM, know as STI. The Cell BE processor, the first processor based on the Cell Broadband Engine Architecture, became commercially available in November 2006 with the release of the Sony Playstation 3. The Cell BE consists of a central Power Processing Element (PPE) based on the 64-bit PowerPC architecture, which is suited to running control and operating system code, as well as eight Synergistic Processing Elements (SPE) suited for compute intensive code. The nine processor cores are connected to external memory via the high performance Element Interconnect Bus (EIB).

The Cell BE is capable of high peak performance rates, and has a theoretical floating point processing rate of over 200 GigaFLOP/s (200 billion floating point operations per second), compared to the traditional current state-of-the-art processor running at just over 20 GigaFLOP/s [15]. Benchmarks have shown that the Cell BE processor is particularly well suited to the speedup of single precision floating point operations, as the performance of the Fast Fourier Transform (FFT) has been shown to be faster than current state-of-the-art processors by a factor of about 10 to 20 [8].

The advantage of using the Cell BE architecture as a platform for signal processing is derived from the high computational density of the Synergistic Processing elements, achieved through thread and data level parallelism, as well as the multiple threads of hardware execution and the support for a large number of concurrent memory accesses [15]. The power of the Cell BE processor can be further extended by implementing the processor as a node in a cluster computing environment [22].

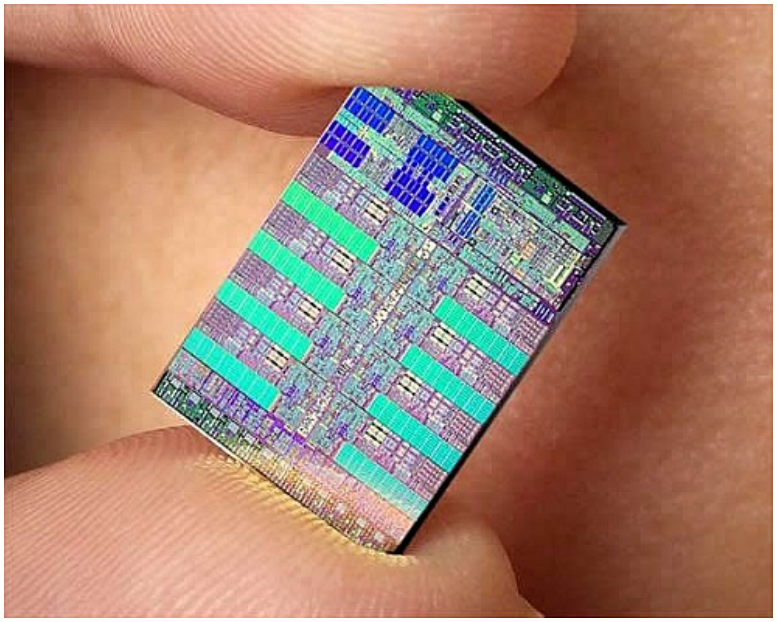


Figure 1.1: The Cell BE Processor

1.2 Objectives

This thesis aims to implement the polyphase filter bank algorithms for signal channelisation on the Cell BE, and to evaluate the performance on this architecture, drawing a comparison with current state-of-the-art traditional multicore processors.

1.2.1 Implement Polyphase Filter Bank algorithms on the Cell BE processor

The polyphase representations for critically sampled and over-sampled filter banks will be implemented in C code. These implementations will be analysed and optimised to run on the Cell BE architecture. The process of developing the code to run efficiently on the Cell BE processor will be described in detail, including motivations for the methods used and a discussion on general issues faced during development.

1.2.2 Evaluate performance of the Cell BE for digital signal processing applications

The performance of the polyphase filter bank algorithms will be carefully measured and benchmarked against the same algorithms running on currently available multicore processors. The factors affecting performance will be determined, and the features of the algorithms that enable performance increases on the Cell BE will be identified. An overlying objective of this thesis is to evaluate the potential performance of the Cell BE for compute intensive signal processing applications, such as a full software correlator implementation, as well as other algorithms used in radio astronomy. The polyphase filter bank performance on the Cell BE will be evaluated using a variety of signals and input parameters. The performance impact due to the data representation will also be investigated.

1.2.3 Provide general guidance for Cell BE development for signal processing applications

This thesis also aims to provide an overview of the procedure used to implement signal processing algorithms on the Cell BE platform, describing in detail the design decisions and the process of development and optimisation of the code for the Cell BE.

1.3 Scope and Limitations

This thesis report investigates the implementation of a specific digital signal processing algorithm, that of the polyphase filter bank, on the Cell Broadband Engine processor in the Sony Playstation 3. The mapping and partitioning of the algorithm described is specific to the heterogeneous multicore architecture of the Cell BE, and the optimisations are based on an analysis of the currently available version of the Cell BE processor. The assumptions and limitations of the algorithm will be discussed in the relevant sections of

this report.

1.4 Document Outline

The thesis will be presented as follows:

Chapter 2 will introduce the Cell Broadband Engine Architecture. The Cell BE processor will be presented and the main components of it will be described in some detail. Performance specifications for the processor will be provided. I will explore the features of the Cell BE processor that make it a suitable candidate for high performance computing, and enable efficient signal processing applications, focusing on the programming models that support this type of algorithm. A brief introduction to the Sony Playstation 3 will describe the hardware that the implementation will be executed on.

Chapter 3 provides the relevant theory of multirate signal processing, along with the mathematical development of filter banks and the polyphase representation that affords an efficient practical implementation. The mathematical framework for polyphase filter banks will be described, specifically for the cases of the critically sampled filter bank implementation, as well as the oversampled filter bank implementation.

Chapter 4 presents the details of the approach taken in mapping the mathematical description of the algorithms for polyphase filter banks onto an efficient implementation executable on the Cell BE processor. The methods used and design decisions will be discussed, and the process of development and debugging on the Sony Playstation 3 console will also be described in detail. The programming model used to implement the algorithms will be explained and justified.

Chapter 5 presents the performance results obtained by measuring the execution of the code on the Cell BE processor. The performance results are

compared and benchmarked, and all the parameters and specifications used are discussed in detail. The details of the performance tests that are performed will be discussed, as well as the relevant comparisons of performance specifically for signal processing algorithms such as the polyphase filter bank algorithms that will be used. This chapter will also predict and investigate parameters affecting performance relating to the implementation of the algorithm on the Cell BE.

Chapter 6 will conclude the thesis, providing a summary of the main results obtained and an analysis of the relevance of the results. The evaluation of the Cell BE as a platform for high performance signal processing applications in radio astronomy, as well as other fields that make use of polyphase filter bank theory, will be discussed. In this chapter I also discuss possible further research that may build on this thesis and explore the use of the Cell BE for a wider variety of signal processing algorithms.

Chapter 2

The Cell Broadband Engine

2.1 Architectural Overview

The Cell Broadband Engine Architecture is designed as a heterogeneous multi-core processor system. The Cell BE contains a Power Processing Element (PPE) and a number of Synergistic Processing Elements (SPE) to which the PPE can delegate computation.

2.1.1 The Power Processing Element

The Power Processing Element (PPE) of the Cell BE is a 64-bit processor based on Power Architecture[1]. The PPE features a Reduced Instruction Set (RISC) design, with in-order processing and support for VMX (AltiVec) vector instructions. The datapath of the PPE allows for execution of two threads simultaneously in hardware. In the Cell BE programming model, the PPE is suited for control, task switching and operating system level support, and it will typically be used to run the operating system. By making use of dual-threading and virtualisation, the Cell BE is capable of running two operating systems simultaneously; it is possible to run a Real-Time Operating System (RTOS) concurrently with a standard operating system.

[1]Family of processors including the PowerPC and POWER processors

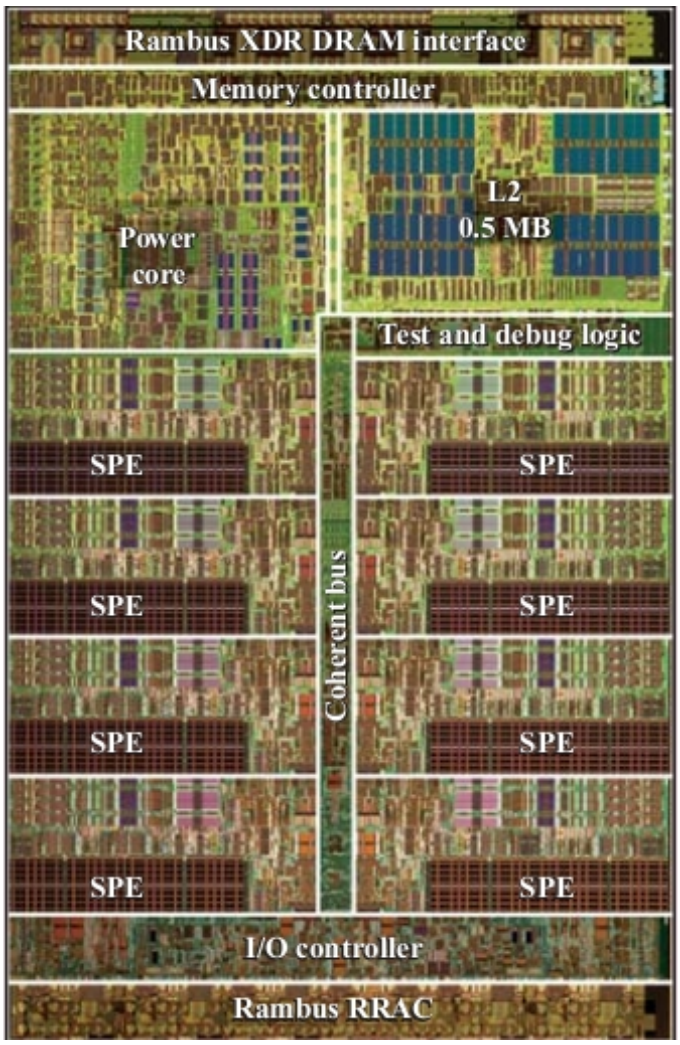


Figure 2.1: The Cell BE Processor Board Layout [19]

The PPE contains a coherent hierarchical cache, featuring an 512KB 8-way set associative write-back L2 cache, with a cache-line size of 128 bits [6], as well as a 32KB L1 cache for instructions and data.

2.1.2 The Synergistic Processing Elements

Each SPE is composed of a Synergistic Processing Unit (SPU), a local memory store (LS) and a memory flow controller (MFC). The SPE design was optimised for low area and low complexity [14]. Unlike modern processors, the SPE does not implement a hardware cache, and all memory operations are software controlled.

The SPE is a vector processor supporting Single Instruction Multiple Data (SIMD) on a 128-bit wide data path. SIMD instructions are pipelined on the SPU, which enables the SPE to execute a vector operation in a single clock cycle. The SPE is designed for wide data-width vector processing, with no separate support for scalar processing. The technique of scalar layering [14] is used to perform efficient scalar operations on the wide data-path. The architecture of the SPE is optimised for compute intensive code, and it contains a large 128 element 128-bit wide register file.

The SPU is only able to address memory in its own LS, which is used to store both instructions and data. The MFC controls the asynchronous coherent Direct Memory Access (DMA) [19] to load and store data between the local store and main memory, as well as mailboxes and signalling between processors. The PPE is able to communicate with the MFC of a PPE through MMIO registers.

To achieve high performance for compute intensive processing, the SPE eliminates the overhead of load and store address translation by only addressing the local store. The SPE is able to achieve higher density due to lack of hardware elements needed to implement and manage a hardware cache and tags. Support for only in-order instruction issuing and elimination of

branch prediction logic provide further performance gains [17]. Instructions and memory operations on the SPU are statically scheduled, which allows the compiler to optimise code for the processor and achieve performance close to the theoretical peak.

The architectural design of the Cell BE processor has enabled it to deliver compute performance equivalent to that of a supercomputer for consumer applications.

