# An Introduction to the Cell Broadband Engine Architecture

Khaled Ali Zulfikar
Electrical and Electronic Engineering
Curtin University
Perth, Australia
khaled.alizulfikar@student.curtin.edu.au

***Abstract*—The Cell Broadband Engine Architecture represents a shift in conventional processor design, offering a revolutionary approach that maximises performance. This makes it suitable for computationally intensive applications. Its design focuses on simplicity, as well as efficiency on a performance per transistor basis. Furthermore, its simplified programming model showcases its potential for development and optimisations.**

**The processors significance is not limited to its time, it has continued to inspire and drive innovations in the field of microprocessor design. It represented a path for future developments, making in a compelling option for various high-performance computing needs.**

***Keywords—Microprocessors, Computer architecture, Design optimization, Process Design***

## I. Introduction

In 2006, Sony released the PlayStation 3 (PS3), which included a Cell Broadband Engine Processor (Cell/B.E.) [1]. The processor was designed to deliver performance beyond conventional processor architecture. In 2002, Sony and Toshiba, which who had previously collaborated on the PlayStation 2, partnered with IBM to create the Cell Architecture, with a goal to create a new general-purpose processor which expanded beyond its use in the PlayStation 3. Sony and Toshiba used the processor to manufacture next generation Blu-ray, HDTV and HD camcorders which all demanded high processing power [12]. IBM also intended to use the Cell architecture in servers, as well as the Roadrunner supercomputer, a $133 million dollar project that was the first to break the petaflop barrier, able to calculate 1 quadrillion calculations per second [13]. Each of the three companies aimed to satisfy their diverse needs which demanded power efficiency, reliability, predictability and compatibility across different generations. The collaborative efforts of the three companies brought approximately 400 engineers from 10 design centres from around the world [9].

The Architecture of the Cell Processor differs significantly from conventional processor design, and has a similarity in design to Single instruction, multiple data (SIMD) based supercomputers, that feature multiple processors. The Cell processor was designed as a general-purpose processor; however, its design makes it optimised for computer intensive tasks, allowing it to handle specific types of code more efficiently. The Cell processor can perform magnitudes better existing desktop CPUs in terms of maximum processing power, where an individual Cell processor is capable of a computing capacity of up to 256 GFLOPS per second when running at 4GHz [1]. The Cell combined a 64-bit power architecture (Performance Optimisation with enhanced RISC Performance Computing, or PowerPC) and several synergistic co-processors which operated on a shared, coherent memory [10]. The Cell avoids slow components overall, ensuring high performance, and are connected by high bandwidth interconnects. The Cell architecture is also characterised by its high potential for performance distributed computing, flexibility and scalability, delivering high performance in a wide range of applications. It brought a new approach to high-performance computing, whilst maintaining affordability.

## II. Evolution of the CPU Architecture

Single chip CPU designs had evolved over several decades, progressing from 4-bit designs in Intel's 4004, to the first 64-bit processor in 1992 [14]. Cache memory was also introduced by Motorola's 68010, bridging the gap between CPU operation and memory speeds, starting with small caches in early processors and gradually increasing in size over time. Pipelining, introduced with Motorola's 68040, allowed the CPU to divide it's instruction execution path, or critical path, into multiple stages [14]. Superscalar execution enabled processors to execute multiple instructions simultaneously, a feature that arrived with processors such as the Intel Pentium 80586. Out of Order Execution (OOO), speculative execution and instruction pre-fetching were all introduced with the Pentium pro processor [14]. Capabilities such as Single Instruction, Multiple Data (SIMD), which added vector processing capabilities were included with PowerPC processors. Intel introduced hyperthreading with the Pentium 4, which allowed better utilization of the CPU's resources [14].

