MASTER

Exploring boundaries in game processing

Oostindie, M.C.N.

Award date:
2006

Link to publication
Master's Thesis

EXPLORING BOUNDARIES IN GAME PROCESSING

M.C.N. Oostindie

Supervisor: prof. dr. H. Corporaal
Coach: dr.ir. A.A. Basten
ir. S. Stuijk
Date: Februari 2006

The Faculty of Electrical Engineering of the Eindhoven University of Technology does not accept any responsibility regarding the contents of Master's Theses
Abstract

Modern 3D computer games consist of several highly computationally intensive tasks, with a high memory bandwidth requirement. The Sony PlayStation 2 is a popular game console that contains several special processors (called the Emotion Engine) to offer these performance requirements at a competitive price. To achieve this performance parallelism in applications need to be found and exploited.

To investigate the possible application speedup different application mappings have been done using a JPEG decoder and several 3D rendering kernels. The 3D kernels are used because the most speedup is to be expected in these 'native' algorithms. To investigate more general multimedia processing we used a JPEG decoder because it contains the block based IDCT image decompression found in most popular video coding standards.

We investigate why different mappings give the specific speedup. Experiments show that applications that do not easily fit in the typical 3D rendering stream found in games, do not map that well on the hardware and thus give only a limited speedup. We propose several changes to the PlayStation 2 architecture to overcome these limitations and make the PlayStation 2 more suited for other application domains. We compare this new version with the upcoming PlayStation 3 Cell architecture.

We conclude that the high processing power offered by the PlayStation 2 architecture is difficult to use in applications that differ much from 3D games. Some of the suggestions we make for a better PlayStation 2 are present in the Cell processor, that is targeted at more applications than games.
Contents

1 Introduction ............................................. 7
   1.1 Problem statement .................................... 7
   1.2 Objectives ............................................ 8
   1.3 Related work .......................................... 8
   1.4 Overview ............................................. 8

2 Application domain ....................................... 9
   2.1 MPEG Video decoding .................................. 9
       2.1.1 VLD ............................................. 10
       2.1.2 Inverse Quantizer ................................. 10
       2.1.3 IDCT ........................................... 10
       2.1.4 Motion Compensation .............................. 11
       2.1.5 Colour Conversion ................................. 12
       2.1.6 Conclusion ...................................... 13
   2.2 3D Graphics rendering ................................. 14
       2.2.1 3D graphics pipeline ............................. 14
       2.2.2 Homogeneous transformations ..................... 14
       2.2.3 Conclusion ...................................... 16

3 PS2 architecture description ............................. 17
   3.1 R5900 Emotion Engine CPU core ....................... 17
       3.1.1 MIPS core ...................................... 18
       3.1.2 Floating Point Unit .............................. 19
       3.1.3 Vector Units (VU0 and VU1) ...................... 19
       3.1.4 Differences from IEEE 754 ........................ 20
       3.1.5 IPU .............................................. 21
       3.1.6 DRAMC .......................................... 21
       3.1.7 Main Bus ........................................ 21
       3.1.8 Graphics Interface ............................... 22
   3.2 Graphics Synthesizer .................................. 22
   3.3 IOP ................................................... 23
   3.4 Software and tools .................................... 24
CONTENTS

A  Glossary  49
Bibliography  51
Chapter 1

Introduction

In the field of consumer electronics there is a large, still growing, market for sophisticated multimedia applications. The computer games industry, with big players such as Sony, Nintendo and recently also Microsoft has a traditional leading role in delivering new state of the art platforms. In the race for the most flashing 3D graphics and realistic gameplay they deliver very powerful game consoles at low costs yet still have a long lifetime. Most consoles are around for 6 years before a new generation is introduced.

To build a system with such specifications (the PlayStation 2 can perform 6.2 GFLOPS and has an internal bus bandwidth of 3.6GB/s), a special platform design is needed. A standard desktop computer will generally not be sufficient, being either too slow or too expensive. The exception is the X-Box by Microsoft, which was almost a normal Intel PC and has about the same performance as the older PlayStation 2. They did this however to enter the market quickly by using an existing design instead of starting from scratch. The console is already being replaced after only 4 years.

