# Final Project: Cell Processors

## ECE 568: Advanced Microprocessor Architecture, Spring 2016

Anthony Gallotta
agallo4@uic.edu

Vishal Mishra
vmishr2@uic.edu

## ABSTRACT

In September 2006, IBM released its first implementation of the Cell Broadband Processor architecture, the first commercial heterogeneous CPU architecture. While originally designed for the PlayStation 3, the Cell quickly gained popularity in the scientific and high performance computing (HPC) communities thanks to its impressive performance in single precision (SP) floating point computations, low power consumption, and low cost. Several of the top performing supercomputers of the late 2000s utilized the Cell extensively as well. IBM released a 2nd implementation of the Cell architecture, the PowerXCell 8i, in 2008, but this became obsolete quickly, and in 2009 they announced that the Cell line would be discontinued. While most people subjectively agree that the Cell became obsolete due to the rise of GPUs, to the authors' knowledge there have been no studies directly comparing Cell performance to GPUs manufactured at the same technology scale, and with similar power and area budgets. This paper attempts to quantify the reasons the Cell architecture is no longer being produced, in terms of performance and energy efficiency. In addition, we discuss some of the architectural principles that have lived on, those that haven't, and why. To do this, we create several models that resemble the Cell architecture. We model an architecture similar to the Cell using GPU cores that resemble the Cell SPEs, but with the memory structure of the Cell. We compare the performance of our model to that of a commercial GPU. We also construct a power and area model for the Cell in order to speculate on how the Cell architecture could have scaled beyond 45nm technology, and how it might compare to GPUs on the same scale.

## 1. INTRODUCTION

With a 4 year and $400 million budget, Sony teamed up with IBM and Toshiba around 2000 to form STI, a group whose goal was to design a new processor for the PlayStation 3, targeting a performance improvement of at least 100x over the PlayStation 2 [13]. With a gaming system as its primary target, the main objectives for the processor were to support high performance on multimedia applications, real time user and network responsiveness, and to be able to apply the architecture to other systems, having a lifetime beyond a single gaming console. The biggest hurdles in achieving these objectives were expected to be memory speed and power constraints. After evaluating several design options, the developers decided that the chip would have one larger core based on IBM's Power Architecture, and a number of low power "synergistic" processors with simple architectures and SIMD instruction sets.

IBM released 2 implementations of the Cell architecture, the first of which was the Cell Broadband Engine (BE), built on a 90nm scale and used in the PlayStation 3. IBM introduced a second and final implementation of the Cell architecture specifically for their Roadrunner supercomputer, the PowerXCell 8i, which has enhancements to the DP pipeline allowing peak performance of up to 108.8 GFLOPs/s, a 7x improvement over the Cell BE [8]. This was a much needed improvement for HPC applications, however it didn't come until 2 years after the initial introduction of the Cell, and by this time GPGPUs were emerging as another option for HPC. The PowerXCell 8i also has support for up to 32GB of memory compared to the Cell BE's limitation of 2GB, but is largely unchanged other than these factors and the smaller 65nm process [8]. Thanks to its low cost in terms of both purchase price and power, the Cell quickly gained popularity in the HPC community. This is evident from supercomputers like the U.S. Air Force's Condor [16] that used a mix of Cell, GPU, and homogeneous multicore processors, and IBM's Roadrunner [8] that primarily uses PowerXCell 8i cores, with support from Opteron cores for coordination.

In 2009, IBM announced that their PowerXCell 8i chip would be the last of its kind, scrapping development plans for a 32 core version. We attribute this to several causes, primarily general purpose graphics processing units (GPGPUs) becoming more widely available, with more portable instruction sets, higher potential for parallelism, and reduced programming complexity.

In the subsequent sections, we present a brief summary of the Cell architecture. Next, we survey prior research related to our topic, demonstrating the value of the Cell at it's peak popularity, as well as the factors in its decline. We then present a strategy for modeling the performance, power, and area of the Cell architecture in order to speculate on how it may have evolved to compete with GPGPUs. Finally, we analyze the results of our models, and draw some conclusions from both our models and our survey of existing research.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

## 2. BACKGROUND

This section presents an overview of the architecture and core design principles for the Cell.

### 2.1 Cell Architecture

The architecture for the Cell is called the Broadband Processor Architecture, although the term "Cell" is commonly used to refer to either the architecture, or an implementation of the architecture, depending on the context. The Cell is a heterogeneous architecture that consists of a single large core, the "Power Processing Element" (PPE), and many smaller, SIMD cores, the "Synergistic Processing Elements" (SPEs). This section focuses on high level architecture components, such as the cores themselves, memory hierarchy, and interconnects. It is noteworthy though, that no detail was neglected in the first Cell implementation. Hardware components were optimized all the way down to the gate level for the first generation of the Cell. Along with other components, STI developed several custom latches and flip flops to improve power efficiency and delay, and observed correct operation at frequencies approaching 5GHz [17].

#### *2.1.1 Power Processing Element*

The PPE is a dual-threaded, dual-issue (in order) 64-bit Power Architecture compliant core. This processor has a traditional cache hierarchy and 2 hardware threads. In typical HPC implementations it is used only to control the 8 synergistic processing elements (SPEs) and run an operating system, however it can also be used to run applications that have not been updated to utilize the SPEs, are highly serial, contain complex control flow logic, or are otherwise not suitable for execution on the SPEs.

#### *2.1.2 Synergistic Processing Elements*

Each SPE has direct memory access (DMA) to main memory, and a fast local memory, as well as a large 128 register file. All 8 SPEs are connected via a 25.6 GB/s element interconnect bus (EIB), which also provides connections to main memory and I/O controllers [13]. A SIMD instruction set (with scalar support) was chosen for the SPE, as it had become the dominant ISA for CPU media support at the time, and it is optimized for power and performance on media and compute-intensive operations.