## III. Diminishing Returns with modern CPU Improvements

It has been observed that for every doubling in the number of transistors in microprocessor design, it will only correlate to approximately a 40% increase in performance [5]. When designing a CPU, the rationale behind the architectural design decision lies in three main governing constraints, the power wall, the frequency wall and memory wall. Every end device is power limited; hence power efficiency must be taken into account. A microprocessors power efficiency can be referred to its energy performance per transistor or operation. The memory wall refers to the increased memory access time, which causes limitations on the systems performance. The main memory speed is usually magnitudes slower than that of the CPU. All CPU's will inherently have a maximum design frequency calculated by system constrains such as power supply variation, reference clock jitter and thermal conditions [7]. The CPU will have a processor capability based on the number of instructions that are executed per clock cycle as well as the design frequency which is determined by these three constraints.

Developments in microprocessor architecture leading up to the Cell/B.E. had shown that there is a trade-off between performance improvements and power consumption [5]. As CPU's add new features and technologies, inherently, performance had increased, but each enhancement had diminishing returns; its power consumption had tended to rise disproportionately to the performance gains. In the case of the Cell Processor, the design choices favoured a simpler, power efficient, higher clock frequency design over the inclusion of the complex features found in many modern processors. In this section, such features will be discussed in more detail and provide reasoning for their diminishing returns on a performance per transistor basis.

### A. *Modern Microprocessor Caches*

Modern general-purpose processors dedicate a significant portion of their transistors to caches and other memory related structures. A 'cache miss' is when data is not found in the cache and is inversely proportional to the size of the cache, improving the processors performance. Gelsinger's law states that cache size needs to continue to increase to keep up with the demand for computing performance [5]. However, despite architectural efforts to reduce the memory wall, data access from the main memory is still a crucial limiting factor. For an example, when a cache miss occurs, the microprocessor may have to wait for several thousand clock cycles in some cases. The limited bandwidth of the main memory proves that the efficient management an organisation of data in memory is crucial for software performance. It Is suggested that more efforts should be given to managing memory locality, optimising how data is stored in the memory to reduce cache misses rather than trying to optimise the cache [1].

### B. *Superscalar Architectures*

The implementation of parallelism by trying to increase the number of executed instructions per cycle is not always the most efficient method in terms on a performance per transistor metric. By increasing this parallelism, the number of transistors dedicated to the microprocessors cache is expanded, as a secondary load-store port is required, which causes the cache to double in size [5]. Introducing a secondary port will also require additional logic to maintain the order of the program instructions that involves loads and stores, which further decreases microprocessor efficiency.

### C. *Out-of-Order Instructions (OOO)*

Most modern microprocessors effort to improve instruction-level parallelism by allowing instructions to be executed in a different order from their appearance in the program, leading to better performance [5]. This process, called Out-of-Order (OOO) processing, however, introduces a significant overhead as the processor needs to keep track of its out-of-order instruction path so it is consistent with the in-order program. This requires additional logic structures within the processor, and these can tend to be large and exceed the size of the processor's dataflows, further complicating the processor design, and limiting processor efficiency per transistor.

### D. *Hardware Branch Prediction*

Hardware branch prediction is a critical aspect of processor design, it involves predicting the outcome of conditional branch instructions in a program, significantly impacting the instruction flow and the execution speed. Branch prediction relies on large register files to store information about branch histories and are used to make predictions about which path the program is likely to take with the aim of improving performance. However, these register files can become too large, and may not offer significant performance gains when using performance per transistor as a metric [5].

### E. *Hyper Pipelining*

In efforts to utilise this increase in transistors by increasing the throughput of instructions, computer architects have implemented parallelism into their CPU's, where CPU instructions can be executed concurrently. A CPU design method known as pipelining helps mask this instruction latency and increase the clock frequency by dividing the critical path into multiple stages. Pipelining, however, has its limitations due to problems such as 'branch miss', causing stalling and flushes [9]. Hyper-pipelining introduces higher code complexity, limits the clock frequency and increases instruction latency due an increase in the length of the critical path, and it was clear that increasing pipeline depth was resulting in decreased performance, and have negative implications on power consumption. The effectiveness on the pipeline depth will depend on factors such as the workload being run, and the specific processor architecture, as different architecture will target a different application, and will be more optimised for higher design frequencies. In efforts to utilise this increase in transistors, microprocessor companies in the early 2000s have produced processing units that work in parallel, such as IMB's dual core Power 4 CPU [8] and give the task of parallelising the code to the programmer.