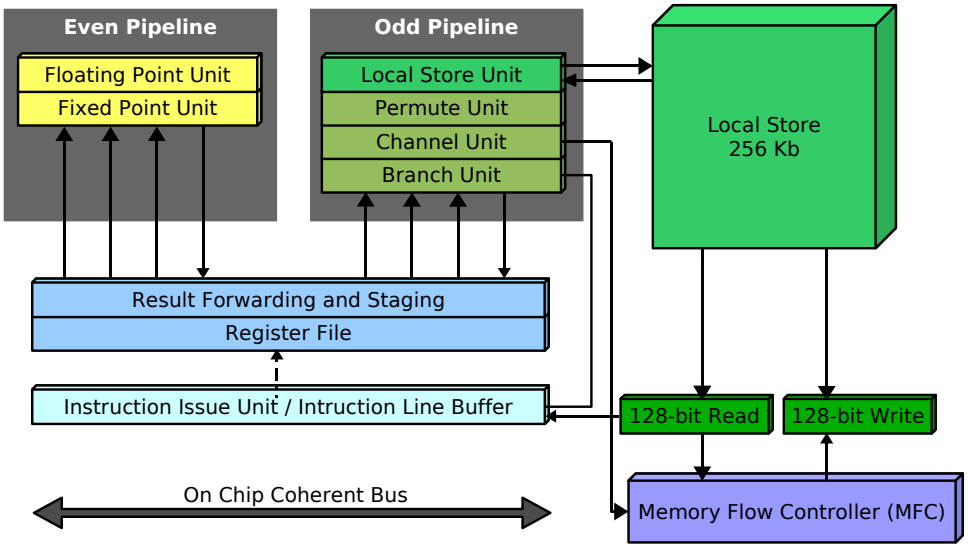


Figure 2.2: Architecture of the Synergistic Processing Element

2.1.3 Memory

The high performance Element Interconnect Bus (EIB) consists of four data rings that connect the PPE, SPEs, DRAM controller and the I/O Controllers. The EIB is optimised to achieve speeds of over 300 gigabytes per second, and a throughput of about 25.6 GB/s is typically achieved per SPU [26] The memory interface is optimised for aligned quadword access, reducing the complexity of the memory control as well as reducing latency [14]. This means that all memory transfer source and destination addresses need to be quadword aligned to optimise bus transfers. The software controlled mem-

ory architecture separates memory load and store latencies from computation, which provides an advantage over a cache-based processor as it enable latency hiding.

The Cell BE processor features a memory subsystem with 16 memory banks that are interleaved on cache line boundaries. Memory access to addresses that are separated by 2KB will access the same memory bank. If all of the memory banks are uniformly accessed in a program, the memory throughput will be maximised.

The DMA queue in the MFC supports up to 16 entries. DMA instructions are issued by the SPU asynchronously to the MFC. DMA transfers in the queue are not guaranteed to be carried out in the order in which they were issued. Two mechanisms exist for synchronisation of the DMA requests in the queue to ensure ordered transfers, the DMA fence and DMA barrier.

Each SPE local store, as well as all external memory and peripherals mapped to a virtual effective address memory space of 2^{64} bytes.

2.2 Cell Broadband Engine Software

The heterogeneous processor elements in the Cell Broadband Engine Architecture require separate programs, as the instruction sets for SPE and PPE are different. The compiler generates a program for the SPE, which can then be embedded into the PPE program as a Cell Embedded SPE Object Format (CESOF) binary, or executed separately on a single SPE. The Cell BE processor is capable of running 32-bit as well as 64-bit PowerPC and POWER applications on the PPE, and this has made it possible to run most distributions of linux that are compiled for the PowerPC, as well as any standard PowerPC binary within the operating system.

Programming the Cell BE differs from standard platform development in that all the hardware is directly exposed to the developer who has full control over the processor. There currently isn't compiler available that is capable of fully optimising code for the Cell BE processor, and so code has to be manually optimised to exploit hardware features.

2.3 Sony Playstation 3

Figure 2.3: The Sony Playstation 3

The *Sony Playstation 3* (PS3) features the Cell Broadband Engine processor and is relatively inexpensive and commercially available. The Cell BE processor in the PS3 runs at $3.2GHz$, and only provides access to six of the eight SPEs on board: one SPE is is assumed disabled (due to fabrication yield) and another is allocated to the operating system virtualisation layer, known as the Hypervisor. The Cell BE in the Playstation 3 is therefore capable of performing up to 153.6 GigaFLOP/s utilising all six of the SPEs. The PS3 has 256MB dual-channel RAMBUS Extreme Data Rate (XDR) memory, with only 200MB RAM available to a 3rd party linux operating system.

External memory is accessed by the XDR memory controller that supports a bandwidth of 25 GB/s.

Due to virtualised hardware access, the PS3 prevents access to the Graphics Processor (GPU) and video RAM, and does not allow computation to be offloaded to the GPU. The PS3 has an on-board Gigabit Ethernet Adapter; although it is not connected to the PCI bus, access is possible through the hypervisor. The Ethernet Adapter does have a dedicated DMA unit, which permits data transfer without the intervention of the PPE [5]. The approach used to enable development under Linux on the PS3 is described in detail in Appendix B

2.4 High Performance Scientific Computing

The Cell BE architecture, with its high performance design and parallelisation, is suited to high performance signal processing applications. The Cell BE has support for large page sizes to minimize memory latency for large data sets. Details of the Huge Translation Lookaside Buffer (HTLB) support that enable this can be found in Appendix B. The current Cell BE design is extremely efficient for single precision floating-point operations, although the accuracy limits imposed make this inappropriate for most scientific applications. The double precision floating-point necessary for scientific applications still exhibit a slight performance increase over conventional processors, although orders of magnitude slower than single precision performance. The Cell BE can achieve performance of over 200 GigaFLOP/s for single precision application, and over 14 GigaFlops for double precision operations. Modifications to the Cell BE architecture to increase the performance of double precision operations have been proposed by Williams et al. [40].

The IBM BladeCenter QS21[2] enables scaled performance and overcomes many of the disadvantages of the Sony Playstation 3 for use in scientific high performance computing. The Cell BE processor is used as a modular

[2]A server mainframe featuring the Cell Broadband Engine processor

component within the BladeServer to create a high performance symmetric multiprocessor (SMP) system.

Chapter 3

Polyphase Filter Banks

This chapter provides an introduction to the theory and mathematical development of Polyphase Filter Banks. Polyphase Filter Banks provide an efficient algorithm for digital filtering used in signal processing applications.

3.1 Finite Impulse Response Filters

By definition, the *Finite Impulse Response* (FIR) filter has a finite number of filter coefficients. Filtering is implemented as convolution of the filter coefficients with the signal samples:

$$y[n] = \sum_{j=0}^{N-1} x[j] * h[n-j], \tag{3.1}$$

where $x[n]$ are the original signal samples, $y[n]$ are the filter output samples and $h[n]$ are the coefficients of the impulse response for the filter with transfer function $H(z)$.

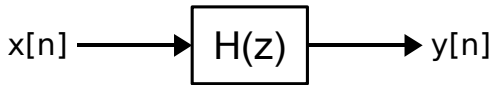


Figure 3.1: FIR filter block diagram

A major advantage of FIR filters is the property of linear phase.

3.2 Multirate Digital Signal Processing

A *multirate filter* makes use of different sample rates of the sampled input signal during the filtering process, using the operations of interpolation (increasing the sample rate) and/or decimation (decreasing the sample rate). Multirate systems are classified as *Linear Time-Varying* (LTV) systems and are of great practical importance as they can achieve much greater processing efficiency than single-rate systems.

3.2.1 Decimation

Decimation refers to the process of reducing the sampling rate of a signal [29]. This is usually implemented as a low-pass filtering operation followed by downsampling (discarding a number of samples).

The ratio of the input sampling rate to the output sampling rate is referred to as the *decimation factor* (denoted as M). As decimation is realised by discarding samples, the decimation factor is limited to an integer value, $M \in \mathbb{Z}$. Although an arbitrary fractional decimation factor is not possible, a rational decimation factor can be achieved through resampling, which involves combining decimation and interpolation to achieve the desired resampling factor. The downsampled signal can be represented mathematically as

$$y[n] = x[Mn], \tag{3.2}$$

where y is the downsampled signal, x is the original signal, and M is the decimation factor.

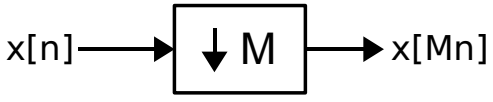


Figure 3.2: Decimation block diagram

In the frequency domain, the downsampling operation can be shown to be

$$Y(e^{j\omega}) = \frac{1}{M} \sum_{k=0}^{M-1} X(e^{j(\omega - 2\pi k)/M}), \tag{3.3}$$

where $Y(e^{j\omega})$ and $X(e^{j\omega})$ are the fourier transforms of $y[n]$ and $x[n]$ respectively.

The signal is passed through a low-pass filter before downsampling to ensure that the decimated signal obeys the *Nyquist* criterion, which states that the new sampling rate must be higher than the bandwidth of the signal to avoid aliasing.

Decimation is practically useful as a means of reducing the cost of processing (computational cost as well as data storage cost of implementation), which is proportional to the sampling rate.

3.2.2 Interpolation

The interpolation process increases the sampling rate of a signal by upsampling and then filtering the signal. Upsampling is achieved by inserting a number of zero-valued samples between each original sample of the signal. The upsampling step of interpolation produces a higher sampled signal that has an identical spectrum as the original signal over the original bandwidth in the frequency domain, but also produces spectral images centered on multiples of the original sampling rate. The undesirable spectral images are then eliminated through the use of a low-pass filter.

The ratio of the output sampling rate to the input sampling rate is referred to as the *interpolation factor* (denoted as L). The interpolation factor is also limited to an integer value, $L \in \mathbb{Z}$, although resampling can be used to achieve a rational interpolation factor. The upsampled signal can be represented

mathematically as

$$y[n] = \begin{cases} x[n/L] & \text{if } n \text{ is an integer multiple of } L \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where y is the upsampled signal, x is the original signal, and L is the interpolation factor.

Figure 3.3: Interpolation block diagram

In the frequency domain, the upsampling operation can be shown to be

$$Y(e^{j\omega}) = X(e^{j\omega L}), \tag{3.5}$$

where $Y(e^{j\omega})$ and $X(e^{j\omega})$ are the fourier transforms of $y[n]$ and $x[n]$ respectively.

Interpolation is useful in converting a signal for input into a system that requires a higher sampling rate, as well as being a stage in the resampling process.

The operations of downsampling and upsampling are a time varying, as a unit delay at the input will cause a non-unit delay of the output.

3.3 Channelisation

Channelisation is a technique used to separate a mixed channel signal into a number of single channels (channels). Various method for signal channelisation have been developed, such as the Hierarchical Multistage Method using a binary tree approach [13], Digital Down Conversion, frequency domain filtering using the FFT [30], and Polyphase Filter Banks [41]. The Polyphase Filter Bank channelisation algorithm, consisting of a polyphase filter bank

in combination with a Fast Fourier Transform (FFT), has been shown to be the most efficient technique for channelisation with large numbers of channels [30] [33].

3.4 Filter Banks

A filter bank is a set of filters with a common input or common output signal. The analysis filter bank splits the input signal $x(n)$ into a number of subband signals $x_k(n)$. The filters are typically uniformly distributed across the band of interest.

3.5 Polyphase Representation of Filter Banks

3.5.1 The Noble Identities

The *Noble Identities* illustrate an equivalent view of a decimator or interpolator. The order of filtering and resampling may be reversed, resulting in the equivalent systems for both the decimator, as shown in Figure 3.4, and the interpolator, as shown in Figure 3.5.

Figure 3.4: Exchanging the order of filtering and downsampling

Figure 3.5: Exchanging the order of upsampling and filtering

The noble identities allow for a more efficient computation as the filtering is applied to the signal at the lower sample rate.

3.5.2 The Polyphase Representation

If we examine the z-Transform of the analysis filter response

$$H(z) = \sum_{n=-\infty}^{\infty} h(n) z^{-n}, \tag{3.6}$$

we can split this up into sequences of even numbered coefficients and odd numbered coefficients

$$\begin{aligned} H(z) &= \sum_{n=-\infty}^{\infty} h(2n) z^{-2n} + \sum_{n=-\infty}^{\infty} h(2n+1) z^{-(2n+1)} \\ &= \sum_{n=-\infty}^{\infty} h(2n) z^{-2n} + z^{-1} \sum_{n=-\infty}^{\infty} h(2n+1) z^{-2n}. \end{aligned} \tag{3.7}$$

Thus, if we define

$$E_0(z) = \sum_{n=-\infty}^{\infty} h(2n) z^{-n}, \quad E_1(z) = \sum_{n=-\infty}^{\infty} h(2n+1) z^{-n}, \tag{3.8}$$

we are able to expressed $H(z)$ as

$$H(z) = E_0(z^2) + z^{-1} E_1(z^2). \tag{3.9}$$

Generalising this decomposition into M components

$$\begin{aligned} H(z) = &\sum_{n=-\infty}^{\infty} h(nM) z^{-nM} \\ &+ z^{-1} \sum_{n=-\infty}^{\infty} h(nM+1) z^{-nM} \\ &\vdots \\ &+ z^{-(M-1)} \sum_{n=-\infty}^{\infty} h(nM+M-1) z^{-nM} \end{aligned} \tag{3.10}$$

this can be compactly expressed as

$$H(z) = \sum_{l=0}^{M-1} z^{-l} E_l(z^M), \tag{3.11}$$

which is known as the Type 1 Polyphase representation, where

$$E_l(z) = \sum_{n=-\infty}^{\infty} e_l(n) z^{-n},$$

with

$$e_l(n) \triangleq h(Mn + l), \qquad 0 \leq l \leq M - 1$$

$E_l(z)$ are the polyphase components of $H(z)$ [37].
The noble identity shown in Figure 3.4 can now be used to move the filter in Equation 3.11 to the other side of the decimator so that filtering is performed at the lower sampling rate.