The largest component in the SPE is the local store (implemented with SRAM), which is where the unit loads data and instructions from, and what differentiates the SPE architecture most from traditional cores that use an L1 cache. An SPE issues DMA commands (up to 16 may be outstanding) in order to load data from main memory (or another SPE core, if it has permission) to its local store. Since the local store must be explicitly managed by software, memory latency is constant, compared to a hardware managed cache, where it is unknown to the programmer what may be in the various cache levels at any given time. While this adds programming complexity, it also results in more predictable performance, and lower power consumption, which is why Cell architects chose this design. This generally results in a much higher memory bandwidth as well, since the software is able to load more useful data then a cache. Branch prediction logic is lacking in the hardware layer, with only a simple static predictor, and must be implemented in software if a better predictor is desired [11]. Mispredicted branches can be costly as well, with a pipeline flush requiring 18 cycles, so software should be cautious of this. This simplicity again results in higher area and power efficiency. The SPE is an in order processor, and does not perform any register renaming, although it can issue two instructions per cycle if execution units are available for both.

## 3. RELATED WORK

Numerous studies have been published ([9] [14] [19]) comparing the performance of the Cell to homogeneous and single core architectures. In [10], published by IBM, the authors compare performance of scientific applications on the Cell against competitive superscalar and SIMD architectures and demonstrate its superiority to both, but GPUs are not considered. In [12], an article published by IBM before the first release of the Cell BE, the authors explore performance per transistor as an efficiency metric. This metric approximates performance per Watt, assuming a constant power draw per transistor, which is reasonable to make under CMOS technology [12]. They detail the design trends of architectures that tend to decrease this efficiency, and advocate architectural details like the local store used in the Cell to improve performance per transistor. While all of their logic makes sense intuitively, the authors fail to quantify any of their statements with experimental results.

High performance LINPACK (HPL) is a benchmark that solves a dense system of linear equations, and is commonly used to measure performance of supercomputers for rankings such as the TOP500 [5] and Green 500 [3]. The U.S. Air Force's Condor Cluster was designed to be competitive with top supercomputers in terms of raw performance, but with much greater energy efficiency. It's target applications were also optimized to use SP computations. To achieve its goals, the system was designed to have 78 compute nodes, each composed of 2 NVIDIA GPGPUs, 1 Intel Xeon X5650 (a 12 core chip), and a connected subcluster of 22 PS3s. In [16], the authors run HPL on a cluster of 2 PS3s and calculate an energy efficiency of 52 MFLOPS/W, which is sufficient to place the PS3 nodes of the Condor in the top 20% of the November 2010 Green 500 List. The authors report the same metric for the NVIDIA C2050 nodes, which obtain an energy efficiency of .966 GFLOPS/W, which would place them in the top 99% of the same list. The authors do not make the comparison here since it is not the focus of their research, we can extrapolate these results to make some comparisons between the Cell and GPU. This is not the PS3's best possible performance since the HPL implementation uses double precision (DP) floating point computations. If we assume the same power consumption (199W average) and the improved DP performance of the PowerXCell 8i of 108.8GFLOPS, we can roughly calculate a best case performance for the Cell architecture of 0.547 GFLOPS/W, which is still just better than half the efficiency of the GPU. There are some flaws in this calculation, such as the fact that the PS3 used in [16] used a Cell chip manufactured using a 90nm technology, versus the 40nm [6] process used for the GPU, but it gives direction for future comparisons.

Cell based systems remained near the top of the Green500 list from 2008 to 2010. The June 2010 Green500 list [2] reports a PowerXCell 8i cluster in position #1, achieving 773 MFLOP/s, the last time a Cell based system topped the list. From November 2010 onwards, the Cell systems were surpassed by both homogeneous multicore and GPU based systems.

# 4. METHODOLOGY

In order to predict how the Cell architecture may have evolved, and level the playing field for comparison with modern architectures, we construct models for performance, power, and area of the Cell.

## 4.1 Performance Modeling

When the Cell was still in production, IBM provided a full system simulator for the architecture. They have since removed all download links from their website, and no longer offer any kind of support for the tool. We were able to find a copy of the simulator from another source, and eventually run the simulator on a virtual machine running Fedora 23. However, we found that the simulator would only be capable of running Fedora 7, or no operating system, which would make it difficult for us to compile and execute modern benchmarks on the simulator. Furthermore, we were unable to find a copy of the Cell SDK that would be required to compile programs to execute on the simulator, as IBM no longer makes this available either, so we would have needed to find benchmark binaries already compiled for the Cell to run. Due to all of these factors, we decided the Cell system simulator would not be feasible for use in our study.

To simulate and model our Cell processor we used the Multi2sim simulator [18]. It simulates various instruction set architectures like x86, ARM, AMD Evergreen, AMD Southern Islands and Nvidia's CUDA. It provides statistics on performance based on the execution of a program. For our study, we have focused on the detailed simulation feature provided with the multi2sim. It is a timing or architectural simulator, that is, the software component models hardware structures and keeps track of their access times. The modeled hardware includes pipeline stages, pipe registers, instruction queues, functional units, cache memories and others. The program is broken up into instructions and a pipeline model is created where ISA like execution is mimicked by the simulator.

In our study we focused on the application only emulation rather than the full system emulation. This choice is made since we work on understanding the working of cell processors and also compare it to the Graphics processors. Since application level changes needed to be implemented in order for the program to take benefit of the architecture. With IBM introducing APIs for programmers where they can leverage branch prediction and memory management in their programs.

Although Multi2sim does not support IBM's Power Instruction set Architecture, we have made an informed choice to see the comparison between cell architecture and GPU architectures in terms of performance, by focusing on the architectural aspect that distinguishes the Cell architecture most from traditional CPUs and GPUs: the local storage. So for our study, we use the x86 Instruction Set Architecture (ISA) for the larger PPE-like core and AMD's Southern Islands ISA for the SPE like cores, as well as the GPU cores. This enables us to perform a fair comparison evaluating only the impact of using local storage over a cache. Once we have the common base configuration set, we can then focus on creating a similar architecture for cell processor and GPUs.

## 4.2 Power/Area Modeling

For power and area simulation, we use McPAT(Multicore Power, Area, and Timing) [15]. McPAT allows for configuration of many processor components and parameters, from general parameters like clock frequency and transistor size to component level details like cache line size and associativity, integer and floating point pipeline lengths, etc. For the SPE cores, we build off the sample configuration McPAT provides for the Niagara architecture. We felt this was a good starting point as this architecture is similar to the Cell in the sense that it seeks to achieve low power per core, and prefers a large number of simple in-order cores over complex superscalars. We substitute all of the published Cell parameters that we can find into this configuration, leaving the rest unmodified. A similar process is used for the PPE core, borrowing some of the baseline parameters from the Intel Xeon configuration provided with McPAT. This is similar to the methodology used in [17] to simulate a GPU using McPAT.