## IV. Efficiency of Processor Design

In today's world, it is an observation that the number of transistors in an integrated circuit is capable of being doubled every two years, as observed by Moore's law [5]. At the same rate it has been shown that microprocessors have become less efficient in the integration of an increase in transistors. This section addresses some efficiency improving mechanisms that a computer architect will consider when designing such a CPU and provides rationale for the overall architectural decisions of the Cell/B.E. on a performance per transistor basis.

### A. *Muti-core Processor designs*

One of the most effective methods for increasing processor efficiency is to increase the amount of processor cores on a single chip. A multi-core design will be a more efficient utilization of transistors than a single or a reduced number of cores with a higher throughput per core [9]. Having multiple cores allows for parallel processing, where different threads or tasks can be executed at the same time, leading to a higher utilization. With this approach, it becomes possible to use simpler and more efficient architectures, such as in-order and scalar designs, which can help improve efficiency. Due to this design, programmers are encouraged to develop applications that utilize multi-threading, where different parts of the program can run on multiple cores. Multi-core designs were traditionally used on Symmetric Multiprocessor (SMP) systems, which are systems with multiple processor units that share memory

[9]. This was the foundation of the Cell processor architecture, as it was found that a higher core processor can lead to better performance on many applications, as it has the potential for code to be executed in parallel across the multiple cores. Continuously increasing the number of cores on a processor design, regardless of an increase in the amount of transistors will cause it to be more efficient and hence will have a better processor performance.

### B. *Three-level storage*

The memory wall is a limitation in conventional computer architectures, caused by the delay of main memory access to the registers of the processor. By scheduling data and code transfers ahead of time by allowing them to be transferred concurrently, this limitation can be mitigated. To make the most out of available memory bandwidth and reduce memory latency, pipelining data transfers in a continuous, overlapping sequence will help to utilise the memory more efficiently. Optimising this memory usage will lead to immediate performance improvements, especially for computer-intensive sets of code.

### C. *Large Register files*

A large register file is valuable for processors with high clock frequencies and deep pipelines, providing storage for instructions in flight. Having a large register file that is capable of different types of instructions to be stored in it can streamline register management and reduce hardware complexity. This will also allow loop unrolling, optimizing the execution speed through combining similar executions together with the compiler. A single register file for all instruction types will reduce cost, complexity and streamlines register management.

### D. *Software Branch Prediction*

Both software and hardware branch prediction strategies improve the instruction execution flow of a microprocessor. Hardware branch prediction in modern microprocessors can be costly on a performance per transistor basis [5]. Software based branch prediction, however, relies on information provided by the software or compiler and is an inexpensive means to enhance the performance as it doesn't require any hardware structures. Compilers can use optimisation techniques to further improve software branch prediction accuracy.

### E. *SIMD dataflow Organisation*

Single Instruction, Multiple Data (SIMD) is an effecting parallel processing architecture approach which enables multiple executions of data through one instruction. A similar approach is vector instructions, which allow the processing of long vectors of data, well suited for supercomputing applications. [5] Similar to SIMD, these vector instructions can process multiple data elements in parallel, and the vector length can vary depending on the architecture. This computing method is often found in specialised processors which generally perform computer-intensive tasks such as weather modelling and scientific simulation.

## V. CELL PROCESSOR ARCHITECHTURE

The Cell processor architecture is based on distributed processing, comprised of hardware and software cells. The architecture also has a high degree of scalability, allowing multiple devices comprising of Cell processors to cooperate with computational tasks. The processor consists of eight Synergistic Processing Elements, which act as individual processors controlled by one PowerPC based Power Processing Element. Additional components include the Direct Memory Access Controller, an XDR memory controller and a FlexIO interface. The many elements of the processor are connected by a high bandwidth interconnect, the Element Interconnect Bus.