3.6 Implementation of Polyphase DFT Filter Banks

3.6.1 Polyphase Structure for Critically Sampled DFT Filter Banks

In the case where the number of channels in the filter bank is equal to the decimation factor ($M = K$), the filter bank is said to be *critically sampled*. The polyphase structure for the efficient realisation of critically sampled DFT filter banks has been shown by Rabiner and Crochiere [10].

$$\begin{aligned} X_k(m) &= \sum_{\rho=0}^{M-1} \sum_{r=-\infty}^{\infty} \bar{p}_\rho(r) W_M^{-k\rho} x_\rho(m - r) \\ &= \sum_{\rho=0}^{M-1} W_M^{-k\rho} [\bar{p}_\rho(m) * x_\rho(m)] \end{aligned} \tag{3.12}$$

where W_K is the k-th root of unity, $\bar{p}_\rho(m) = h(Mm - p)$ are the polyphase filter components, and $x_\rho(r)$ is the decimated channel signal.

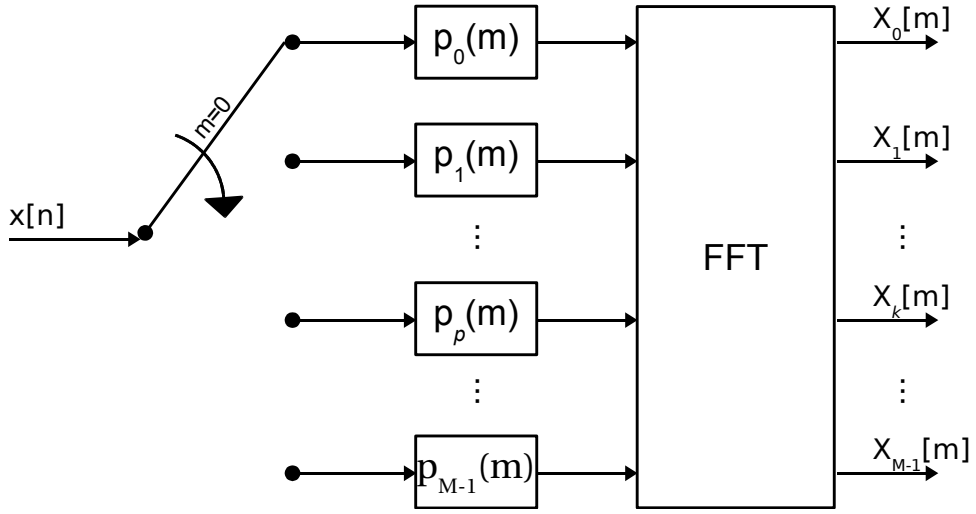


Figure 3.6: Critically Sampled DFT Filter Bank

3.6.2 Polyphase Structure for Oversampled DFT Filter Banks

For the oversampled case where the number of channels is an integer multiple of the decimation factor, i.e $K = MI$ for $I \in \mathbb{N}$, a polyphase structure provides an efficient realisation [10].

$$X_k(m) = \sum_{\rho=0}^{K-1} W_K^{-k\rho} [\sum_{r=-\infty}^{\infty} \bar{p}_\rho(m - rI) x_\rho(r)] \tag{3.13}$$

where W_K is the k-th root of unity, $\bar{p}_\rho(m-rI) = h(Mm-p)$ are the polyphase filter components, and $x_\rho(r)$ is the decimated channel signal.

3.7 The Fast Fourier Transform

The FFT is described in Algorithm 1, using a modified version of the Radix-2 Decimation in Time (DIT) FFT developed by Mullin and Small [25]. A more detailed introduction to the Fast Fourier Transform may be found in Appendix A.

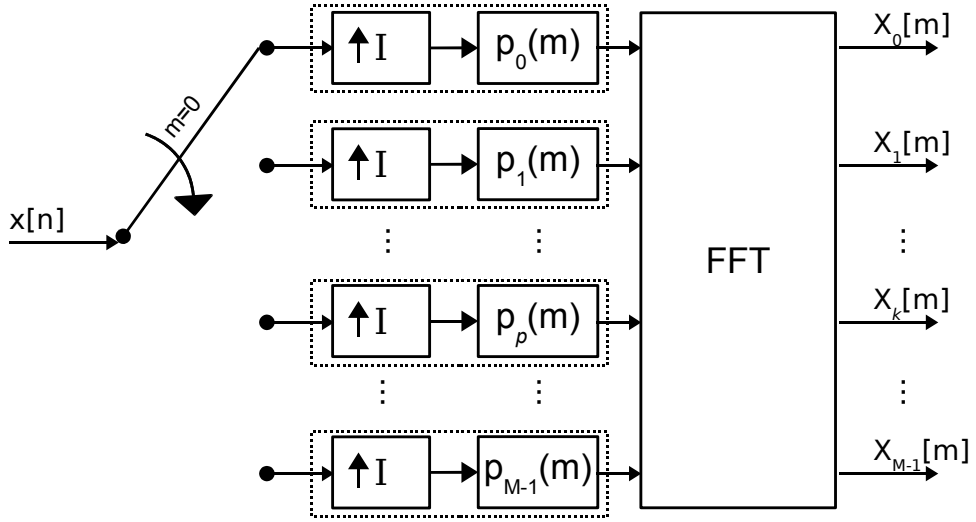


Figure 3.7: Oversampled DFT Filter Bank

Algorithm 1 Modified Fast Fourier Transform

Require: $x \in \mathbb{C}^n$ and $n = 2^t$

```
for q = 1 to t do
    L ← 2^q
    for j = 0 to L/2 − 1 do
        ω(j) ← e^{(2πij)/L}
    end for
    for k = 0 to n − 1 step L do
        for j = 0 to L/2 − 1 do
            c ← ω(j) × x(k + j + L/2)
            d ← x(k + j)
            x(k + j) ← d + c
            x(k + j + L/2) ← d − c
        end for
    end for
end for
```

Chapter 4

Implementation Details

In order to fully utilize the power of the Cell BE processor, the algorithm needs to be analysed for parallelisation, profiled and mapped to the appropriate elements of the processor.

4.1 Programming Model

There are multiple techniques that can be used to leverage the heterogeneous processor architecture of the Cell BE for efficient computation, and the programming model needs to be determined from a study of the algorithm that is required to be implemented. Examples of some programming models that may be employed on the Cell BE processor are:

- **Streaming (Pipelined) Model**: Each SPE functions as a stage in a pipelined algorithm. Data is streamed from the PPE into the first SPE where processing occurs. After data is processed in a particular SPE, it is streamed to the next SPE in the pipeline using DMA transfer. The last SPE in the pipeline will store the results back into main memory. In this model, a different program needs to be developed for each SPE containing the relevant algorithm for a particular stage in the pipeline. This model is suitable for algorithms that consist of a number of separable consecutive computationally intensive stages.

- **Parallel Model**: The parallel data model exploits data parallelism within the algorithm, and distributes equal workloads of the partition data across the available SPEs. The SPEs perform identical computation in parallel and return the results back into main memory on completion.

- **Functional offload**: The PPE runs a program, and offloads computationally intensive functional procedures to be performed on a SPE, which then returns the results to the PPE. In this model the algorithm needs to be separated into functional blocks and a separate program written for each function to be executed on a SPE.

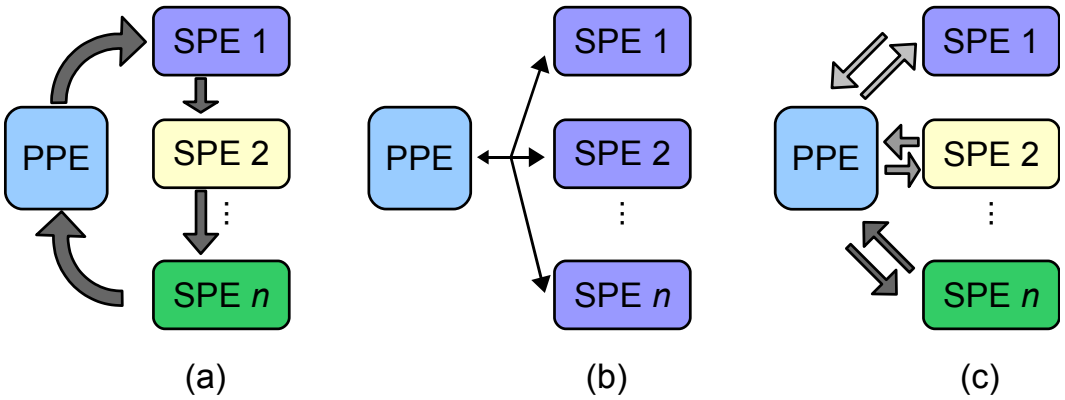


Figure 4.1: Examples of the programming models: *(a)* Streaming (Pipelined); *(b)* Parallel; *(c)* Functional Offload.

The model used for the implementation of Polyphase Filter Banks will exploit data parallelism and view the Cell BE processor as a homogeneous data parallel machine (Parallel Model) with heterogeneity in the PPE control code [39]. The computationally expensive code is partitioned to be executed in parallel across the available SPEs. The PPE program is written to format the data in memory, and for control and management of the SPEs. This programming model requires the efficient parallelization of the algorithm as well as static scheduling of DMA operations to marshal data between the

relevant local stores in the SPE and main memory. Two different programs are written, the PPE control program, compiled with the PPU compiler[1], and the SPE workload program that is compiled with the SPE compiler[2]. The SPE program is embedded within the PPE program, which loads the SPE program into threads that execute on each SPE at runtime.

Due to the limited size of the SPE Local Store, a large program may have code that does not entirely fit within this limit. To address this issue a technique called Code Overlays can be used [18]. Only a small segment of code is loaded into the SPE, known as the root segment, and this code then will load the relevant segments of code into an overlay region from main memory when needed. In this way, the entire SPE program does not need to be present in the local store at the same time.

4.2 Mapping the algorithm to the Cell BE Architecture

4.2.1 Algorithm complexity analysis

The Polyphase Filter Bank uniform DFT channelisation algorithm is $O(n \log_2 n)$, derived from the FIR filtering process that is $O(n)$, where n is the order of the filter, and the FFT of $O(n \log_2 n)$.

4.2.2 Algorithm partitioning

A control program was written and compiled for the PPE, which embeds the compiled SPE program within it and creates threads for each SPE that run the SPE program. The computationally intensive code is mapped to the SPE processors that are designed for high performance. The input data and filter coefficients are initially split into the K separate channels in memory.

[1]IBM XL C PPU compiler
[2]IBM XL C SPU compiler

The polyphase filter banks algorithm can be divided into two stages:

FIR filtering : Each channel is processed by upsampling (only in the case of over-sampled filter bank) and then convolving the filter tap coefficients with the channel coefficients.

FFT : For each output sample of the FIR filtering stage, a Fast Fourier Transform is performed across all the channels.

The input signal data for the filter is aligned in main memory by the PPE in the form of a 2-dimensional matrix, with the relevant channel-padding required to ensure quadword DMA transfers. This data is then partitioned to run across the six available SPEs, allocating $K/6$ channels to be processed in each SPE. This is illustrated in Figure 4.2 (Step 1). The FIR filtering stage (Step 2) retrieves the data for each channel from main memory, which it then processes. The results of each channel are then partitioned further across the six SPEs, and each segment, of size $\frac{N}{6K}$ is transfered to the relevant SPE via DMA. Once all filtering has been completed in all the SPEs, the FFT processing is started (Step 3), performing a 1-dimensional FFT on each column of data. The results of the FFT stage are then copied back to main memory via DMA (Step 4).

4.2.3 SPE program algorithm

This section describes the details of the algorithm implemented on the SPE for Polyphase filter bank processing. The program code that runs on each SPE is described in Algorithm 2. The algorithm runs on SPE n, where $0 \leq n \leq N-1$, assuming an input signal of length S, and filter coefficients of length F.