The remainder of this chapter is organised as follows: In Section 1.1 we will formulate the problem statement. The objectives of this thesis are the presented in Section 1.2. Related work is listed in Section 1.3. An overview of the structure of this document is presented in Section 1.4.

1.1 Problem statement

At the introduction of the PlayStation 2, in 1999, the console was hyped to have the processing power of super computers while at the same time being much cheaper than normal desktop PCs. The question rises, while the PlayStation 2 definitely has a lot of processing power, why it didn’t appear in some form or another on the desktop. Until now, there hasn’t been another instantiation of the architecture, and since its successor, the PlayStation 3, is bound to be released, it is highly unlikely that there will ever be one.
1.2 Objectives

We will investigate the PlayStation 2 architecture, what possibilities there are for high performance processing and in what kind of applications the can actually be achieved. We will present an improved version of the PlayStation 2, and compare this virtual architecture with the PlayStation 3/Cell architecture.

1.3 Related work

In [11] the possibilities to use the PlayStation 2 for scientific computing (in chemistry). They achieved some speedup, but no dramatic improvements. They point to Operating System overhead and limited memory as the main bottlenecks.

1.4 Overview

The outline of this Report is as follows: We first describe the Application domain we are interested in in Chapter 2. Chapter 3 discusses the PlayStation 2 architecture in detail. In Chapter 4 we explain different techniques to increase the computational power of processors using different forms of parallelism. We explain where the PlayStation 2 fits in the picture. In Chapter 5 we investigate the actual performance that can be achieved on the PlayStation 2. We do this by mapping several kernels (small programs) on architecture. The results of these experiments are presented and explain why the specific performance is achieved. We investigate why the performance not always meets our expectations. In Chapter 6 we suggest improvements on the PlayStation 2 architecture to increase the performance of the platform. We compare these suggestions in Chapter 7 to the new PlayStation 3 Cell architecture. Finally, in Chapter 8 the work is summarized and conclusions are drawn. Recommendations for future work are given. Common abbreviations are explained in the glossary, Appendix A.
Chapter 2

Application domain

In this chapter we will describe two types of application domains which are both very common and computational intensive inside multimedia processing. The first is MPEG-2 video decoding, which is a block based video coding standard. It is a popular video coding standard, being the format of choice for DVD and satellite stations. The second application are the drawing of real-time 3D graphics. The 3D environments found in most current computer games require many operations for each pixel inside the screen to create a realistic 2D image.

2.1 MPEG Video decoding

MPEG video is a block based video coding standard. A video stream consists of a sequence of frames, which are divided into blocks of 8 x 8 pixels each. Both spatial and temporal redundancies are used to reduce the required information to describe each frame.

In Figure 2.1 a block diagram of an MPEG video decoder is shown. The input of the decoder is a bit-stream containing the compressed video information. This information is Huffman decoded in the Variable Length Decoder (VLD). The stream contains both image (colour) information and motion vectors. The quantized image information is dequantized using an Inverse Quantizer and then transformed into the actual pixel representation using an Inverse Discrete Cosine Transform (IDCT). There are three types of frames, Intra (I), Predictive (P) and Bi-Predictive (B). The P and B frames exploit temporal redundancy (similarities between different frames) to achieve higher compression using previous decoded I and P frames and the supplied motion vectors inside the Motion Compensation block as shown in Figure 2.3.

This motion compensated frame is added to the new frame resulting in the Current Image. The resulting image is converted to red green and blue (RGB) colour space using the Colour Conversion block.
2.1.1 VLD

The first step in decoding an MPEG bitstream is decompressing the bitstream in 8 bit quantized DCT-coefficients. The compressed values have a variable length ranging from 1 to 16 bits, with an average length much lower than 8 bits for each 8 bit value.

2.1.2 Inverse Quantizer

The decoded DCT coefficients that are quantized, and need to be dequantized. The dequantizing consists multiplying every coefficient with a specific constant, which is unique to every position in the $8 \times 8$ block. The dequantizing process thus involves looking up of a constant and multiplying it with the decoded coefficient, totalling 64 loads and multiplications per $8 \times 8$ block.