Our model does have some inherent flaws, however, since we are limited to the capabilities of McPAT. Although the Cell architecture features 7 specialized execution units for integer and floating point operations per SPE [11], McPAT does not allow for this level of detail in its component modeling, so we simply tune these parameters such that they occupy a similar fraction of area. An SPE contains two pipelines, which can be modeled in McPAT, however this does not completely reflect the SPE architecture, as McPAT only allows separate integer and FP pipelines to be configured, while the SPE has 1 mixed integer and FP pipeline [11]. A slight consolation is made modeling the register file, where McPAT requires explicit modeling of integer versus FP registers, while the SPE has general purpose registers that can be used for either [11]. Although the SPE contains a simple static branch predictor, we do not model this because McPAT only includes a tournament style branch predictor. We find this an acceptable omission considering the branch predictor accounts for only 3% of the SPE area. Unfortunately, McPAT does not provide for the possibility of a local memory instead of a cache. McPAT models the L1 cache as part of the instruction fetch unit, and the data cache as part of the load and store unit [15]. Therefore, we compensate for the necessity of using a cache by adjusting the L1 parameters until the area of our modeled chip is comparable to its corresponding component in the Cell architecture. We label our model architecture "Cell$" to indicate the addition of a cache. For area modeling, we believe this is a reasonable consolation to make since we are trading one memory structure for another. For power modeling, we will need to compensate for this differently, as the cache will surely consume more power than a local memory store would.

# 5. EXPERIMENTAL RESULTS

## 5.1 Area Model Validation

The dimensions of the initial 90nm SOI technology SPE were 2.54mm by 5.81mm (14.76 $mm^2$), per [11]. We use this as a target for our McPAT model. We use the image of a SPE provided in [11] and shown in figure 1 to calculate the percentage of the die occupied by each unit, based on the number of pixels it occupies, and compare this to our McPAT output. Overall, the Cell occupies an area of 221 $mm^2$ [4] , while our model occupies 291 $mm^2$. McPAT provides output in terms of functional blocks within the core - categorizing all subcomponents as part of the instruction

| # SPEs | Technology Scale (nm) | Peak Power (W) |
|---|---|---|
| 8 | 90 | 146 |
| 8 | 45 | 51 |
| 8 | 28 | 19 |
| 16 | 45 | 89 |
| 16 | 28 | 33 |
| 32 | 45 | 165 |
| 32 | 28 | 60 |

**Table 1: Cell model power by technology**

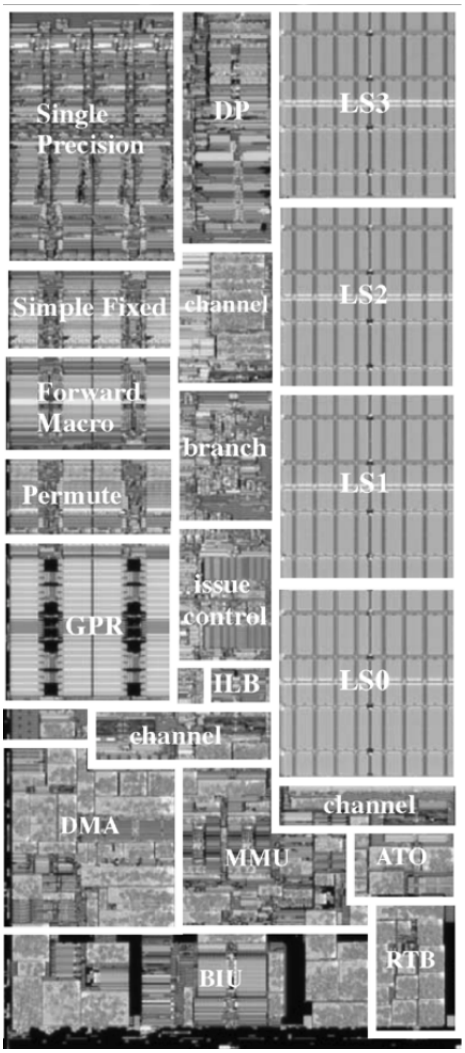


**Figure 1: A photo of the SPE die**

fetch unit (IFU), load-store unit (LSU), memory management unit (MMU), or execution unit (XU). For the total area of the core, it reports a number larger than the sum of these 4 units, so we categorize this area into "Others". [15] details the subcomponents contained in many of these units. However, there are still some parts of figure 1 that we cannot determine a one to one mapping for. We intuitively categorize the local storage as part of the LSU, since this is where McPAT puts the L1 data cache, and we put the remainder into the "Others" category as this makes sense from the perspective of chip area allocation.

Next, we compare our 45nm simulation to the PowerXCell 8i. The PowerXCell 8i occupies roughly half the area of the 90nm Cell BE, at 109 $mm^2$, while our model occupies about 100 $mm^2$. This is surprising since our 90nm model was larger than it's actual implementation, and an issue that we consider further in our power model discussion below.

## 5.2 Power Model Validation

To validate our power model, we compare our model SPE's power consumption to the voltage/frequency shmoo published for an SPE in [11], both at 90nm. The published results we compare to are integer values only, which makes it difficult to judge the trend. We see that in the worst case, our model consumes 3x the power of the actual SPE, and 2x at a higher voltage ($V_dd = 1.3$). We can attribute this to several factors. First, our model uses a cache, which we know to draw more power than the local storage used in an SPE. Second, we know that STI designed several custom latches and latches [17] to improve power efficiency and latency on the Cell, whereas McPAT has been validated against more commonly used hardware components. The results published in [11] also note that they are reporting "active power", which they define as the amount of power consumed by switching logic and array circuits. While this sounds the same as the peak power numbers we gather from McPAT, there may be some difference in the methodology of power reporting. Unfortunately it is quite difficult to find published TDP values for the entire Cell BE chip at 90nm. IBM has removed all Cell specifications from their site, however unofficial sources report TDP as high as 110W for the chip alone, and a peak of 199W has been observed for the PlayStation 3 running a 90nm Cell BE core, and powering other components such as a hard drive, etc, giving a high upper bound. Our model gives a peak power of 146W for the chip overall. Even with some uncertainty in the published SPE results, and inconsistent reports of overall TDP, it seems probable that our model is consuming more power than the actual Cell BE would.