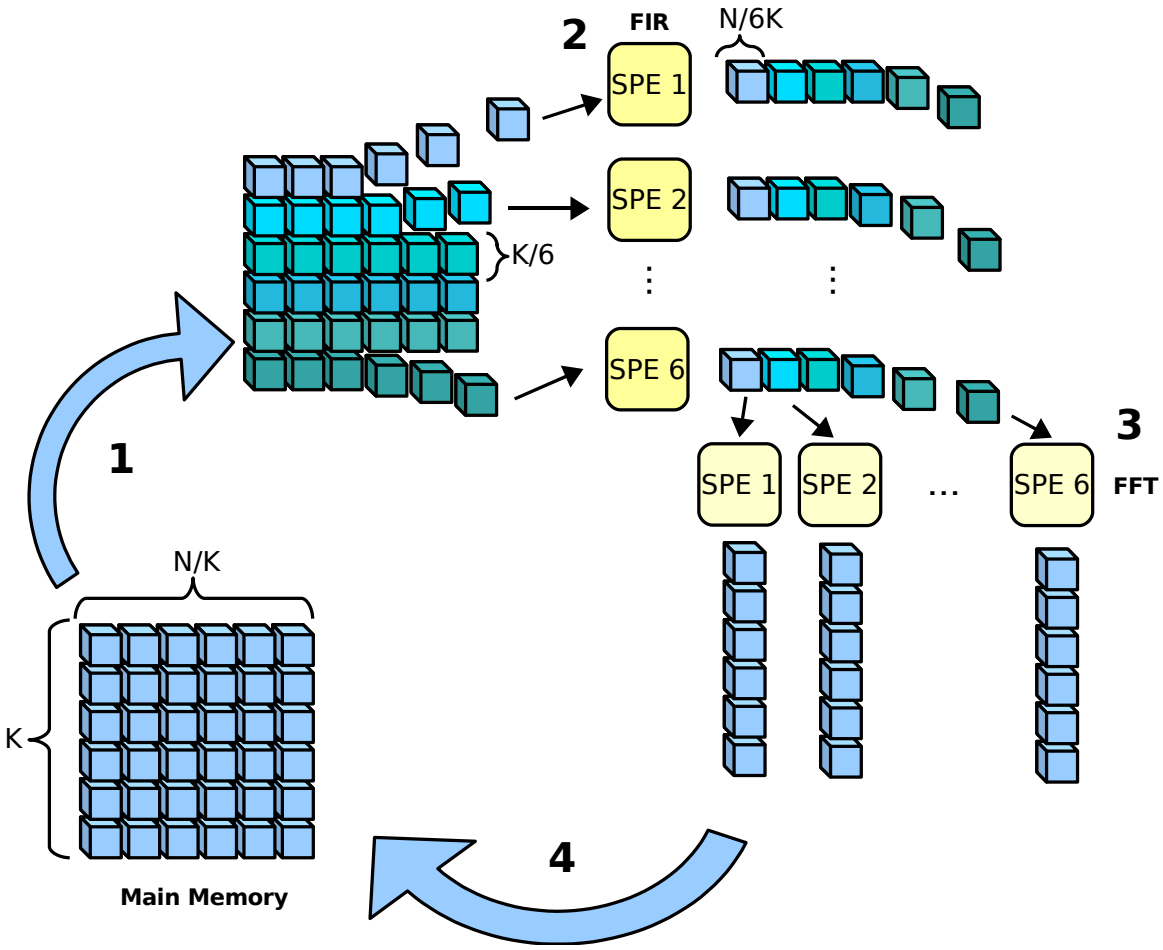


Figure 4.2: Algorithm stages for implementation using six SPEs

Algorithm 2 Program for SPE n

1: Wait for signal from PPE to begin FIR filter stage
2: Assign $i \leftarrow 0$
3: DMA request to get channel nK/N signal data into SBuffer i
4: DMA request to get channel nK/N filter data into FBuffer i
5: **for** $k = 1$ to $K/N - 1$ **do**
6: $j \leftarrow i \oplus 1$
7: DMA request to get channel $nK/N + k$ signal data into SBuffer j
8: DMA request to get channel $nK/N + k$ filter data into FBuffer j
9: Wait for DMA requests for Buffers i to complete
10: **for** $x = 0$ to $N - 1$ **do**
11: Calculate FIR filter results for values $x((S + F)/K - 1)N$ to $(x + 1)((S + F)/K - 1)N - 1$ by convolving SBuffer i and FBuffer i
12: DMA request to put FIR filter results into local store of SPE x
13: **end for**
14: $i \leftarrow j$
15: **end for**
16: Wait for DMA requests for Buffers i to complete
17: **for** $x = 0$ to $N - 1$ **do**
18: Calculate FIR filter results for values xS/N to $(x + 1)S/N - 1$ by convolving SBuffer i and FBuffer i
19: DMA request to put FIR filter results into local store of SPE x
20: **end for**
21: Send signal to PPE to indicate end of FIR filter stage
22: Wait for signal from PPE to begin FFT stage
23: **for** $k = 0$ to $(S + F)/K - 1$ **do**
24: Calculate K-point FFT for column k
25: DMA request to put FFT results for column k into main memory
26: **end for**
27: Send signal to PPE to indicate end of FFT stage

4.3 Communication Analysis

Communication between the individual processors is an important consideration on many-core architectures such as the Cell BroadBand Engine Architecture. The algorithm partitioning strategy determines the amount and type of communication that is required.

All DMA transfers are non-blocking and unordered, issued through the MFC using special instructions. The MFC instructions include a tag for DMA requests, and the MFC provides functions that stall the SPU until all requests that have a specified tag have completed. This mechanism is used in the polyphase filter bank implementation to synchronise the SPEs after the FIR filtering stage, as the FFT stage is only able to begin once all the channels have been processed and the data has been transfered to the relevant buffers within the local store of the SPE. The MFC must be manually controlled though intrinsics provided as language extensions and libraries in the IBM Cell SDK. The software control of the MFC allows the static scheduling to be optimised to perform efficiently for the particular implementation.

4.3.1 Inter-processor communication

Communication between the processor elements is facilitated by mailboxes, signal channels and the ability to transfer data between the Local Stores of the SPEs.

The communication required between processors in the implementation proceeds as:

- The PPE sends a mailbox message to the SPE, currently waiting on a mailbox.
- The SPE receives the mailbox message and begins the FIR filtering stage.

- The SPE pulls the required data for processing each channel from the main memory, performs the filtering, and transfers the results into the LS of the appropriate SPE.
- The SPE sends a flag via the mailbox to the PPE to indicate that it has completed the FIR filtering stage.
- The PPE sends a mailbox message to the SPE, currently waiting on a mailbox.
- The SPE receives the mailbox message and begins the FFT stage.
- The data required for processing has already been transfered into the local store. It is processed and the results are transferred back into main memory.
- The SPE sends a flag via the mailbox to the PPE to indicate that it has completed the FFT stage, and indicating the number of times the algorithm needs to be performed in order to completely process all the data.
- If there are no more repetitions of the algorithm, the PPE waits for the SPE to complete and then exits, otherwise the whole algorithm is repeated for the next segment of data.

Communication between the synergistic processors can be achieved with DMA by mapping the local store of an SPE to an effective address in the main memory through the PPE. The PPE can then provide the addresses of each SPE local store to all the the SPEs via the DMA or the mailbox mechanism. An information control structure is constructed in the PPE that contains information about the algorithm needed by the SPE, as well as memory addresses of the input and output buffers and the addresses of each of the local stores for the SPEs. This control structure is passed to each SPE as it starts so that it will be able to use the information to carry out the processing.

4.4 Performance Considerations

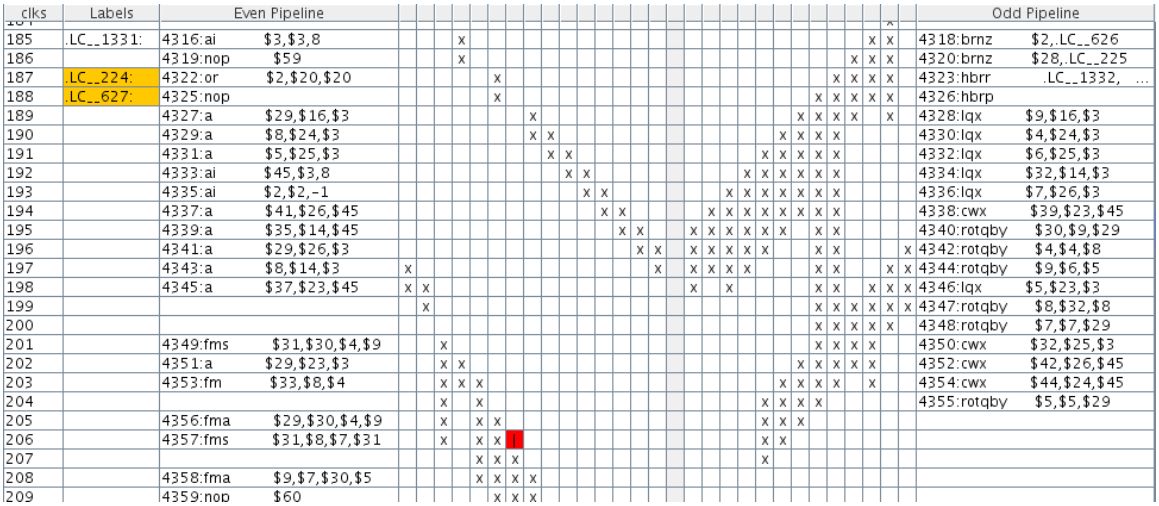
There are numerous architectural features that need to be taken into account in developing code for the Cell BE processor. An efficient load balancing strategy is required to ensure that all of the SPEs run at peak performance with as little computational overhead as possible and the effect of memory latency needs to be minimized. Software controlled explicit DMA scheduling for has a positive impact on power consumption and memory latency.

The model also needs to be aware of DMA latency issues, and so a double buffering scheme has been employed to hide the DMA latency. Although it is possible to initiate DMA transfer from the PPE via the SPE MFC proxy, this should be avoided as SPE-initiated DMA requests are performed in less cycles. The SPE command queue is double the size of the PPE proxy command queue, and the high number of SPEs relative to the single PPE further supports this reasoning.

Double-buffering techniques are used to hide memory latency, as DMA requests are asynchronously carried out at the same time as computation. In the first stage of the algorithm, the data is continuously brought into the local store of the SPE, and it is continuously put back into main memory at the end of the FFT stage. This large volume of communication degrades performance. Overlapping the memory transfers with computation within the SPE allows the latency to be hidden when the computation is more expensive than a single data transfer. This occurs with a larger input signals and a higher channel count in the Polyphase Filter Bank as the FIR filtering and FFT processing are able to hide the latency of the data transfers. Algorithms withpredictablee memory access patterns are able to exploit the software managed MFC to increase performance.

The SPE processor architecture implements Even and Odd pipelines (see Figure 2.2) in the datapath, where the each instruction in the instruction set is mapped to one or the other. To achieve maximum performance, instruc-

tions can be interleaved in such a way that both pipelines are filled, effectively allowing the SPE to execute two instructions simultaneously. Software pipelining is used to statically allocate the order of instructions which allows much greater control over the performance of the processor.



clks	Labels	Even Pipeline		Odd Pipeline	
185	.LC__1331:	4316:ai	$3,$3,8	4318:brnz	$2,.LC__626
186		4319:nop	$59	4320:brnz	$28,.LC__225
187	.LC__224:	4322:or	$2,$20,$20	4323:hbrr	.LC__1332, ...
188	.LC__627:	4325:nop		4326:hbrp	
189		4327:a	$29,$16,$3	4328:lqx	$9,$16,$3
190		4329:a	$8,$24,$3	4330:lqx	$4,$24,$3
191		4331:a	$5,$25,$3	4332:lqx	$6,$25,$3
192		4333:ai	$45,$3,8	4334:lqx	$32,$14,$3
193		4335:ai	$2,$2,-1	4336:lqx	$7,$26,$3
194		4337:a	$41,$26,$45	4338:cwx	$39,$23,$45
195		4339:a	$35,$14,$45	4340:rotqby	$30,$9,$29
196		4341:a	$29,$26,$3	4342:rotqby	$4,$4,$8
197		4343:a	$8,$14,$3	4344:rotqby	$9,$6,$5
198		4345:a	$37,$23,$45	4346:lqx	$5,$23,$3
199				4347:rotqby	$8,$32,$8
200				4348:rotqby	$7,$7,$29
201		4349:fms	$31,$30,$4,$9	4350:cwx	$32,$25,$3
202		4351:a	$29,$23,$3	4352:cwx	$42,$26,$45
203		4353:fm	$33,$8,$4	4354:cwx	$44,$24,$45
204				4355:rotqby	$5,$5,$29
205		4356:fma	$29,$30,$4,$9		
206		4357:fms	$31,$8,$7,$31		
207					
208		4358:fma	$9,$7,$30,$5		
209		4359:nop	$60		

Figure 4.3: SPE program with Dual Issue Pipelined instructions

The use of multi-threaded code must be carefully analysed as context switching on the SPE is very expensive: the full Local Store and DMA queues need to be saved and restored, causing a performance hit.