The processor contains approximately 235 million transistors and is capable of clock speeds greater than 4GHz. It has a memory bandwidth of 25.6GB/s, and an I/O bandwidth of 76.8GB/s. The processor has the potential to calculate up to 256 GFLOPS of single precision operations at 4GHZ. There are many variants of the Cell processor. The one designed for the PlayStation 3 was clocked at 3.2GHz, limited, due to heat dissipation, economics and manufacturing constraints. It featured seven SPEs, six for game code and one for the operating system. Other variants such as ones used in consumer electronics contained only 6 SPEs.

### A. *The Power Processing Element*

The Power Processing Element (PPE) is used in the Cell as a conventional microprocessor core, with its primary role is to set up tasks for the SPEs to perform. It typically runs the operating system and most applications, as well as determines which tasks should be performed by the SPEs. The PPC has a 64-bit architecture, and an instruction set similar to many PowerPC and Power processors. The PPE follows simpler architecture, being a dual threaded RISC, in-order processor. This simpler design also consumes less power. Being a simple, efficient design, the PPE benefits from the Cells' high bandwidth memory and I/O systems, as well as its high clock frequency and dual-threading capabilities. The PPE is able to run multiple operating sytems, inheriting some technology from IBM's PowerPC series processors which contain similar hypervisor technology. The PPE also includes support for VMX (Vector Multimedia Extension), which can accelerate various tasks [8].

Being a general-purpose core allows the Cell uses to perform well on most general-purpose code [12]. The key to the Cell processor is its collaborative processing model, where most computer intensive tasks are offloaded to the SPEs. This ensures that even if there is a performance gap with the PPE, that this does not impact the overall system performance. Although its reduced design may lead to lower instructions per cycle, this potential performance difference is made up for its potential to run higher clock speeds.

### B. *The Synergistic Processing Elements*

The Synergistic Processing Elements (SPE) process the bulk of the operations of the Cell and execute them in

parallel amongst the 8 cores. Each SPE is comprised of a Memory Flow Controller (MFC) alongside a Snippet Processing Unit (SPU) and an Atomic Unit, and carry out two operations per clock cycle. One operation, done by the MFC, provides a memory access translation operation, and the SPU does one execution operation. The atomic unit provides synchronisation between the other SPEs. The data from the SPE can be moved to the main memory via the EIB. Each of the eight SPEs function as an independent processor and are equipped with a 128x128-bit register file [12]. The SPU performs SIMD vector executions, and contains a local storage (SRAM). Similar to the PPEs, the SPU performs its executions in-order, and it is up to the compiler to construct parallelism by distributing tasks between the eight SPEs. The processing power of a single SPU is capable of executing 32 GigaFLOPS of integer operations per second when running at 4GHz, despite measuring only 15 square millimeters in size. Each SPE will consume less than 5 watts of power when running at 4 GHz [12]. Despite its power efficiency and size, it can still deliver performance results comparative with high-end desktop processors for specific types of code.

The SPEs are designed for vector processing, they can perform multiple operations simultaneously using a single instruction. This makes them capable of four 32-bit instructions per cycle [12]. To efficiently harness the capability of the SPEs, the program must be vectorized. The SPEs can also operate on non-vectorized code, but its efficiency will depend on the compiler; some compilers can auto-vectorize the code without intervention from the developer.

*1) SPE Instruction Set Architecture*

The SPEs ISA combines elements of VMX (Vector Multimedia Extensions) as well as the PlayStation 2's Emotion engine ISA [12]. SPE programs can perform both vector and scalar operations for non-vectorizable code, allowing for a wide range of computational tasks, including single and double precision operations that are 4, 8, 16, 32 and 128-bit in length. The ISA also supports saturation arithmetic as well as rounding modes for single precision operations, as well as for logical operations for manipulating data. It also includes byte operations such as permute, shift and rotate, which can be applied to a 128 bit register from a set of 128 registers, which allow data storage and manipulation. Since SPEs can also perform operations for branches, loads, stores and more. Exchanging data with the rest of the system is done through the local store.

### C. The Element Interconnect Bus and The Dynamic Memory access Controller

The Element Interconnect Bus (EIB) and the DMAC (Dynamic Memory access Controller) do not perform processing tasks, they are still a crucial aspect for managing data flow withing the cell architecture. The DMAC handles memory access control for the SPEs and the PPE, with a memory access request queue of 128. Together, they are responsible for coordinating memory access, data flow and communication within the Cell processor, and are highly efficient in handling the large volumes of data that the various Systems on Chips (SOC) process and transfer.