To further evaluate the accuracy of our model, we compare a 45nm scale version to the PowerXCell8i, whose only

significant architectural change other than SOI technology was the floating point pipeline. At 45nm, our model gives a peak power of 50W, which is significantly lower than the 92W TDP reported for the PowerXCell 8i in [7]. This leads us to have some doubts in the validity of our model. Table 5 shows the peak power output from McPAT for different numbers of SPE cores in our model, and at different technology sizes. The power consumption appears to decrease at a rate faster than linear with technology scale. It is possible that the 45nm PowerXCell did not fully take advantage of technology improvements that McPAT is modeling at this scale, but this still leads us to be skeptical that the McPAT power levels would actually be attainable. Taken at face value, these results do give some hope for the theoretical scalability of the Cell architecture, as far as its ability to remain power efficient.

## 5.3 Energy Efficiency vs. Modern GPU

We compare our Cell models to a modern GPU architecture. The AMD Radeon 7970 is an ideal candidate, as an example configuration is provided with Multi2Sim that has been validated against this architecture, and AMD has published results on its theoretical peak performance and power consumption. Released in 2012, the Radeon 7970 was built using a 28nm process, has a peak theoretical DP performance of over 1 TFLOPS, and a peak power draw of 250W [1]. Therefore, it has a peak theoretical performance/power ratio of 4 GFLOPS/W. With a peak DP performance of 108.8 GFLOPS in the PowerXCell 8i, assuming we could scale this to 32 SPEs without encountering communication or other bottlenecks, we would expect a peak theoretical DP performance of 435.2 GFLOPS on a 32 SPE Cell processor. For our 32 SPE Cell model, we quadruple all shared resources as well. We obtain a peak power consumption of 60 W, which using our theoretical peak performance gives a peak theoretical power efficiency of 7.25 GFLOPS/W. This result gives some hope that it may be possible for the Cell architecture to compete with GPGPUs, however, we have some skepticism due to the concerns about the validity of our power model discussed in the previous sections. In addition, our performance models discussed subsequently indicate that the Cell's memory system is a limiting factor on its achievable performance.

## 5.4 Performance analysis

To better understand how the architecture works and its eventual demise, we tried to replicate the processor in x86 based ISA. We did this in part due to the difficulty of finding a simulator for the Power ISA, and to rule out ISA differences among Cell processors and Graphics processing units as a factor in performance. To understand our analysis, we must think of the architecture of Cell processors and GPUs in the year 2009.

IBMâĂŹs next planned line of cluster computing grade Cell Processors would have had 32 SPEs per chip. Comparing this to a GPU, we see a minimum of 32 Streaming processing threads on chip. This allows for a configuration where we can have one central processor to manage and give work to the other processing elements. Here we can single out the major portion of the processor. So using Multi2sim we create an architecture based on x86 ISA. With a base design in mind, we can proceed towards creating a similar configuration on Multi2sim.

Since, SPEs did not have caches in them, we have not included caches in their design. They however had a SRAM access which was slower than a cache but was available on chip. To mimic that we simulated the cell processor with adequate latency. All the SPEs can share a common SRAM-based on chip memory coupled with an interconnect network that would allow for high bandwidths. We use AMDâĂŹs graphics processor elements to mimic a Cell architecture. So all of the SPEs run a specialized Southern Islands ISA. The GPU model on the other hand has a Level one cache shared by a group of 8 of the streaming processors, and a Level 2 cache that is shared by all of the streaming processors. This is the difference between the two architectures on a level where everything else is normalized for our study.

To model the PPE, a general purpose core responsible for executing scalar applicationsm, we included a standard x86 based architecture core. The use of a PPE core for our simulations, where no operating system is involved, is to provide the data computation for SPEs, which can execute parallel sections of the application much more efficiently.

Lastly, we picked benchmark applications that were coded to take use of the SPE cores or the streaming cores (GPU). This allows the central processor to assign parallel jobs to the specialized units, and itself be responsible for executing scalar and managing the processes on the operating system.

Ideally we would have liked to use the industry standard unit of Floating point operations per second (FLOPS), however we were not able to get this output from Multi2Sim for our selected architectures. Since, our analysis is not just towards high performance or supercomputing, we will take a general view of performance to see where the Cell processor can be used and how it is out of the market now. For this, we use Operations (Instructions) per second (OPS) and Operations per second per Watt (OPS per Watt) as our units of measurements. All results are normalized such that the Cell model has a value of 1 for each measurement.

From figures 3 and 4 we see how much performance difference can be seen between Cell processors and GPUs. Tables 5.3 and 5.3 present the full details of our simulation output. We ran through multiple parallelized computational applications to see how their underlying architectures govern their performance.

First, we start our analysis to measure the peak performance capabilities (figure 3), for this reason we do not focus on power consumption. Here we see for some programs the Cell processors performed almost as well as the GPU. There is only a marginal gain in performance in benchmark applications such as BinomialOption, FFT and FloydWarshall. This can be due to the low inherence cache requirement in these applications. So the performance of both the parallel applications is similar to each other in the different architecture.

In other applications we can see a fair amount of performance gain when using GPUs. Cell processors we have taken for our study includes a 32 SPE configuration without a cache memory. So applications which are computationally heavy will tend to store more data in cache and get better performance out of it. Here we see Binary Search which is a fairly scalar method of computation doing better on GPUs but applications which are highly parallel like MatrixTranspose, FastWalshTransform, Reduction and SimpleConvolution use the benefits of caches and interconnects to perform significantly better than the cell. The GPU uses its Stream-