As there is no support for branch-prediction, it is important to eliminate branching in code as much as possible to avoid performance degradation. The SPE supports instructions for branch hints, allowing behaviour of the processor to be software controlled and optimised. Loops withing code should be unrolled to minimize branching, and this is made possible by the large register file provided by the SPE. Unrolled loops in code also provide additional instructions for the developer to efficiently schedule using further optimisation techniques.

The DMA only supports quadword aligned memory transfers of multiples of 16 bytes, and so data needs to be aligned in memory and sufficiently padded to ensure that the system bus will be able to efficiently execute DMA requests with no errors.

4.5 SPE code vectorisation

The Cell BE processor supports the Altivec Single Instruction Multiple Data (SIMD) instruction set. The SPE has a 128 entry 128-bit wide register file, and the SIMD vectorisation intrinsics allow for simultaneous operation on 4 single precision floating point values. In code that is not fully vectorised, scalar operations cause performance loss, as they are processed in the vector datapath due to lack of unaligned load support, with extra instructions needed to permute data for scalar access. The technique for vectorising both the inner and outer loops [34] was applied to the FIR filter algorithm. Vector instructions such as Multiply and Add (spu_madd) allow the FIR filtering to be efficiently computed on the SIMD architecture.

Reordering calculations to increase data locality is able to provide a performance increase in algorithms that are hard to optimise, such as the FIR (convolution) processing. Kraszewski [21] has shown that optimised AltiVec FIR code is 8 to 14 times faster than scalar code, in spite of only 4-way SIMD parallelism.

4.6 Implementation Limitations

Due to the nature of the Cell BE architecture, the code that was developed has limits on the size and type of inputs that are supported. The size of the Local Store memory available on each SPE limits the amount of data that can be stored for processing in a single pass. If the memory requirements of the algorithm exceed the capacity of the local store (250 Kilobytes), a

partition will need to be created and the data will be processed in multiple passes.

The algorithm will encounter problems when run with a large number of channels, as the implementation creates a storage buffer for the FFT processing (which is larger as it has to support complex numbers). This restricts the value of the number of channels to less than a few thousand ($K < 2048$). For a similar reason, in creating a buffer for the channel filtering, data sets that are heavily oversampled will cause problems as the upsampling stage needs to allocate memory to hold the upsampled signal, and this puts a limit on the length of the channel input when used with oversampling.

4.7 Compiler Optimisations

The code was compiled for the Cell BE using the IBM XLC C compiler provided with the IBM Cell SDK version 2.1. The efficiency of the compiled code will not be explored. The performance of the code relies on the assumption that the SPE intrinsics that are provided with the Cell SDK are compiled with reasonable efficiency.

4.8 The IBM Full System Simulator

The Cell Software Development Kit (SDK) includes the IBM Full System Simulator which is useful for both simulation and performance modelling. The algorithm was run on both the IBM Full System Simulator as well as the Playstation 3. The system simulator is useful for debugging during the development of the code, using a modified GDB[3] to support debugging of both PPE and SPE code. The profiling tools that are available with the SDK are very useful as a means of finding bottle-necks in the program and testing program performance.

[3]The GNU Debugger

4.9 Filter design

The filter selected for use in the PFB algorithm was designed using the classic windowed linear-phase FIR filter design method. A plot of the low-pass FIR filter of order 128, designed using the Hamming-window, with normalized cutoff frequency of 1/16,shown in Figure 4.4 along with the frequency response.

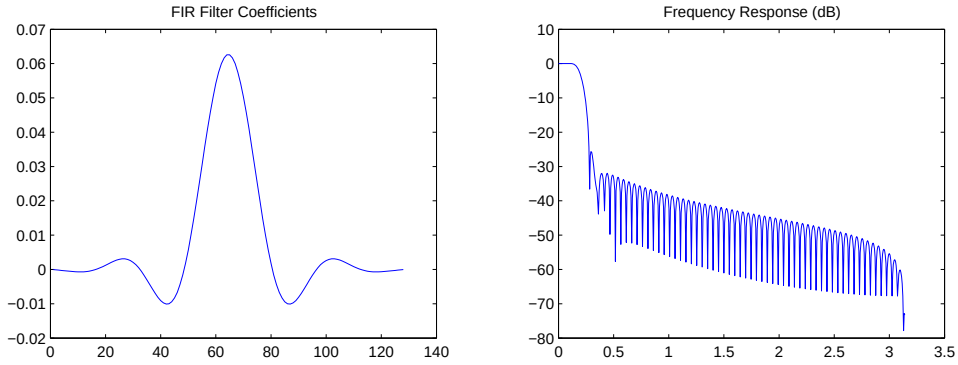


Figure 4.4: Windowed FIR filter

Chapter 5

Performance evaluation

5.1 Bandwidth and Latency

The system bandwidth plays an important role in assessing the performance of a processor. Often the memory latency is the bottleneck of the system design, as processing speeds are orders of magnitude faster than external memory bandwidth. Software controlled memory offers greater optimisation over the traditional hardware cache, making performance more predictable and efficient [20]. As mentioned in Section 2.1.3, the Cell BE processor features a 300 GB/s element interconnect bus (EIB) with system memory access at a peak transfer rate of 25.6 GB/s. To minimize the impact of the slower system memory, programs can try and keep communications and data transfers on chip by using mechanisms such as inter-processor DMA (Local Store to Local Store transfer between SPEs) and signalling (Mailboxes and signal registers) for small data communication.

The Cell BE processor supports up to 128 simultaneous DMA transfers between the local stores of the eight SPEs and main memory, which is significantly higher than that of a standard processor. Due to the memory structure of the Local Stores, main memory and the large register file on the SPE, as well as asynchronous DMA transfers, static scheduling is able

to hide memory latency by making use of double buffering, as described in Section 4.4.

With a transfer rate of 25.6 GB/s in the SPE, at least four single precision floating point operations would need to be performed on a floating point (4 byte) value in order to hide the communication latency. This leads to a total of 24 operations across 6 SPEs required to hide the latency. The ability to fully hide memory latency and achieve higher throughput would depend on the nature of the algorithm that is implemented. This is not always possible for some algorithms such as sparse linear algebra that have irregular memory access patterns and use large amounts of indirect memory addressing. For the Polyphase Filter Bank implementation, effective latency hiding would only be possible with larger data and larger numbers of channels.

Figure 5.1 shows the DMA profile for the Polyphase Filter Bank implementation on the SPE. The first stage of the algorithm issues DMA GET commands to retrieve the channel and filter data into the local store. After applying the FIR filtering to the channel, DMA PUT commands are used to transfer the results to the relevant SPE Local Store. After synchronisation, the final phase consists of calculating the FFT for each sample and then using the DMA PUT command to transfer the results back into main memory.

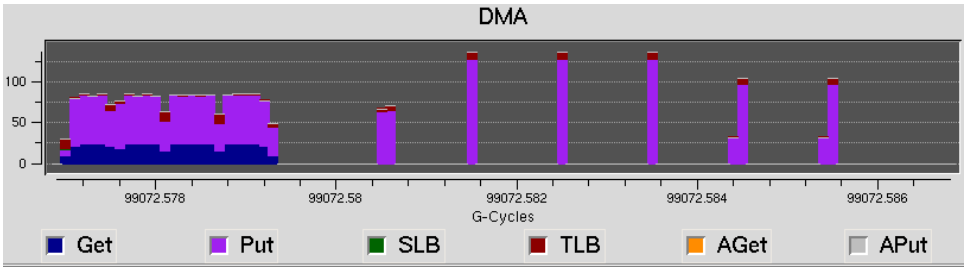


Figure 5.1: DMA profile

5.2 Performance Per Watt

An important metric often used to measure processor efficiency is the performance per Watt, which measures the performance throughput of the processor against its power comsumption [16].The architecture of the Cell BE processor emphasizes efficiency per watt, with a design that prioritises bandwidth over latency. The SPE architecture also favours peak computational throughput at the expense of higher complexity code. The memory hierarchy of the cell is software controlled, and is therefore able to achieve a higher degree of power efficiency. The Cell BE processor is thus able to deliver high frequency computation at a low voltage [36].

The Cell BE processor operates at about 30 Watts, and with the IBM BladeCenter QS21[1] the performance has been measured to achieve 1.05 GigaFLOP/s per watt. This gives the Bladecenter a peak performance of approximately 460 GigaFLOP/s, which can easily be scaled to achieve performance of up to 6.4 TeraFLOP/S in a single BladeCenter chassis, and over 25.8 TeraFLOP/s using a rack.

5.3 Precision and accuracy

5.3.1 Floating-point representation

Floating point numbers are represented digitally in the form $x = x_m 2^c$ where x_m is the mantissa ($\frac{1}{2} \leq |x_m| < 1$) and c is the exponent. The floating-point representation offers an advantage of a larger dynamic range than that of a fixed-point represenation. A disadvantage of the floating point representation is the tradeoff of dynamic range to the Signal-to-Noise ratio (SNR).

[1] A server mainframe featuring the Cell Broadband Engine processor

The Cell Broadband Engine supports limited IEEE compatible floating point arithmetic, including rounding, NAN-handling[2], overflow and underflow indication and exceptions [14]. To achieve high computation throughput, the SPU architecture is optimised for reduced area and power requirements, therefore floating point arithmetic is limited to only the most common modes and is not fully IEEE 754 standard compliant. This causes denormalised numbers to be automatically cleared to zero for all floating point operations on the SPU, as well as no support for trapping exception handling [27]. The extremely high performance single precision floating-point operations are further limited: only truncation rounding[3] is supported, and IEEE NaN and Inf are not recognized, causing overflow results to saturate to the largest representable positive or negative values. This design decision was driven by the target applications of high media and real-time game workload throughput and three-dimensional graphics operations. These applications do not require full IEEE compliance, as small errors are tolerable as a tradeoff for high performance throughput.

The double precision floating point operations are not fully pipelined, which results in a throughput that uses 7 clock cycles (1 instruction and 6 stall cycles) [19]. Single precision operations on the SPU have a theoretical throughput of 25.6 GigaFLOP/s and can be expected to run close to theoretical peak performance, whereas the double precision throughput rate is only about 1.83 GigaFLOP/s [23, 24], resulting in performance expectations of a factor of 14 below that of peak performance [39]. Computations that demand full precision accuracy will therefore have a peak system-wide performance of only 14.4 GigaFLOP/s on the Cell BE processor (almost 11 gigaFLOP/s on the PlayStation 3 with only six SPEs).

[2]Not-A-Number

[3]Rounding towards zero

The SPU does not support common saturating integer data types that are available on IBM VMX[4] architectures, as they cause a lower dynamic range which degrades results. The integer data types are extended for intermediate operations and are then performed without saturation, followed by saturating pack operations. Repeated round-off during computation is also avoided. This produces reliable results as well as conserving memory and reducing memory bandwidth [14].

5.3.2 Signal-to-Noise ratio

The floating point representation of numbers provides a way to trade off signal-to-noise ratio for an increase in dynamic range. For the n-bit floating point representation of a number, with $n - c$ bits in the mantissa (used for precision) and c bits in the exponent (used for extending the dynamic range), the dynamic range of the number can be calculated as

$$\begin{aligned} \text{Dynamic Range (dB)} &= 20 \log 2^{2^c} \\ &= 6.02 \times 2^c, \end{aligned} \tag{5.1}$$

with a Signal-to-Noise ratio equal to

$$\begin{aligned} \text{SNR (dB)} &= 20 \log 2^{n-c} \\ &= 6.02 \times (n - c) \end{aligned} \tag{5.2}$$

For applications such as signal processing where the dynamic range is unpredictable or large, the floating-point representation is preferable over the fixed-point representation. The IEEE floating-point format supported in the Cell BE processor is 32-bits wide, using 24 bits for the mantissa and the remaining 8 bits for the exponent. Therefore, in the case of the Cell BE processor SPU, the dynamic range offered for floating-point is 1541 dB, with a Signal-to-Noise ratio of 144 dB.