2.1.3 IDCT

Spatial redundancies are similarities inside a frame, such as a smooth area of the same colour, e.g. a blue sky. The Discrete Cosine Transform (DCT) is used for spatial redundancies. In Equation 2.1 the 2D DCT is shown which is used to transform the image samples of an $8 \times 8$ block into an $8 \times 8$ block of DCT-coefficients. To recreate the image samples an Inverse DCT (IDCT) 2.2 is used.
\[ S(u, v) = \frac{C(v)}{2} \frac{C(u)}{2} \sum_{y=0}^{7} \sum_{x=0}^{7} s(x, y) \cos \left( \frac{(2x+1)u\pi}{16} \right) \cos \left( \frac{(2y+1)v\pi}{16} \right) \] (2.1)

\[ s(x, y) = \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{C(v)}{2} \frac{C(u)}{2} S(u, v) \cos \left( \frac{(2x+1)u\pi}{16} \right) \cos \left( \frac{(2y+1)v\pi}{16} \right) \] (2.2)

All the cosines are constant values, which can be pre-calculated. The meaning of the symbols in Equations 2.1 and 2.2 are:

\[ C(x) = \begin{cases} 1, & (x = 0) \\ \frac{1}{\sqrt{2}}, & (x > 0) \end{cases} \]

\[ s(x, y) = 2D \text{ sample value} \]

\[ S(u, v) = 2D \text{ DCT coefficient} \]

Because the 2D IDCT is separable, it can be computed by applying two times eight 1D IDCTs sequentially, one in the horizontal (row) direction and one in the vertical (column) direction (see Equation 2.3). This reduces the number of operations by a factor of four. The cosines and the constants \( C(u) \) in a 1D IDCT (Equation 2.4) can be combined. The resulting transform requires 64 multiplications and 56 additions, or 1024 multiplications and 896 additions for each 8 \( \times \) 8 block, which can then be dequantized. Much research has been done on implementations using less operations, where the most optimal known is published by Arai, Agui and Nakajima (their paper is in Japanese, however, an English description can be found in [8]). Their implementation (see Figure 2.2) requires only 13 multiplications and 29 additions for each 1D IDCT. When dequantization has to be performed afterwards, 8 of the multiplications can be combined with the dequantization, which leaves only 5 multiplications for each 1D IDCT.

\[ s(y, x) = \left[ \sum_{v=0}^{7} \frac{C(v)}{2} \cos \left( \frac{(2y+1)v\pi}{16} \right) \right] \left[ \sum_{u=0}^{7} \frac{C(u)}{2} S(v, u) \cos \left( \frac{(2x+1)u\pi}{16} \right) \right] \] (2.3)

\[ s(x) = \sum_{u=0}^{7} \frac{C(u)}{2} S(u) \cos \left( \frac{(2x+1)u\pi}{16} \right) \] (2.4)

### 2.1.4 Motion Compensation

Temporal redundancies are similarities between sequential frames, such as a background. There are three types of frames, Intra (I), Predictive (P) and Bi-Predictive (B). The P and B frames exploit temporal redundancy (similarities between different frames) using previous decoded I and P frames and the supplied motion vectors inside the Motion Compensation block as shown in Figure 2.3.
2.1.5 Colour Conversion

A display, such as a computer screen, uses red, green and blue components to create colours. Inside MPEG, however, colours are created by a luminance ($Y$) and two chrominance ($C_r$ and $C_b$) components. These components need to be transformed using the transformations in Equations 2.5–2.7. For every pixel we need 4 multiplications and 4 additions, resulting in a total of $4 \times 64 = 256$ multiplications and additions per block.

$$R = Y + 1.40200 \times C_r$$  \hspace{1cm} (2.5)

$$G = Y - 0.34414 \times C_b - 0.71414 \times C_r$$  \hspace{1cm} (2.6)

$$B = Y + 1.77200 \times C_b$$  \hspace{1cm} (2.7)