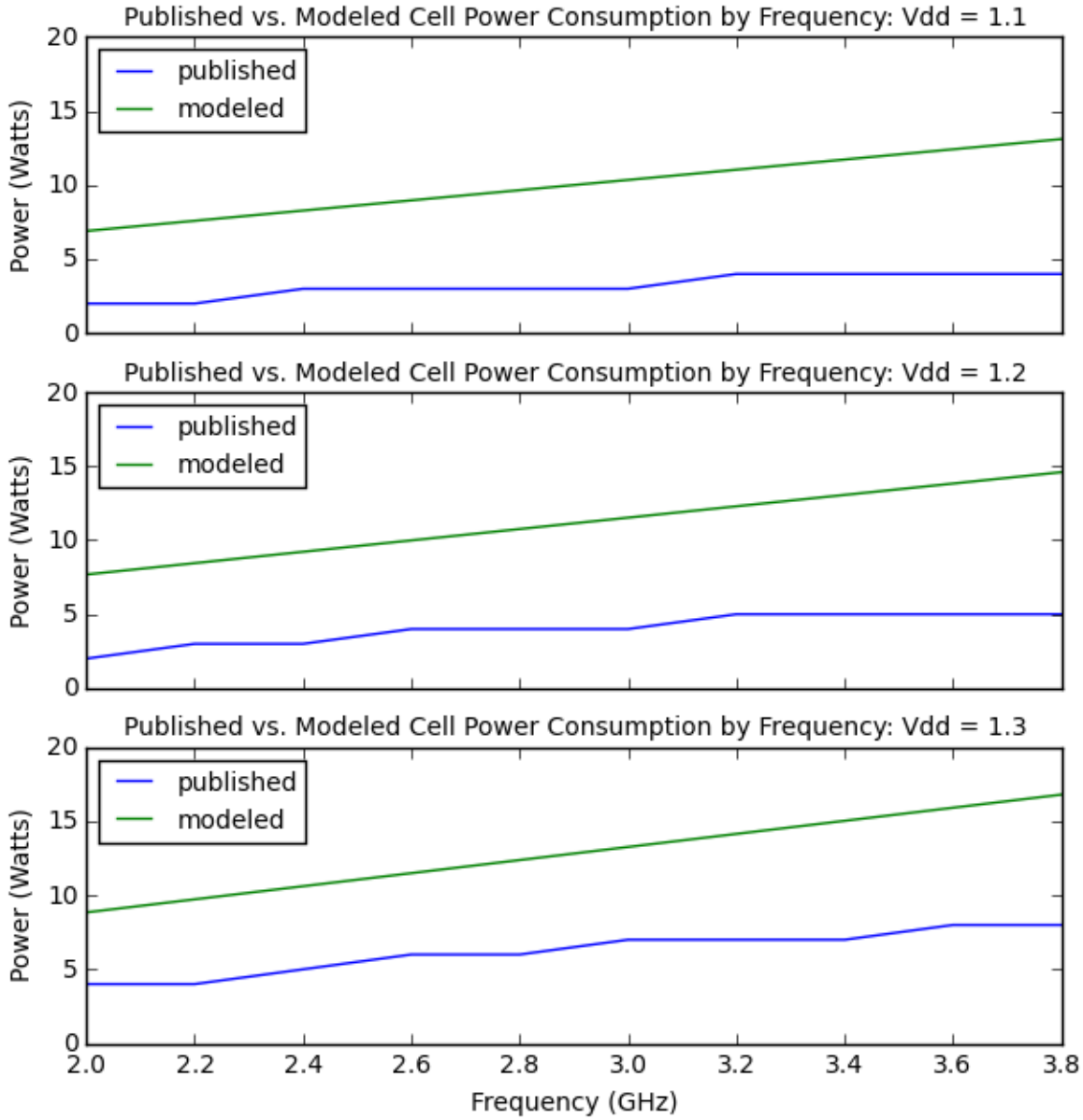


**Figure 2: Model vs. published power consumption for an SPE**

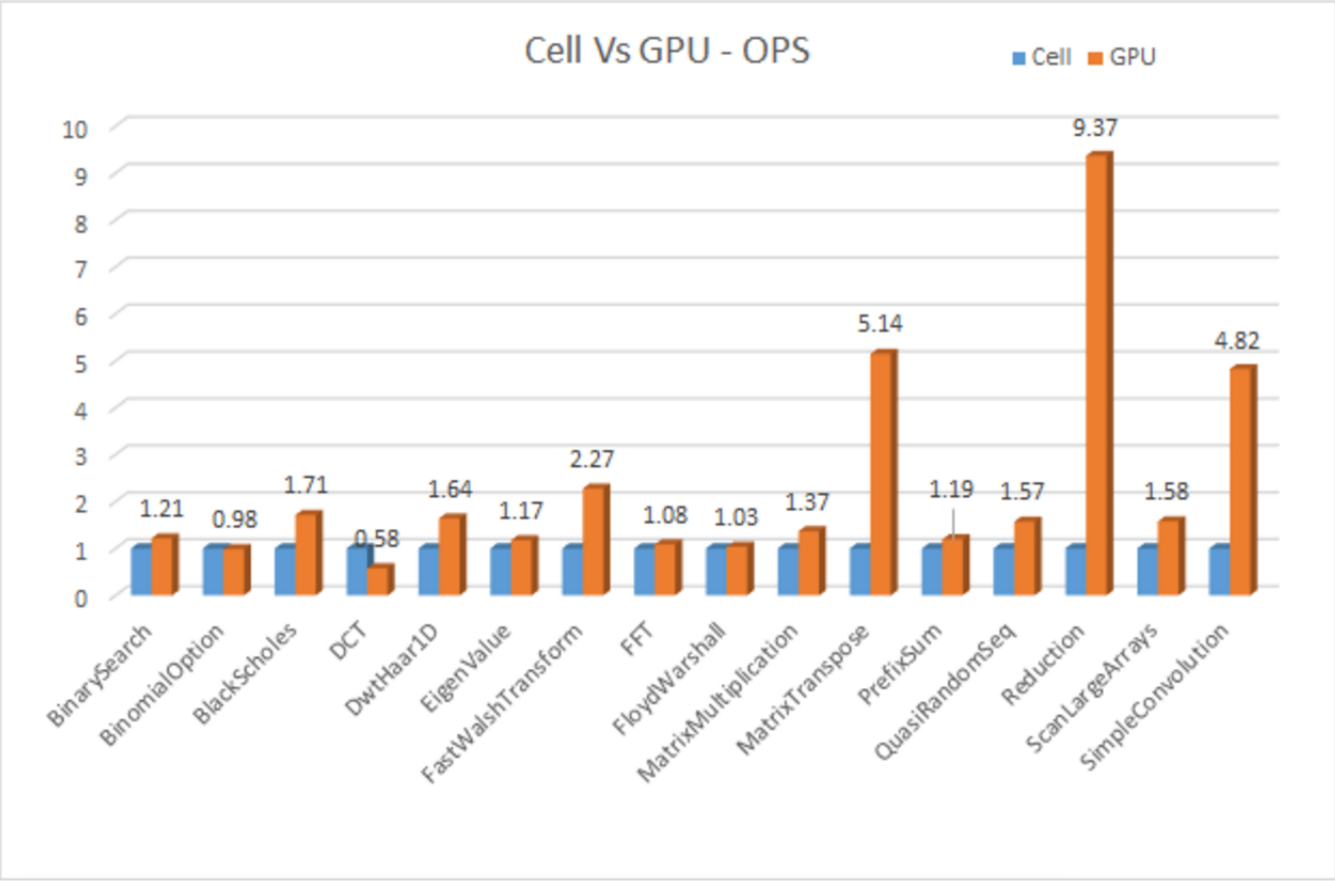


**Figure 3: Modeled Cell vs. GPU performance - normalized to 1 for Cell**

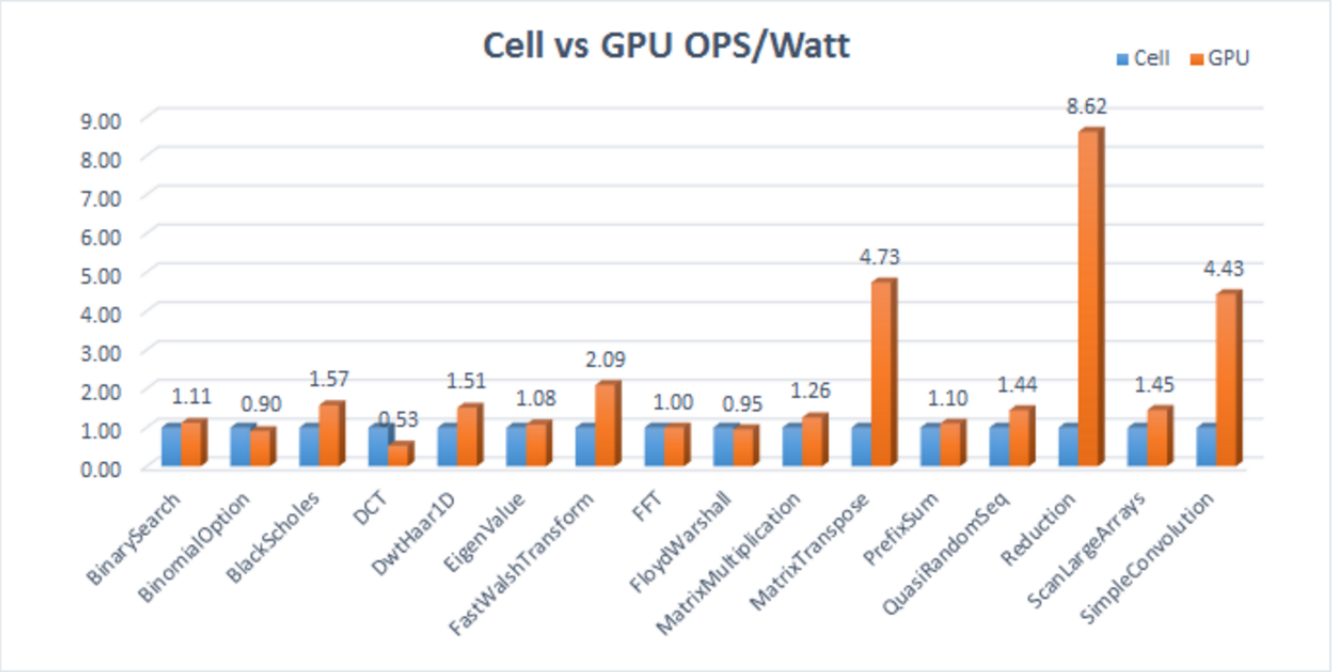


**Figure 4: Modeled Cell vs. GPU performance / power - normalized to 1 for Cell**

| Category (McPAT) | Subcomponent | Cell (%) | Cell ($mm^2$) | Cell$ (%) | Cell$ ($mm^2$) |
|---|---|---|---|---|---|
| **Instruction Fetch Unit** | | 7.4% | 1.09 | 10.0% | 2.17 |
| | Issue Control | 4.2% | 0.61 | | |
| | Branch Predictor | 3.2% | 0.47 | | |
| **Load Store Unit** | | 36.1% | 5.32 | 30.3% | 6.58 |
| | Local Storage | 29.2% | 4.31 | | |
| | DMA | 6.8% | 1.01 | | |
| **Memory Management Unit** | | 6.8% | 1.01 | 6.4% | 1.39 |
| | MMU | 6.8% | 1.01 | | |
| **Execution Unit** | | 29.4% | 4.33 | 27.6% | 6.01 |
| | Floating Point | 13.8% | 2.03 | | |
| | Integer | 9.7% | 1.44 | | |
| | Register File | 5.8% | 0.86 | | |
| **Others** | | 20.4% | 3.01 | 25.8% | 5.61 |
| | Channels, etc. | 6.5% | 0.96 | | |
| | Bus Interface Unit | 9.4% | 1.39 | | |
| | Relational Translation Buffer | 4.5% | 0.66 | | |
| **Total** | 100% | 14.75 | 100% | 21.75 | |