[4] Also known as AltiVec or Velocity Engine

5.3.3 Quantisation noise

Quantisation noise is introduced by the digital representation of a signal and results in irreversible errors in the signal. The quantization noise can be viewed as an additive noise source $e(n)$, that is uncorrelated with the original signal, as shown in Figure 5.2.

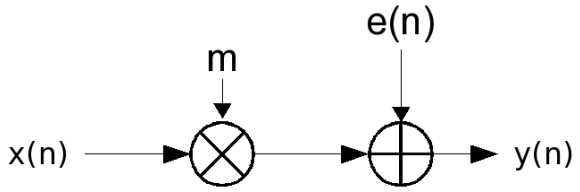


Figure 5.2: Noise model for quantization

Quantization noise is introduced in finite precision floating point addition and multiplication, and can be modeled as

$$Fl\{x_1 + x_2\} = x_1 + x_2 - (x_1 + x_2)n_a, \tag{5.3}$$

$$Fl\{x_1 x_2\} = x_1 x_2 - (x_1 x_2)n_p \tag{5.4}$$

where n_a and n_p are random variables with zero mean that are independent of other errors in the signal [12].
The variances are approximately

$$\sigma^2_{n_a} \approxeq 0.165 \times 2^{-2(n-c)}, \tag{5.5}$$

$$\sigma^2_{n_p} \approxeq 0.180 \times 2^{-2(n-c)} \tag{5.6}$$

where $n-c$ is the number of bits in the mantissa [35]. Therefore, for the Cell BE processor SPU where $n-c = 24$, the quantization noise exhibits variances of 5.8×10^{-16} and 6.4×10^{-16} for floating-point addition and multiplication respectively. A detailed model of round-off and scaling noise in multirate systems such as the polyphase filter bank can be found in Olsson et. al. [28].

Errors that in the coefficients of a digital filter will cause changes to the frequency and phase response characteristics of the filter. These errors cannot be avoided in a digital system, but can be reduced by using higher precision for the digital representation of the coefficients. This effect is more severe for Infinite Impulse Response (IIR) filters, as it could cause instability in the system by affecting the placement of the poles and zeros of the transfer function. In this implementation we are using FIR filters, therefore these errors can be overlooked.

5.4 Performance Results

5.4.1 Testing

The Polyphase Filter Bank algorithm was written for the standard Intel x86 architecture in order to provide a comparison with the Cell BE processor. I have chosen to keep much of the algorithm the same, with optimisations applied on the Cell BE processor code to allow the algorithm to take advantage of the architecture. The results have been generated on an Intel Core 2 Duo processor with a clock speed of 2.0 GHz and 2 Gb RAM. These results were compared against the code running on the Sony Playstation 3, as well as performance statistics generated from the IBM Full System Simulator.

5.4.2 Input signal

This section will examine the effect of the input signal on the performance characteristics of the algorithm. The size of the input signal will be measured against the speed of computation to obtain a growth rate estimation for the PFB algorithm on the Cell BE processor. As the size of the input increases such that the memory space in the SPE local store is no longer able to contain all the data required to process the PFB, the code splits the processing into smaller segments. The program then processes each segment through both stages of the PFB (the FIR filtering as well as the FFT) in a loop until all

of the data has been processed. This does not have a major performance impact, as the latency of the multiple stages of memory reading and write-back as the algorithm in the SPE is executed in a loop is hidden due to the high computational load brought about by the larger data set.

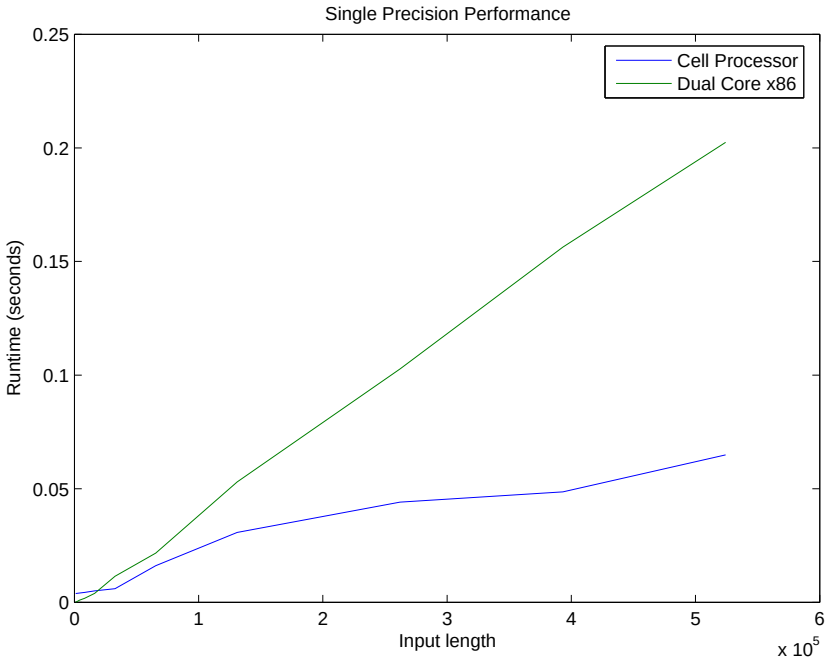


Figure 5.3: Single Precision Performance comparison

Figure 5.3 shows a comparison of the results for single precision computation as the input length increases. The performance of the code on the Cell BE processor grows much more efficient as the length of the input increases. A performance speedup of a factor of 3 over the standard processor can be observed with input length of 524288, and this speedup continues to grow as the graphs diverge. Figure 5.4 shows a comparison of the results for double precision computation as the input length increases. This follows the same trend as the results for the single precision. A performance speedup of a factor of 6 over the standard processor can be observed with input length of 262144, and this speedup continues to grow as the graphs diverge. Due to the

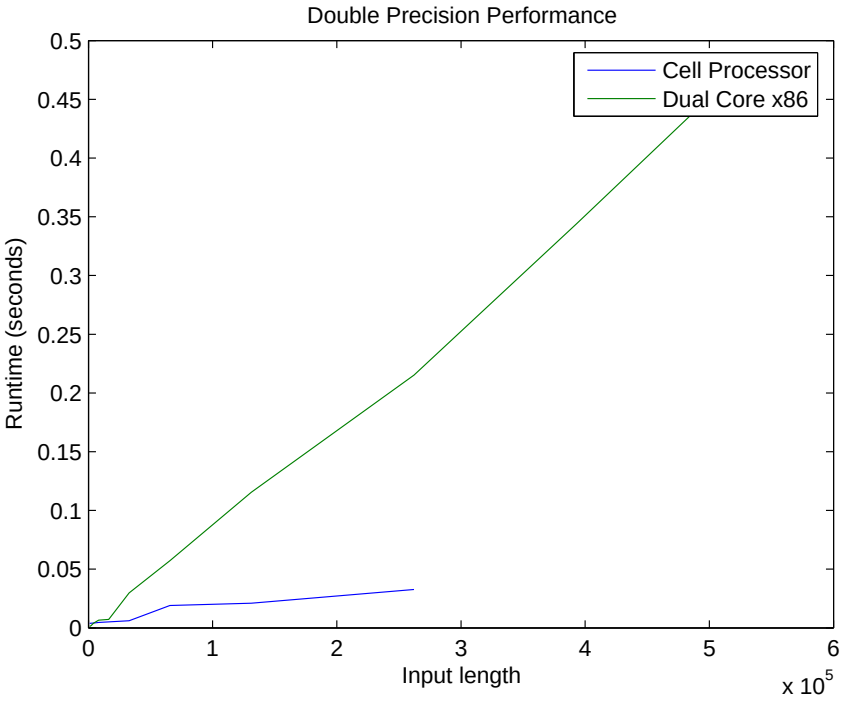


Figure 5.4: Double Precision Performance comparison

limited size of the local store, the algorithm could not be tested further for larger inputs in its current implementation. Figure 5.4 shows that the results for double precision could not be calculated for values larger than 262144 due to the larger memory requirements of the double precision representation. In order scale beyond these limits, additional techniques such as code overlays would need to be used, as well as a distributed FFT algorithm to partition the FFT processing between stages.

In Figure 5.5 a profile of the SPU cycle time is shown. From this figure it can be seen that on small input, the proportion of the CPU cycle time spend idle waiting for synchronisation is extremely large. In this case, the low number of input samples results in the computational time relatively low compared to that of overhead. As the input sample number increases, the SPU is able to hide most of the stalled latency as the computation time is relatively high compared to the overhead incurred.

An increase in the order of the filter used in the algorithm is not explored as it has the same effect as an increase in the signal length on the performance.

Figures 5.6, 5.7 and 5.8, show that the throughput rate achieved on the IBM Full System Simulator reaches only about 200MegaFLOP/s for single precision and 30MegaFLOP/s for double precision on a single SPE. This lower performance is due to the limit on the data set sizes that cause the algorithm to be unable to fully hide the communication latency. The overhead of the latency is not small enough relative to the processing workload.

5.4.3 Channel Count

The effect of the number of channels on the performance can be seen in Figure 5.9. As the number of channels grows, the processing load of each individual SPE grows slightly as the filtering algorithm uses a double buffered

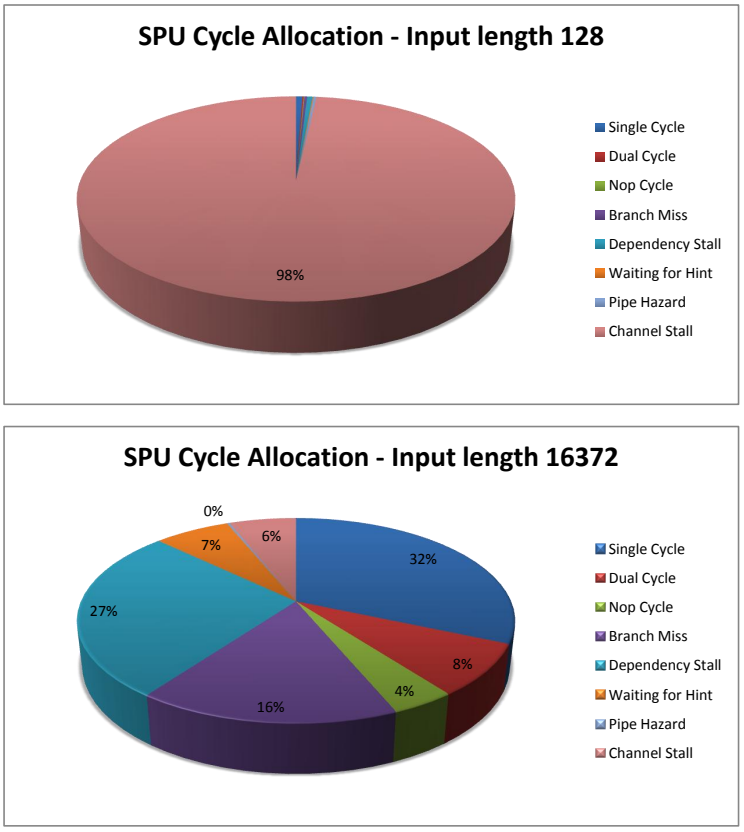


Figure 5.5: SPU Cycle Allocations

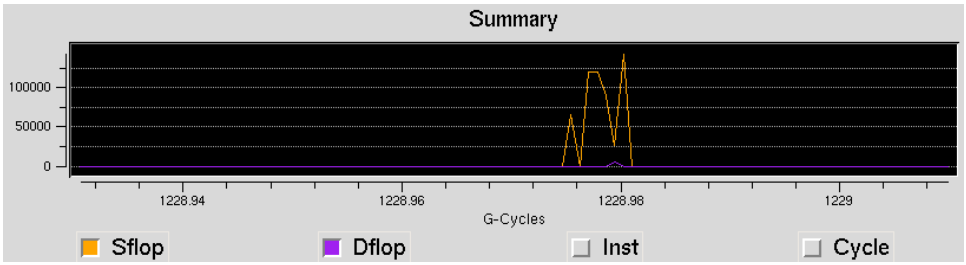


Figure 5.6: Single Precision SPU Performance

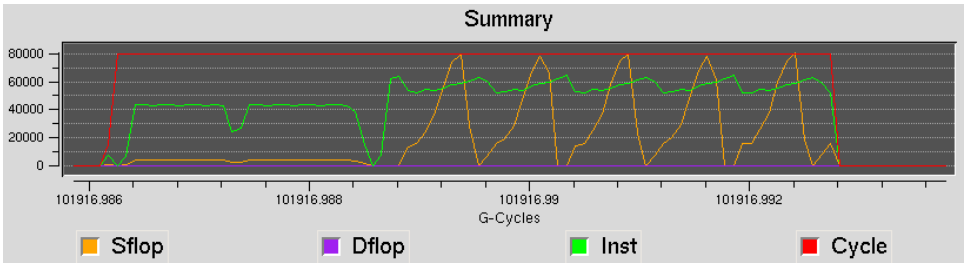


Figure 5.7: Single Precision SPU Profile

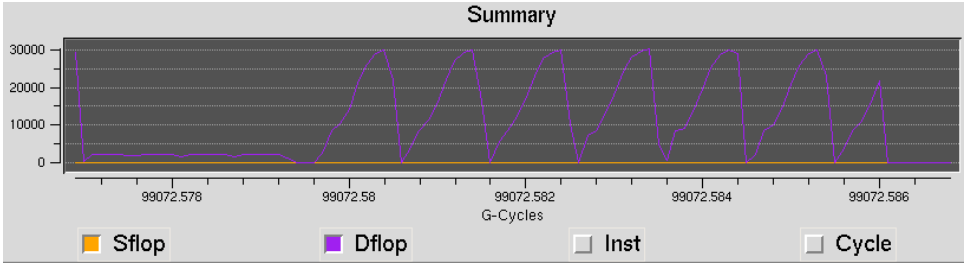


Figure 5.8: Double Precision SPU Profile

scheme to fetch the data for each channel from the main memory. the number of channels is limited by the size of the local store buffer, and could not be tested beyond 2048. The increase in performance over a wide range of channel values is very small, as the variation from 32 channels to 2048 channels only results in a difference of a single millisecond.