Because the human visual system (the eyes) is less sensitive to colour (chrominance) information than to black and white (luminance) information, the colour information is often processed at a lower resolution using the 4:2:0 format. For every 4 blocks of luminance information there are only two blocks of chrominance information which need to be upsampled to the same resolution as the luminance blocks together (see Figure 2.5). This upsampling requires 1 addition and 1 multiplication (which can be a bitwise shift) for every generated sample value. Since three samples are generated for every original value (see Figure 2.4), a total of $3 \times 64 = 192$ multiplications and additions
CHAPTER 2. APPLICATION DOMAIN

2.1

2.1.1 Usage of I, P and B frames to exploit temporal redundancy using motion compensation.

per block are required. A combination of four luminance and two chrominance blocks is called a macro block, which has a dimension of $16 \times 16$ pixels.

Figure 2.4: Upsampling of chrominance samples. For every original sample three samples are generated as indicated by the arrows.

Figure 2.5: $Y C_r C_b$ colour sampling configuration.

2.1.6 Conclusion

As can be seen in Table 2.1, per macro block we need 6056 operations. Because motion compensation involves mostly pixel-copying, the operations needed are much lower ($768$ for a full macro block addition). When leaving motion out (and only doing Intra-picture decoding), the decoder has the same functionality as a JPEG decoder. In our experiments we will therefore only use a JPEG decoder.
2.2 3D Graphics rendering

Rendering is the process of producing the pixels of an image from a higher-level description of its components. In the case of 3D graphics it involves the drawing of 3D objects into a 2D space, such as a screen. A 3D scene is built using 3D objects, which are built using triangles (ranging from several tens to millions per object). A more detailed explanation can be found in [3].

2.2.1 3D graphics pipeline

To generate a 2D representation of a 3D scene, several conversion steps are done inside what is called the graphics pipeline (see Figure 2.6).

First the geometry translations need to done: In a typical 3D scene, a few hundred objects (ranging from simple cubes to complex models such as characters, trees and cars), are placed in the world. To do this the local object coordinates are transformed into world coordinates. After the world has been built, the world (and thus all objects) are placed in the right location relative to the viewer, i.e. the eye coordinates. To get a realistic image, perspective correction is applied in the projection step and the coordinates are translated into normalized device coordinates (i.e. screen coordinates). All these transformations are done in the frontend of the rendering pipeline. After these transformations the world is ready to be drawn into 2D space in a process called Rasterization. This is done in the backend, which is often implemented in hardware as a graphics processor.

2.2.2 Homogeneous transformations

In this Section we will explain why and how to use a 4-dimensional homogeneous coordinate-system to describe a 3-dimensional world.

<table>
<thead>
<tr>
<th>Operation</th>
<th>multiplications</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dequantization</td>
<td>384</td>
<td>0</td>
</tr>
<tr>
<td>IDCT</td>
<td>480</td>
<td>2784</td>
</tr>
<tr>
<td>Colour Conversion</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>Upsampling</td>
<td>372</td>
<td>372</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2260</strong></td>
<td><strong>4180</strong></td>
</tr>
</tbody>
</table>

Table 2.1: Number of operations per macro block for each functional block.

A more in-depth discussion on MPEG and a description of an encoder can be found in [1].
We define the vectors $X, Y, T \in \mathbb{R}^3$ and $V, V' \in \mathbb{R}^4$. The dimensions of the transform matrices $M$ and $H$ are $3 \times 3$ and $4 \times 4$ respectively, where $H$ is a homogeneous matrix.

Internally all 3D coordinate transformations are represented using 4D homogeneous coordinates. Each vector $(x, y, z) \in \mathbb{R}^3$ is uniquely mapped onto a vector $(x, y, z, 1) \in \mathbb{R}^4$. Other vectors $(x, y, z, w) \in \mathbb{R}^4$ are projected onto the hyperplane $w = 1$ by $(x, y, z, w) \rightarrow (\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1)$. Such coordinates that map to the same location on the $w = 1$ hyperplane are equivalent.

While a common transformation such as translation (moving an object to a different location) $Y = X + T$ cannot be represented as a linear transformation $Y = MX$, we can represent it as a linear transformation $V' = HV$ using 4D homogeneous coordinates.