The EIB along with the DMAC in the processor, operate at half the speed of the PPE and the SPEs, and allow for a central communication channel amongst the various systems, including the memory and I/O. It comprised of four sixteen-bit wide circular data rings, and each data ring will allow up to three separate data transfers to occur at the same time. It also contains one address bus. The data ring of the EIB operates by processing credit requests from each component, giving priorities to more optimal transfers as the ring does not allow interference of two transfers on the same data bus. The EIB also prioritises the memory controller to prevent stalling of the component requesting to read memory, whilst the remaining components are given shared priorities.

As the EIB operates at half the processor frequency, along with its 16 bytes per component, per clock cycle, the peak bandwidth of the EIB at 4GHz can be given by [11]:

$$128 \text{ Bytes} \times 2\text{GHz} = 256 \text{ GB/s}$$

The effective data bandwidth will depend on the number of hops per data transfer, i.e., the distance between the source and the receiver relative to each other and latencies due to components inhibited from using the same ring [9]. Hence, it can be shown that the number of SPEs is inversely proportional to the maximum bandwidth of the EIB.

### D. Rambus XDR RAM and the Rambus FlexIO Interconnect

The XDR RAM and FlexIO interconnect technologies are designed for the scalability and flexibility of the Cell processor and are high bandwidth and low latency. The XDR RAM has a bandwidth of 25.6Gb/s and is used for memory in the Cell architecture. The memory capacity of the XDR interface is configurable and can be connected to a large amount of memory. The I/O interconnect is used when connecting multiple Cell processors together and when the SPEs need to access memory from another Cell. In such a way, a longer stream can be created with multiple Cell processors. This I/O interconnect consists of twelve 8-bit busses, seven of which are outgoing and five are incoming; each running at 6.3 gigabits per second [12]. The brings the total bandwidth of the interconnect to 76.8 gigabytes per second. In this way, the Cell/B.E. has a high potential for scalability. An example of which can be seen in IBM's Cell Blade system consisting of 64 Cells and capable 16 teraflops in processing power [12].

## VI. Further Advantages of the Cell Processor

### A. SPE Local stores

Cell architecture does not include a local cache unlike other conventional CPUs that use caches to improve memory access performance. Instead of caches, the cell architecture uses eight 'local stores,' one for each SPE, acting like an on-chip memory, but having different design and purpose than caches. The SPEs perform operations on

registers that are read from or a the local stores. The output is of the SPE is stored back into the stores, and then can access the main memory directly in 1-16KB blocks. Therefore, the SPE can move data to and from these stores, without directly interacting with the main memory. The architects of the Cell processor didn't include a cache to decrease hardware complexity, and by doing so, making it more efficient. Not including a cache will also increase transistor usage efficiency, as there will also be additional complex logic required to operate a cache.

The local stores operate at 16 bytes per cycle, which is equivalent to 64GB/s. Caches offer similar data rates, however, only in short bursts. A local store can continue to maintain this high data transfer rate without needing to access the RAM. A problem that may arise from the local stores is contention, where the data transfer to the SPEs and the reading from the memory operations may compete for resources. This is mitigated allowing a significant amount of data to be transferred in one go. The local store design allows for ease of scalability of the Cell processor. The number of SPEs can be increased with relative ease, and scaling such a system is much easier than conventional processors that have a conventional cache.

Local stores in the Cell's SPEs allow for efficient processing and are advantageous for applications that require low latency and rapid data access. An example of where this is beneficial is audio processing applications, which requires minimal delays due to human auditory sensitivity. These applications benefit from the local stores, as processing can be extremely fast due to no memory access to slow it down, and data can be processed in small blocks which fit in the local store. Due to the limited size of a conventional processor cache, it is challenging to ensure all data for a required task is available, and hence will produce some memory latency.