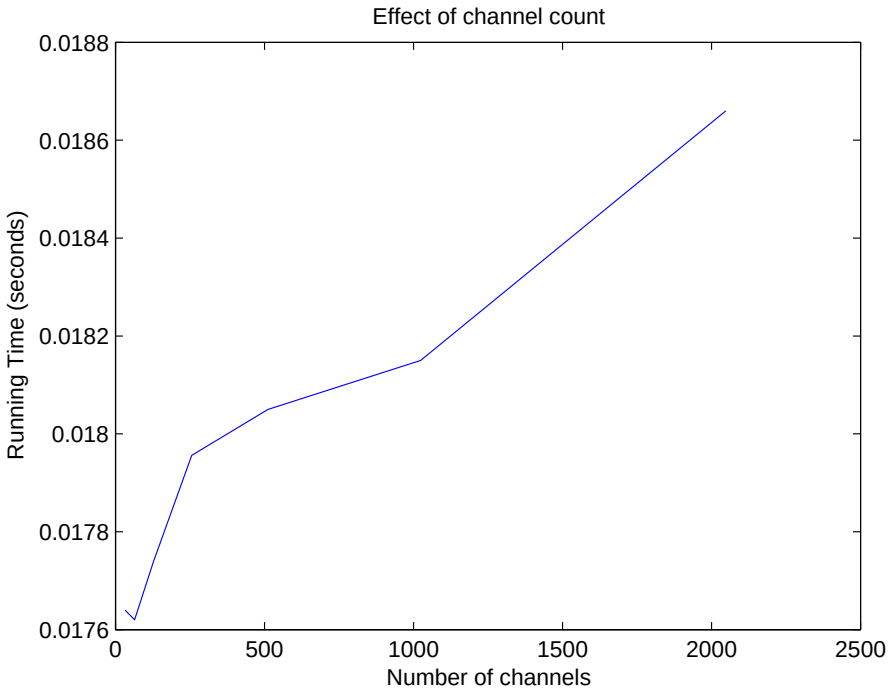


Figure 5.9: Single Precision Performance comparison

5.4.4 Oversampling ratio

In this implementation of the algorithm, there is a limit on the oversampling ratio as the storage space for the expanded signal for each channel assigned to an SPE for processing needs to collectively fit within the local store limits along with the code. As the system becomes more oversampled, the performance decreases at a high rate. This is due to the extra processing load required when filtering the upsampled signals.

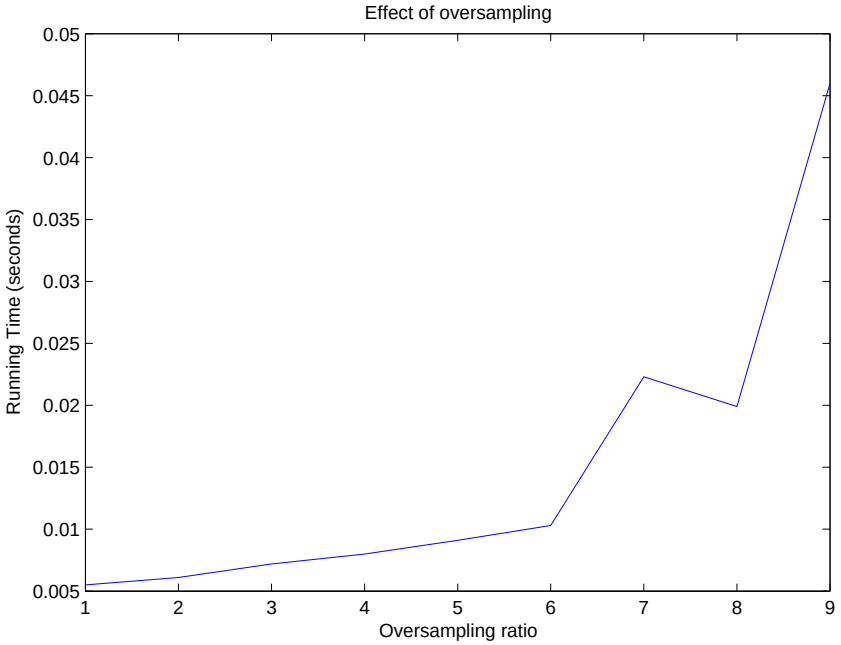


Figure 5.10: Effect of Oversampling Ratio on performance

5.5 Polyphase Filter Bank Response

The Polyphase Filter Bank algorithm is shown to be more effective at eliminating crosstalk between channels than the traditional FFT method of channelization. The Frequency response for a 32 Channel Twice Oversampled PFB, using the Hamming Windowed filter, is shown in Figure 5.11. Figure 5.11 shows the Frequency response for a 16 Channel Twice Oversampled PFB, using an equiripple filter designed using the McLellan-Parks algorithm, is shown in Figure 5.11.

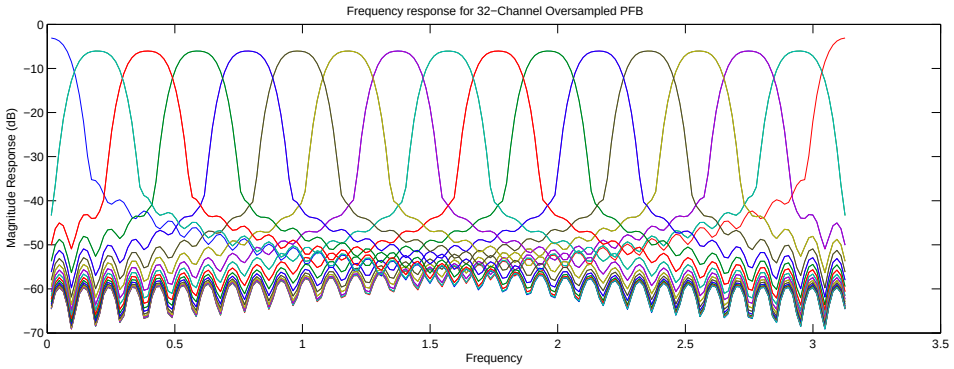


Figure 5.11: Polyphase Filter Bank Frequency Response for Windowed filter

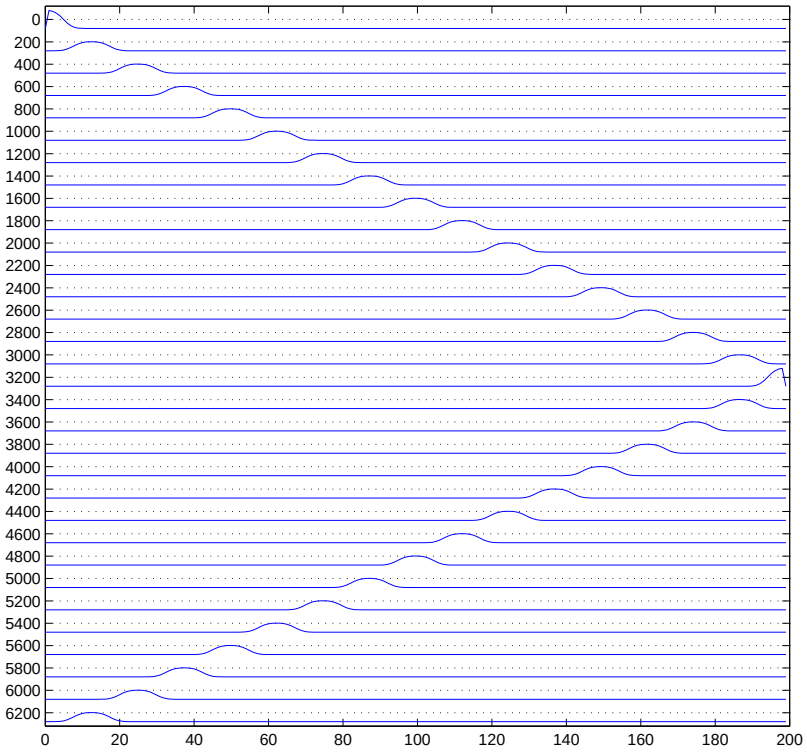


Figure 5.12: Polyphase Filter Bank Channels

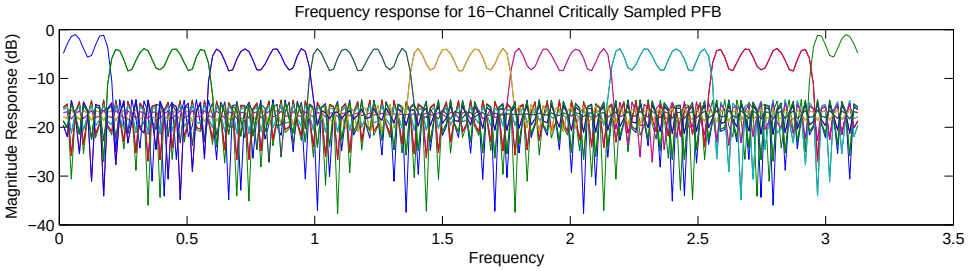


Figure 5.13: Polyphase Filter Bank Frequency Response for Equiripple filter

Chapter 6

Analysis

The Cell BE exhibits favourable performance over a traditional homogeneous multicore processors. The Polyphase Filter Bank algorithm was severely limited by the size of the local store in the SPE, but it shows a significant speedup over the range of inputs that were tested. The Cell BE processor performance should become particularly efficient for very large workloads, where the raw computational power of the SPEs will be able to effectively hide communications and memory latency. The Cell BE processor allows direct control over memory, which is an advantage over cached processors for algorithms that have predictable memory access patterns to achieve peak performance. The effectiveness of the Cell BE for high performance processing would be dependant on the nature of the algorithm.

The Sony Playstation 3 itself imposes further limits such as

- Slow main memory access through virtualised hardware.
- Limited main memory capacity on the PS3.
- GigaBit ethernet connection that will become a bottleneck in a cluster environment.
- Large performance difference between single precision and double precision computation.

The disadvantage of the Cell BE Processor is the lack of portability of code developed to run efficiently on it. The code needs to be highly optimised manually to run specifically for the Cell BE architecture, which takes a considerable investment of time to learn to use efficiently.

6.1 Further Research

This implementation focused on a limited implementation of the Polyphase Filter Bank algorithms. This implementation could be expanded to run in a cluster environment (using MPI), or on a SMP BladeServer system to enable it to processes larger data sets. Further exploration into signal processing applications and the implementation of the Polyphase Filter Bank within a software correlator.

As processor design turns towards many-core architectures, effective development techniques will need to be explored. These techniques need to be able to leverage the architectures efficiently for high performance computation of the most common classes of algorithms (the dwarfs [1]). As the Cell BE becomes a mature architecture, more libraries and support should become available that make it efficient to run most of the dwarfs efficiently on the Cell Broadband Engine Architecure.

Appendix A

The Fast Fourier Transform

This appendix describes the algorithm that is used to efficiently implement the Discrete Fourier Transform (DFT), known as the Fast Fourier Transform (FFT).