Transformations can be applied to homogeneous coordinates to obtain other homogeneous coordinates. Such a $4 \times 4$ matrix $H$ is called a homogeneous transformation as long as the lower-right-hand element $h_{33} = 1$. Examples of various transformations are shown in Equations 2.8, 2.9 and 2.10.

\[
H_{\text{scaling}} = \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \tag{2.8}
\]
A homogeneous vertex $V$ can be transformed into $V'$ by multiplication with a transformation matrix, $V' = HV$. Multiple transformations can be combined into one transformation, e.g. $H_{\text{combined}} = H_{\text{translation}} \cdot H_{\text{scaling}}$. To transform a vertex using a $4 \times 4$ matrix, 16 multiplications and 12 additions are required. When we have a reasonable complex scene which consists of $10^5$ vertices and a framerate of 50 frames per second, a single transform per vertex requires $50 \cdot 10^5 \cdot (16 + 12) = 140 \cdot 10^6$ operations per second. Because several homogeneous transformations can be combined to a single homogeneous transformation matrix, on average less operations are needed for processing 4 dimensional coordinates instead of 3 dimensional coordinates.

2.2.3 Conclusion

We present an overview of the 3D graphics pipeline and a more detailed explanation of the use of homogeneous 4 dimensional coordinates to represent 3 dimensional coordinates. On average, 4 dimensional transformations require less operations than 3 dimensional because they can be multiple transformations can be combined into a single transformation.
Chapter 3

PS2 architecture description

In this chapter we give a detailed description of the PlayStation 2 architecture. The PlayStation 2 architecture consists of essentially three blocks: control, graphics and I/O. Control and calculations are done in the Emotion Engine (EE), graphics rendering is done in the Graphics Synthesizer (GS) and I/O is done in the IO processor (IOP).

![PlayStation 2 block diagram.](image)

In the following sections we will discuss the details of the Emotion Engine (Section 3.1), the Graphics Synthesizer (Section 3.2) and the IO processor in Section 3.3. In the final section we discuss the software and tools that have been used to develop and map software on the PlayStation 2.

3.1 R5900 Emotion Engine CPU core

In this section, we highlight some of the interesting aspects of the PlayStation 2 architecture. A more detailed description can be found in the manuals provided by Sony [10].
3.1 CHAPTER 3. PS2 ARCHITECTURE DESCRIPTION

3.1.1 MIPS core

The core CPU of the Emotion Engine (EE) (see Figure 3.2) is a MIPS III (partly MIPS IV) core running at 300MHz. It supports 2-way superscalar operation, allowing 2 instructions to be executed in parallel every cycle. It has support for non-blocking load instructions and supports 128 bit SIMD multimedia instructions. The 32 general purpose registers are 128-bit wide, although the non-multimedia instruction only use the lower 64 bits.

The multimedia instructions operate on packed integer data in 128 bit registers. The instructions fall into two categories:

- applying the same operation, such as additions, multiplications, shifts on the packed values using a single instruction,

- rearranging and conversion operations, such as converting 16 bit integers to 8 bit values or interleaving of values stored in two registers.

The CPU has an instruction cache (I-Cache) and a data cache (D-Cache). The data
CHAPTER 3. PS2 ARCHITECTURE DESCRIPTION

3.1

The cache has the ability to load a necessary word from a cache line first (sub-block ordering) and to permit a hazard-free cache-line hit while a previous load is still in process (hit-under-miss). Since hit-under-miss effect is similar to the prefetch (PREF) instruction, it is effective when the address to be accessed is known in advance. Both caches are 2-way associative. The output from the cache is also buffered in the Write Back Buffer (WBB). The WBB is a FIFO of 8 qwords. Write requests are stored here, and then written to memory according to the state of the main.

Besides 16kB instruction cache and 8kB data cache, there is a 16kB scratchpad (SPR). There are two coprocessors available to the MIPS, a normal FPU which operates on 32-bit floats, and a Vector Processing Unit (VU0).