### *B. Stream Processing*

The Cell processor architecture has the ability to chain multiple SPEs together and together act as a stream processor. The output of an SPE at a local store can be read by the next SPE in the sequence, allowing for a continuous processing of data. Depending on the application, the process can involve multiple SPEs, and each SPE can also access the main memory. This stream processing design allows the processors to be chained in tandem to tackle more complex and computer intensive tasks. In conjunction with this, the Cell is equipped with the high bandwidth interconnect system, allowing multiple communications streams to occur simultaneously which further enhances the parallel processing capabilities of the Cell. The Cell will achieve its theoretical maximum performance when executing computer intensive streaming applications, with a bandwidth far greater than general desktop processors.

### *C. Distributed Processing*

The fundamental parallel architecture of the Cell offers the potential for significant scalability and adaptability, allowing for distributed processing with multiple Cell processors. When more computing power is required, additional cells can be connected together, and the system can distribute software cells between them. This allows the user to adapt the Cell to accommodate their specific needs, harnessing the collective processing power of multiple Cells. This concept can also be further extended to a network of Cell processors, where software cells can be distributed between various resources, regardless of physical location, aligning with trends in distributed and edge computing. Practical implementations of this concept will introduce challenges such as synchronisation and communication between distributed processors, as well as network infrastructure and software support.

## VII. Developer Environment for the Cell Processor

Developers must unlock its full potential of the Cell processor by utilise multi-threading effectively, so the software is optimised for parallel execution. Proper cache management is also essential, as well as optimising local memory (local store) associated with each SPE to reduce the need for memory access. Code will also need to be optimised for branch predictions on instructions in order to reduce branch mispredictions [11]. Optimisation for stream processing will also reduce memory access requirements, as well as proper coordination between the SPEs.

The primary programming language that is used for developing on the Cell processor is C, as well as C++ and Fortran [11]. Developing on the cell processor includes concepts such as multithreading, cache management and SIMD, also programming techniques such as SSE, VMX and AltiVec. Thread synchronisation techniques can be implemented in the code to control the execution on the Cells different cores. Developers on the Cell processor have multiple options for managing and optimizing code execution, unlike the PlayStation 2, which was restrictive. The operating system manages the tasks given to the SPEs, or it can alternatively be compiled into applications. Developers also have the option to create their own assembly code to create a task distribution system. Sony have described a method to utilise SPEs, where program defined jobs are organized into a queue, and an instruction is given to each SPE. Upon the completion of a task, the next job in the queue is given to that SPE [12]. In this system, the job assignment is controlled by the PPE, and the SPEs are dynamically assigned tasks so developers don't need to worry about which SPE is performing which task.

Support for Linux on the Cell was established in the early development process. Linux is advantageous for the Cell, due to its compatibility with PowerPC instruction set architecture found on the PPE. The processor is designed to be portable, allowing multiple operating systems to run on the Cell platform. Toshiba implemented a software for the Cell processor, based on the Linux OS, where tasks can be executed using one or more SPEs [12]. Using this, they demonstrated the capabilities of the Cell processor to decode and process 48 standard definition MPEG2 streams simultaneously on a single HDTV, running smoothly with no framerate drops. This showcased the potential consumer applications of the technology, and showed the practical use for the media industry, which often require high-

performance processing capabilities to handle multiple video streams.

Porting and optimising an application for the Cell processor can unlock the full potential of its architecture, especially for computer intensive tasks that can benefit from parallel processing on the SPEs [12]. The first step is to port the application to the PowerPC ISA so it is compatible with the Cell's PPE. Following this, the developer must decide which tasks will benefit from the parallelism of the SPEs. The code moved to the SPEs should be isolated and should run without excessive dependency on the rest of the application. Initially, the synchronisation and the communication code is set up to interact with the SPE in a non-vectorized form, and later can be further optimised with vectorized SIMD units to take full advantage of its vector processing ability.