**Table 2: Cell model vs. actual area at 90nm**

| Benchmark | Cell | | | | | Instr/Sec | Instr/W |
|---|---|---|---|---|---|---|---|
| | | x86 (PPE) | | SPE | | | |
| | Total Time | Time | Instr. | Time | Instr. | | |
| BinarySearch | 0.47 | 0.37 | 1020506 | 0.09 | 128 | 2,171,562 | 9,442 |
| BinomialOption | 709.42 | 0.38 | 796163 | 709.22 | 1984512 | 3,920 | 17 |
| BlackScholes | 479.87 | 6.93 | 22983346 | 473.71 | 1139712 | 50,270 | 219 |
| DCT | 11.99 | 3.01 | 6450648 | 9.01 | 20672 | 539,726 | 2,347 |
| DwtHaar1D | 6.38 | 5.51 | 12132204 | 0.84 | 2113 | 1,901,931 | 8,269 |
| EigenValue | 154.8 | 2.93 | 8761627 | 151.87 | 1617502 | 67,049 | 292 |
| FastWalshTransform | 8.5 | 3.23 | 7379484 | 5.29 | 4000 | 868,645 | 3,777 |
| FFT | 1.42 | 1.41 | 1530000 | 0.58 | 1316 | 1,078,392 | 4,689 |
| FloydWarshall | 551.7 | 3.78 | 7682561 | 548.31 | 188820 | 14,268 | 62 |
| MatrixMultiplication | 27.54 | 16.11 | 46121929 | 11.39 | 8000 | 1,675,016 | 7,283 |
| MatrixTranspose | 10.02 | 0.78 | 1636251 | 9.24 | 3392 | 163,637 | 711 |
| PrefixSum | 1.66 | 1.18 | 3973824 | 0.46 | 1341 | 2,394,678 | 10,412 |
| QuasiRandomSeq | 18.91 | 13.87 | 27073094 | 5.04 | 32288 | 1,433,389 | 6,232 |
| Reduction | 0.75 | 0.53 | 1352960 | 0.25 | 714 | 1,804,899 | 7,847 |
| ScanLargeArrays | 5.71 | 3.97 | 8891798 | 1.74 | 3688 | 1,557,878 | 6,773 |
| SimpleConvolution | 12.3 | 0.53 | 1060946 | 11.79 | 17900 | 87,711 | 381 |

**Table 3: Cell model performance**

ing processors along with its level 1 and level 2 caches to outperform in such applications. These applications have a tremendous amount of parallel segments in them, this can be utilized by the GPU architecture along with the programming APIs that help it to do so. The results here are conclusive enough to show how much of performance increase can be achieved by GPUs while the area and power constraints are kept same.

Lastly, we shift our focus on the performance to power aspect of our analysis. Here we try to keep both of the architectures at a similar scale and simulated them while keeping power into consideration. Traditionally, when IBMâĂŹs cell architecture was running supreme in supercomputers, they were featured as one of the most energy efficient processors made. So as a unit of measurement we use Operation per second per Watt.

From figure 4, we can see if by including the power into our study, the cell architecture could perform close to the GPUs. Since we had low confidence in the scalability of our power model, and we were unable to directly model the local storage, the primary variable in our performance model, we once again use the published TDP for the AMD Radeon 7970. Our GPU is modeled after this architecture exactly, and our Cell model deviates from only in the sense of the memory hierarchy. For the GPU, we use the published value of 250W directly in the denominator for our OPS/Watt computations. For the Cell, we deduct 20W. To arrive at this amount, we looked at results of a slightly older GPU modeled in **??**, where the authors saw a maximum power consumption of near 20W for the L1 and L2 caches combined. We take this to be an upper bound, since we know the local storage will consume a nonzero amount of power.

We can see for some applications Cell processors perform marginally better than the GPUs. In our analysis, cell processors takes less power due to the lack of caches and simple branch predictors. This is consistent in the simulation re-

| Benchmark | GPU | | | | | Instr/Sec | Instr/W |
|---|---|---|---|---|---|---|---|
| | | x86 (PPE) | | SPE | | | |
| | Total Time | Time | Instr. | Time | Instr. | | |
| BinarySearch | 0.47 | 0.37 | 1020506 | 0.09 | 128 | 2,171,562 | 9,442 |
| BinomialOption | 709.42 | 0.38 | 796163 | 709.22 | 1984512 | 3,920 | 17 |
| BlackScholes | 479.87 | 6.93 | 22983346 | 473.71 | 1139712 | 50,270 | 219 |
| DCT | 11.99 | 3.01 | 6450648 | 9.01 | 20672 | 539,726 | 2,347 |
| DwtHaar1D | 6.38 | 5.51 | 12132204 | 0.84 | 2113 | 1,901,931 | 8,269 |
| EigenValue | 154.8 | 2.93 | 8761627 | 151.87 | 1617502 | 67,049 | 292 |
| FastWalshTransform | 8.5 | 3.23 | 7379484 | 5.29 | 4000 | 868,645 | 3,777 |
| FFT | 1.42 | 1.41 | 1530000 | 0.58 | 1316 | 1,078,392 | 4,689 |
| FloydWarshall | 551.7 | 3.78 | 7682561 | 548.31 | 188820 | 14,268 | 62 |
| MatrixMultiplication | 27.54 | 16.11 | 46121929 | 11.39 | 8000 | 1,675,016 | 7,283 |
| MatrixTranspose | 10.02 | 0.78 | 1636251 | 9.24 | 3392 | 163,637 | 711 |
| PrefixSum | 1.66 | 1.18 | 3973824 | 0.46 | 1341 | 2,394,678 | 10,412 |
| QuasiRandomSeq | 18.91 | 13.87 | 27073094 | 5.04 | 32288 | 1,433,389 | 6,232 |
| Reduction | 0.75 | 0.53 | 1352960 | 0.25 | 714 | 1,804,899 | 7,847 |
| ScanLargeArrays | 5.71 | 3.97 | 8891798 | 1.74 | 3688 | 1,557,878 | 6,773 |
| SimpleConvolution | 12.3 | 0.53 | 1060946 | 11.79 | 17900 | 87,711 | 381 |

**Table 4: GPU model performance**

sults we get. Even though this is true, GPU still performs better in most of the applications. Cell processor do well in applications which do not require it to use a cache. Since IBM provided API to programmers to manage memory and predict branches in code itself, it also provided compiler modifications to suit the parallel needs. Still with the introduction of thread level parallelism introduced by the GPU architecture, the progress in Cell architecture was nullified. We see that GPUs perform better even when we take power into consideration. Since only a small amount of power is taken extra by GPUs when compared with Cell processors, but they provide an exponential speedup.

# 6. CONCLUSION

In this paper we have studied how a Cell processor architecture was well suited to high performance computing at the time of its introduction. Although, it was developed as a CPU in a game console, its use was realized in supercomputing and other applications. From our study we can draw several conclusions as to why the Cell processor line was discontinued.

## 6.1 Factors in the downfall of Cell

We summarize several factors that contributed to the discontinuation of the Cell architecture.