3.1.2 Floating Point Unit

The Floating Point Unit (FPU) is a high performance single precision (see Subsection 3.1.4 for IEEE compliance) floating point unit operating at 300MHz with a throughput of 1 operation per cycle for most operations, such as multiply and add. It contains a 1 cycle throughput multiply-add operation and support for division. A total of 32 registers, each 32-bit in size, are available, giving a lot of programming flexibility. The FPU is connected to the MIPS as coprocessor cop1.

3.1.3 Vector Units (VU0 and VU1)

The EE contains two Vector Units, both operating at a clock-speed of 300MHz. A block diagram of a Vector Unit (VU) is shown in Figure 3.3. Each VU is a programmable LIW DSP containing 2 functional units (FU). One FU (called Upper Execution Unit) can perform 4 multiply-accumulates (FMAC) every cycle, while the other FU (Lower Execution Unit) executes other operations, such as divisions (FDIV), load/store (LSU), integer operations (IALU) and branch operations (BRU). A random number generator (RANDU) is provided. Besides 16 16-bit integer registers (used for addressing data and and loop counters) each VU has 32 128-bit registers capable of holding 4 32-bit float values each. Like the FPU, the floating-point operations behave slightly different from the IEEE 754 specifications (see Subsection 3.1.4). An Upper and a Lower instruction can be issued in parallel.

Each VU contains a data expansion unit called Vector Interface Unit (VIF). It can perform simple conversions such as converting 4 bytes to 4 floats.

VU0 contains 4kB instruction memory and 4kB data memory. It can operate as a coprocessor to the MIPS (macro mode) or as a stand alone processor (micro mode). The MIPS has a direct 128-bit wide access to the VU0 registers using normal load and store operations. When operating in macro-mode, only one instruction (either upper or lower) can be issued.

The second vector processing unit VU1 operates as a standalone processor with 4 times as much memory than VU0 (both instruction and data memory). This processor
has support for trigonometric and exponential functions in the Elementary Function Unit (EFU). To relieve the main bus (Subsection 3.1.7) VU1 can directly access data in the VU0 data memory.

3.1.4 Differences from IEEE 754.

The floating-point operations in both the FPU and the VUs are not fully IEEE 754 [4] compliant. Only single-precision floating-point values are supported. There can be rounding errors leading to a wrong value in the least significant bit (LSB). These small rounding errors are the same for the FPU and the VUs. Whenever an exception occurs, such as under- or overflow or a division by zero, the processing is not interrupted, but a flag is set. If desired, a program can use a flag check after the calculation to check if something abnormal did occur. In case of underflow, overflow, or a division by zero, results are saturated to the lowest or highest supported value. There is no support for non-values such as not-a-number and infinity. The rationale behind this behaviour is that a small error in the display is less of a problem than a part of a frame not being rendered at all because of exception handling.
3.1.5 IPU
To accelerate on the fly texture and video decoding, an Image Processing Unit (IPU) provides hardware accelerated block-level MPEG-2 decoding (entropy decoding, IDCT transform and colour space conversion). If motion compensation has to be performed, as is the case in the MPEG-2 video, this has to be done on the MIPS.

3.1.6 DRAMC
A memory controller (DRAMC) provides access to 32MB of main memory at a speed of 3.2GB/s. This memory can be accessed by the EE in three different modes, cached, uncached and uncached accelerated, as shown in Figure 3.4. In any mode, the data written to the memory passes through a write-back buffer (WBB) to and written to the main memory according to the state of the main bus.

![Figure 3.4: Three memory access modes](image)

In cached mode, all data written to and read from the main memory is stored in the data cache. When having a one-way data path, e.g. data generated to be used in another processing unit, it is preferable that written data is reflected in main memory immediately. This can be achieved using the uncached mode. Besides that, because this data is not used it makes no sense to cache it anyway. To speed up reading while writing synchronously, an uncached accelerated mode using a read-ahead buffer (UCAB) is also available. This buffer speeds up the reading of adjoining memory locations.

3.1.7 Main Bus
All these devices are connected to a 128-bit wide data bus that operates at 150MHz, providing a maximum of 2.4GB/s bandwidth. To accommodate efficient data transfers
there is a 10 channel DMA controller to move data around between these devices, the scratch pad and the main memory. The different devices, such as the vector units, but also the scratch pad are addressed as a source or destination using a dedicated channel.