## VIII. Conclusion

The Cell Broadband Architecture follows an aggressive design strategy, maximising performance through a combination of vector processors, memory subsystems and interconnects that set it apart from conventional processor design. This makes it suitable for computer intensive tasks, with a claim to be up to ten times faster than some of the top-tier desktop processors of its time. The Cell architecture includes local stores, a memory subsystem to support the processing power capabilities of the SPEs, which replaces conventional caches. It allows for parallel programming, where the operating system can distribute tasks efficiently between the SPEs, as well as the capability of distributed processing, where Cells can be connected directly through interconnects, or distributed over a network.

The Cell processer represents a significant transformation from the ordinary from the traditional processor design and introduced a revolutionary microprocessor architecture designed to deliver exceptional performance improvements and advantages. Being a general-purpose processor, the Cell can be suitable for a variety of applications beyond gaming and graphics. Its innovative approach sets it apart from conventional design, allowing for the integration of higher clock speeds, as well as several cores on a single chip. The design focuses on power efficiency, and a simpler design, with performance per transistor in mind. This high performance at low power consumption makes it revolutionary for constrained environments. The Cell is designed to be produced in large quantities, making it a powerful and yet cost-effective solution for adaption to various domains that require high-performance computing needs. The simplified programming model, which is highly accessible and adaptable, shows the potential for development and optimisation of applications with the right tools and techniques.

## References


[1] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer and D. Shippy, "Introduction to the Cell multiprocessor," in IBM Journal of Research and Development, vol. 49, no. 4.5, pp. 589-604, July 2005, doi: 10.1147/rd.494.0589.

[2] B. Kuchera. "Sony answers our questions about the new PlayStation 3." Arstechnica. https://arstechnica.com/gaming/ (accessed Sep. 13, 2023).

[3] P. Thurrott. "Sony Ups the Ante with PlayStation 3." Waybackmachine. https://web.archive.org/web/20070930155439/ (Acessed Oct 01, 2023).

[4] M. Linklater. "Optimizing Cell Core". Game Developer Magazine, April 2007. pp. 15–18.

[5] H. P. Hofstee, "Power efficient processor architecture and the cell processor," 11th International Symposium on High-Performance Computer Architecture, San Francisco, CA, USA, 2005, pp. 258-262, doi: 10.1109/HPCA.2005.26.

[6] S. Koranne, "Practical Computing on the Cell Broadband Engine," Boston, MA, Springer US, 2009, pp. 3-71.

[7] T. Chen, R. Raghavan, J. N. Dale and E. Iwata, "Cell Broadband Engine Architecture and its first implementation—A performance view," in IBM Journal of Research and Development, vol. 51, no. 5, pp. 559-572, Sept. 2007, doi: 10.1147/rd.515.0559.

[8] C. R. Johns and D. A. Brokenshire, "Introduction to the Cell Broadband Engine Architecture," in IBM Journal of Research and Development, vol. 51, no. 5, pp. 503-519, Sept. 2007, doi: 10.1147/rd.515.0503.

[9] M. W. Riley, J. D. Warnock and D. F. Wendel, "Cell Broadband Engine processor: Design and implementation," in IBM Journal of Research and Development, vol. 51, no. 5, pp. 545-557, Sept. 2007, doi: 10.1147/rd.515.0545.

[10] A. E. Eichenberger et al., "Using advanced compiler technology to exploit the performance of the Cell Broadband Engine™ architecture," in IBM Systems Journal, vol. 45, no. 1, pp. 59-84, 2006, doi: 10.1147/sj.451.0059.

[11] M. Gschwind, D. Erb, S. Manning and M. Nutter, "An Open Source Environment for Cell Broadband Engine System Software," in Computer, vol. 40, no. 6, pp. 37-47, June. 2007, doi: 10.1109/MC.2007.192.

[12] N. Blachford "Cell Architecture Explained" Blackford.info. http://www.blachford.info/computer/Cell/Cell0_v2.html (Acessed Oct 14, 2023).

[13] "Best Inventions of 2008". Time Magazine. Dec. 2008.

[14] M. R. Betker, J. S. Fernando and S. P. Whalen, "The history of the microprocessor," in Bell Labs Technical Journal, vol. 2, no. 4, pp. 29-56, Autumn 1997, doi: 10.1002/bltj.2082.