A.1 Discrete Fourier Transform

The *Discrete Fourier Transform* (DFT) consists of a mapping $\mathcal{F} : \mathbb{C}^n \rightarrow \mathbb{C}^n$ that transforms a discrete time signal into its discrete frequency domain representation. defined as for $j = 0, \ldots, n-1$,

$$y_j = \sum_{k=0}^{n-1} x_k e^{\frac{-2\pi ikj}{n}} \tag{A.1}$$

Using matrix notation [38], equation A.1 can be expressed as

$$y = F_n x,$$

where

$$F_n = (f_{kj}), \qquad f_{kj} = W_n^{kj} = e^{\frac{-2\pi ikj}{n}} \tag{A.2}$$

is the n-by-n DFT matrix and W_n is the n-th root of unity. The DFT exhibits a computation complexity of $O(n^2)$.

A.2 Fast Fourier Transform

The *Fast Fourier Transform* is a efficient way of calculating a n-point DFT from a pair of $\frac{n}{2}$-point DFTs. This technique exploits the realization that in viewing the DFT in terms of sparse matrix multiplication, only the diagonals of the blocks are used to decompose the matrices. The is known as the *Radix-2* FFT, and was first demonstrated by Cooley and Tukey [9] in 1965. The FFT exhibits a computation complexity of $O(n \log_2 n)$

A.3 The Cooley-Tukey Framework

The Cooley-Tukey Radix-2 DIT FFT in-place algorithm is described by Van Loan [38].

Algorithm 3 The Radix-2 Fast Fourier Transform

The initial bit permutation, $x \leftarrow P_n x$, can be calculated as in Algorithm 4

Require: $x \in \mathbb{C}^n$ and $n = 2^t$

$x \leftarrow P_n x$
for $q = 1$ to t **do**
  $L \leftarrow 2^q; \quad r \leftarrow n/L; \quad L_* \leftarrow L/2$
  for $j = 0$ to $L_* - 1$ **do**
    $\omega \leftarrow \cos(2\pi j/L) - i\sin(2\pi j/L)$
    for $k = 0$ to $r - 1$ **do**
      $\tau \leftarrow \omega \cdot x(kL + j + L_*)$
      $x(kL + j + L_*) \leftarrow x(kL + j) - \tau$
      $x(kL + j) \leftarrow x(kL + j) + \tau$
    end for
  end for
end for

Algorithm 4 FFT Bit Permutation

Require: $x \in \mathbb{C}^n$ and $n = 2^t$

Ensure: $x \leftarrow P_n x$

```
for k = 0 to n − 1 do
    j ← 0;    m ← k
    for q = 0 to t − 1 do
        s ← floor(m/2)
        j ← 2j + (m − 2s)
        m ← s
    end for
    if j > k then
        x(j) ↔ x(k)
    end if
end for
```

Algorithm 5 Matrix Form of The Radix-2 Fast Fourier Transform

The Radix-2 FFT algorithm may alternatively be represented in matrix form as follows:

Require: $x \in \mathbb{C}^n$ and $n = 2^t$

```
x ← P_n
for q = 1 to t do
    L ← 2^q;    r ← n/L
    x_{L×r} ← B_L x_{L×r}
end for
```

where

$$B_L = \begin{bmatrix} I_{L_*} & \Omega_{L_*} \\ I_{L_*} & -\Omega_{L_*} \end{bmatrix}, \qquad L = 2^q, \quad r = n/L, \quad L_* = L/2$$

I_{L_*} is the L_*-dimensional Identity matrix, and Ω_{L_*} is a diagonal matrix with diagonal elements $1, \omega_L, \ldots, \omega_L^{L_*-1}$.

Appendix B

Configuration of the Sony Playstation 3

This appendix will describe the process of configuring the Sony Playstation 3 by installing the Linux operating system and the IBM Cell 2.1 SDK, as well as the development tool-chain that was used to compile, debug and test the program on the Cell BE.

B.1 Installing Linux

The Ubuntu operating system was selected to be installed on the PS3. A custom kernel was needed to support the PS3 virtualized hardware layer. The Cell BE PPE processor has a 64-bit PowerPC core, and is capable of running the PowerPC (ppc) version of any linux distribution. The linux kernel has had support for the Cell BE since version 2.6.16, which includes the SPU filesystem. Advice and step by step instruction can be found from the PSUbuntu[1] community.

[1] http://www.psubuntu.com

B.2 Huge TLB support

The Cell BE SDK supports the huge translation lookaside buffer (TLB) filesystem, which allows a large continuous memory area of up to 16MB to be allocated. The large memory area is useful for applications with large data requirements. By default, in the binary distribution of the kernel provided with Ubuntu, support for huge TLB pages has been disabled. This means that the kernel needs to be recompiled to support the huge TLB psuedo-filesystem (*hugetlbfs*). Allocating large pages has a speed advantage over allocating memory from the heap in the PPE. From the program code, a call to the linux function *mmap* will allocate a huge page and avoid TLB misses on the processor.

B.3 The IBM Cell SDK

IBM has release the Cell SDK (currently version 2.1), which provides all the development tools that are needed to compile, debug and test code for the Cell BE. The bus width of the PowerPC core is 64 bits, so compilation is directed at the 64-bit target architecture.

Glossary

CESOF Cell Embedded SPE Object Format

DMA Direct Memory Access

DSP Digital Signal Processing

EIB Element Interconnect Bus

FIR Finite Impulse Response

FFT The Fast Fourier Transform

GigaFLOP/s Giga-floating point operations per second

LS Local Store

MFC Memory Flow Controller

PFB Polyphase Filter Bank

PPE Power Processing Element

PS3 Sony Playstation 3

PSD Power Spectral Density

RISC Reduced Instruction Set

SIMD Single Instruction Multiple Data

SNR Signal-to-Noise Ratio

SPE Synergistic Processing Element

SPU Synergistic Processing Unit

TeraFLOP/s Tera-floating point operation per second

XDR Extreme Data Rate memory

Bibliography

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical report, University of California, Berkeley, December 2006.

[2] J. Bunton. An Improved FX Correlator. *ALMA memo 342*, December 2000.

[3] J. Bunton. Multi-resolution FX Correlator. *ALMA memo 447*, February 2003.

[4] J. Bunton and R. Navarro. DSN Deep-Space Array-Based Network Beamformer. *Experimental Astronomy*, 17(1-3):299–305, June 2004.

[5] A. Buttari, J. Dongarra, and J. Kurzak. Limitations of the PlayStation 3 for High Performance Cluster Computing. Technical report, University of Tennessee, Knoxville, 2007.

[6] A. Buttari, P. Luszczek, J. Kurzak, J. Dongarra, and G. Bosilca. SCOP3: A Rough Guide to Scientic Computing On the PlayStation 3. Technical report, University of Tennessee Knoxville, May 2007.

[7] Y. Chikada, M. Ishiguro, H. Hirabayashi, M. Morimoto, K. I. Morita, K. Miyazawa, K. Nagane, K. Murata, A. Tojo, S. Inoue, T. Kanzawa,

and H. Iwashita. A Digital FFT Spectro-Correlator for Radio Astronomy. In J. A. Roberts, editor, *Indirect Imaging*, pages 387–404. Cambridge University Press, 1984.

[8] L. Cico, R. Cooper, and J. Greene. *Performance and Programmability of the IBM/Sony/Toshiba Cell Broadband Engine Processor*, 2006.

[9] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[10] R. E. Crochiere and R. R. Rabiner. *Multirate Digital Signal Processing*. Prentice Hall Signal Processing Series. Prentice Hall, 1983.

[11] A. T. Deller, S. J. Tingay, M. Bailes, and C. West. DiFX: A Software Correlator for Very Long Baseline Interferometry Using Multiprocessor Computing Environments. *The Publications of the Astronomical Society of the Pacific*, 119(853):318–336, March 2007.

[12] P. S. R. Diniz, E. A. B. da Silva, and S. L. Netto. *Digital Signal Processing: System Analysis and Design*. Cambridge University Press, 2002.

[13] H. Goeckler. A modular multistage approach to digital FDM demultiplexing for mobile SCPC satellite communications. *International Journal of Satellite Communications*, 6:283–288, July-September 1988.

[14] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *IEEE Micro*, 26(2):10–24, April 2006.

[15] H. P. Hofstee. Introduction to the Cell Broadband Engine. Technical report, IBM Corporation, 2005.

[16] H. P. Hofstee. Power Efficient Processor Architecture and The Cell Processor. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. IEEE Computer Society, 2005.

[17] IBM. *Cell Broadband Engine Programming Tutorial v2.1.* IBM, March 2007.

[18] IBM. *Cell Broadband Engine Software Development Kit 2.1 Programmer's Guide*, second edition, March 2007.

[19] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development.*, 49(4/5):589–604, July-September 2005.

[20] M. Kondo, H. Okawara, H. Nakamura, T. Boku, and S. Sakai. SCIMA: A novel processor architecture for high performance computing. *4th International Conference on High Performance Computing in the Asia Pacic Region*, 2000.

[21] G Kraszewski. Fast FIR Filters for SIMD Processors With Limited Memory Bandwidth. In *XI symposium AES: "New trends in audio and video"*, September 2006.

[22] M. Krishna, A. Kumar, N. Jayam, G. Senthilkumar, P. Baruah, R. Sharma, S. Kapoor, and A. Srinivasan. Feasibility study of MPI implementation on the heterogeneous multi-core cell BE architecture. In *19th Annual ACM Symposium on Parallel Algorithms and Architectures (ISPA07)*. ACM, 2007.

[23] J. Kurzak and J. Dongarra. Implementation of the Mixed-Precision High Performance LINPACK Benchmark on the CELL Processor. Technical Report UT-CS-06-580, University of Tennessee Computer Science Tech Report, September 2006.

[24] J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra. Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy. Technical Report UT-CS-06-574, University of Tennessee Computer Science Tech Report, April 2006.

[25] L. R. Mullin and S. G. Small. Four Easy Ways to a Faster FFT. *Journal of Mathematical Modelling and Algorithms*, 1:193–214, 2002.

[26] K. O'Conor, C. O'Sullivan, and S. Collins. Isosurface Extraction on the Cell Processor. In *Proceedings Eurographics Ireland*, pages 57–64, 2006.

[27] H. Oh, S. M. Mueller, C. Jacobi, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. H. Dhong. A Fully Pipelined Single-Precision Floating-Point Unit in the Synergistic Processor Element of a CELL Processor. *IEEE Journal of Solid-State Circuits*, 41(4):759–771, April 2006.

[28] M. Olsson, P. Lowenborg, and H. Johansson. Scaling and Roundoff Noise in Multistage Interpolators and Decimators. In *Proceedings of the 4th International Workshop Spectral Methods Multirate Signal Processing*, September 2004.

[29] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, New Jersey, 1992.

[30] L. Pucker. Channelization Techniques for Software Dened Radio. In *Proceedings of SDR Forum Conference*, November 2003.

[31] L. Rabiner, J. McClellan, and T. Parks. FIR Digital Filter Design Techniques using Weighted Chebyshev Approximation. *Proceedings of the IEEE*, 63(4):595–610, April 1975.

[32] J. D. Romney. Theory of correlation in VLBI. In J. A. Zensus, P. J. Diamond, and P. J. Napier, editors, *Very Long Baseline Interferometry and the VLBA*, volume 82 of *Astronomical Society of the Pacific Conference Series*, 1995.

[33] G. Savir. Scalable and Recongurable Digital Front-End for SDR Wideband Channelizer. Master's thesis, Delft University of Technology, 2006.

[34] A. Shahbahrami, B. H. H. Juurlink, and S. Vassiliadis. Efficient Vectorization of the FIR Filter. In *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005*, pages 432–437, November 2005.

[35] L. M. Smith, B. W. Bomar, R. D. Joseph, and G. C. Yang. Floating-point Roundoff Noise Analysis of Second-Order State-Space Digital Filter Structures. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(2), 1992.

[36] O. Takahashi, S. R. Cottier, S. H. Dhong, B. Flachs, and J. Silberman. Power-Conscious Design of the CELL Processor's Synergistic Processor Element. *IEEE Micro*, 25(5):10–18, 2005.

[37] P. P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice Hall Signal Processing Series. Prentice Hall PTR, 1993.

[38] C. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[39] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. Scientic Computing Kernels on the Cell Processor . *International Journal of Parallel Programming*, 35(3):263–298, June 2007.

[40] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. A. Yelick. The potential of the cell processor for scientific computing. In *Proceedings of the 3rd conference on Computing Frontiers*, pages 9–20, 2006.

[41] K. C. Zangi and R. D. Koilpillai. Efficient filterbank channelizers for software radio receivers. *ICC 98 IEEE International Conference on Communications*, 3:1566–1570, June 1998.