### *6.1.1 Main Memory access rate*

Execution units can generate floating point results at a speed that is much higher than the speed at which the memory can feed data to the execution units. The main memory of the Cell processor is capable of a 25.6 GB/s peak transfer rate, which in turn limits each SPE to a peak performance of 25.6 Gflop/s in single precision, assuming instructions on a single operand. This is not just for the cell processors alone, but the memory developments have not kept up with the developments in the processor technology.

### *6.1.2 Double precision performance*

Peak performance of double precision floating point arithmetic was a factor of 14 below the peak performance of single precision in the original Cell BE. Computations that demand full precision accuracy will see a peak performance of only 14.4 Gflop/s. Although this was later improved in the PowerXCell 8i to achieve about half the performance of SP operations, by this time other architectures such as GPGPUs were offering better performance for both SP and DP.

### *6.1.3 Programming difficulties*

Writing efficient and fast code for the Cell processor is, in fact, a difficult task since it requires a deep knowledge of the processor architecture, of the development environment, and some experience. High performance can only be achieved if very low level code is produced that is on the border line between high level languages and assembly. Besides requiring a lot of effort, source code developed for the Cell processor is not portable at all on other architectures. IBMâĂŹs architecture relied on the POWER Instruction set architecture, which was not as prevalent as x86 architecture. This made the software not portable and a cross compiling together with code manipulations was required to be implemented on Cell architecture. So to get the desired performance, more time and money was required to be spend in developing code catered to Cell architecture. When GPUs came into computing market, they released APIs and programming constructs to make use of the parallel architecture. They unlike IBM, introduced programmer friendly construct and provided good support to developers. These are extremely important factors in a time where human resource (developer) costs generally exceed hardware costs for typical applications. The risks associated with having application code that is not portable also take away from the appeal of the Cell.

### *6.1.4 Lack of Caches*

The Cell processorâĂŹs SPEs did not have caches in them. And it would require the programmer to predict the branching and cache misses and program it. Although this allowed the programmer more flexibility and performance potential,

it increases programming difficulty and can lead to worse performance if not utilized correctly.

#### 6.1.5 The rise of GPU

Since in the late 2009s, the GPU technology was seen as a coprocessor. It had more number of threads to provide for the same size of processor. This allowed for the parallelism to increase 100 folds. Since a cell processor could only provide 10 to 100 SPEs, a GPU could provide more than a 1000 threads for parallel execution. Even more, the GPUs had a unified language which made it easier to program it than a Cell processor.

#### 6.1.6 Lack of Bandwidth

From our study, we saw how data hungry processors are when the number of parallel processors are increased. To address this, GPUs have Graphics RAM on the same motherboard with clever interconnects to increase bandwidths. Cell processor could not keep up with the data demand.

#### 6.1.7 Fewer hardware contexts

At the height of Cell architecture development, a maximum of 32 SPE based core was proposed (which we used in our analysis). Comparing this with the GPUs of its time was not enough. Since most GPUs could provide more than 1000 threads to be executed according to SIMD parallelism.

### 6.2 Concluding Remarks

In this study, we have presented several reasons for the downfall of the Cell architecture. We have presented models for both power and performance simulation, and although we believe there are some flaws in our methodology, they support our theory, and with further refinement could be better used to simulate the Cell. Through published studies measuring the performance of the Cell and GPUs independently, and the Green500 list, we have seen that GPGPUs can achieve greater energy efficiency than the Cell. Through standardization of programming models and libraries, we have seen even greater adoption of GPGPUs for massively parallel HPC applications.

Although IBM's Cell processor line is no more, this was a pioneering architecture for HPC. One can speculate that this was a motivating factor in GPU manufacturers' adoption of general purpose programming models, and as the first commercially successful heterogeneous architecture, it certainly paved the way for this category of processors. Although elements such as the local storage have been removed, and the characteristics of the "synergistic" processors have changed, today's single chip CPU + GPU devices carry on key architectural principles from the Cell architecture of having different cores for different jobs. They strategy of having one type of core to run an operating system and coordinate tasks, and using many smaller, simplified cores to accelerate computations is still very relevant and lives on in these devices.

## 7. REFERENCES


[1] Amd radeon hd 7970 ghz edition review: Battling for the performance crown.

[2] The green500 list - june 2010.

[3] The green500 list news and submitted items.

[4] Isscc 2005: The cell microprocessor.

[5] The linpack benchmark.

[6] Techpowerup.

[7] With its new powerxcell 8i product line, ibm intends to take accelerated processing into the hpc mainstream.

[8] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho. Entering the petaflop era: the architecture and performance of roadrunner. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 1. IEEE Press, 2008.

[9] A. Buttari, J. Dongarra, and J. Kurzak. Limitations of the playstation 3 for high performance cluster computing. 2007.

[10] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation - a performance view. *IBM Journal of Research and Development*, 51(5):559–572, 2007.

[11] B. Flachs, S. Asano, S. H. Dhong, H. P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, et al. The microarchitecture of the synergistic processor for a cell processor. *IEEE Journal of Solid-State Circuits*, 41(1):63–70, 2006.

[12] H. P. Hofstee. Power efficient processor architecture and the cell processor. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 258–262. IEEE, 2005.

[13] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, et al. Introduction to the cell multiprocessor. *IBM journal of Research and Development*, 49(4/5):589, 2005.

[14] J. Kurzak, A. Buttari, P. Luszczek, and J. Dongarra. The playstation 3 for high-performance scientific computing. *Computing in Science and Engineering*, 10(3):84–87, 2008.

[15] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM, 2009.

[16] R. Luley, C. Usmail, and M. Barnell. Energy efficiency evaluation and benchmarking of afrl's condor high performance computer. Technical report, DTIC Document, 2011.

[17] D. Pham, S. Asano, M. Bolliger, M. Day, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, et al. The design and implementation of a first-generation cell processor-a multi-core soc. In *Integrated Circuit Design and Technology, 2005. ICICDT 2005. 2005 International Conference on*, pages 49–52. IEEE, 2005.

[18] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. Multi2Sim: A Simulation Framework for CPU-GPU Computing . In *Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2012.

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

## APPENDIX

The configurations and simulation output used for power modeling are available at https://github.com/tonygallotta/ece568-final-project