3.1.8 Graphics Interface

The Emotion Engine is connected to the Graphics Synthesizer (see Section 3.2) using a 64-bit wide bus at 150MHz, giving a transfer rate of 1.2GB/s. Data can be sent through the Graphics Interface (GIF) by VU1 or via the main bus. When data is sent from VU1 to the GS, the main bus can be used in parallel for data transfers between other devices.

3.2 Graphics Synthesizer

The Graphics Synthesizer (GS) is the graphics processor of the PlayStation 2. It is responsible for rendering all the graphics primitives, such as triangles, to the screen. The GS runs at a clock speed of 150MHz.

\[
\begin{align*}
(a) \text{ Lines} & \quad (b) \text{ Triangles} \\
(c) \text{ Triangle strip} & \quad (d) \text{ Triangle fan}
\end{align*}
\]

Figure 3.5: Several drawing primitives support by the GS

The chip contains 4MB of DRAM as a local memory to save pixel information (frame buffer and Z-buffer) generated by the drawing functions and texture information. Because the memory is located inside the chip, it has a very high bandwidth to the memory. The frame buffer is accessed through a 1024 bit port for reading and another 1024 bit port for writing. The textures are accessed through a separate 512 bit wide access port. This leads to a maximum frame buffer bandwidth of 38.4GB/s and a texture bandwidth of 9.6GB/s.

When rasterizing the 3D scene to the screen, the GS processes 16 pixels per cycle. When applying texture mapping, an additional multiplication is required for every pixel, resulting in a throughput of 8 pixels per cycle. Besides texture mapping, the
GS is capable of shading, fogging and anti-aliasing. Depending on the configuration of operations the performance ranges from 16 million to 75 million triangles per second.

The GS supports several drawing primitives, such as points, lines, triangles, triangle strips and triangle fans. All of these are drawn by specifying their vertices. The triangle strip and triangle fan are very efficient by requiring only one vertex to specify the next triangle. Some of them are shown in Figure 3.5.

### 3.3 IOP

A separate processing unit is used for communicating with the outside world. It supports various I/O interfacing standards such as USB, IDE for the DVD drive and hard-
disk and IEEE1394 (Firewire). Access to the sound processor is also handled through this I/O processor. Exact details about both the I/O processor and the sound processor are only disclosed to Sony licensed developers, but are available through Linux kernel calls, as discussed in the next section.

PlayStation 1 legacy games can be run on a MIPS R3000 included in the I/O processor, which operates at 34MHz. This processor cannot be accessed by PlayStation 2 software. It is probably being used for (most of) the PS2 I/O operations.

3.4 Software and tools

To develop and execute software on the PlayStation 2, there are two options available:

- The TOOL, the professional development version of the PlayStation 2, which contains more memory and extra debug facilities, but requires the status of being a Sony licensed game developer, and costs a large sum of money,

- The Linux kit, aimed at enthusiasts which comes at a very modest price and turns your PlayStation 2 into a fully functional Linux desktop.

We used a Linux kit, made available by Sony. The Linux kit consists of a harddisk and network adapter that need to be installed into a standard PlayStation 2. A special version of Linux (included with the kit) can then be installed on the PlayStation 2. It is a fairly complete Linux distribution. For development of software the same GNU toolchain (including gcc, gdb and gmake) as supplied with the TOOL is used. The compiler does not generate executables for any components other than the MIPS, such as the Vector Units. These need to be programmed directly in assembler. Extensive reference manuals for all hardware (except I/O subsystems such as DVD access) are included.
Chapter 4

Parallelism in micro-architectures

In this Chapter we describe different techniques to achieve parallelism in architectures to improve performance of processors. We compare several processors with the Emotion Engine.

Credits: Most of Section 4.1 has been taken with kind permission from "Trends in programmable instruction-set processor architectures" by H. Corporaal.

4.1 Architecture design space

In order to achieve the performance of a processor, it takes more than just increasing the clock frequency. The vast majority of processors applies one instruction at a time to a single data stream. This technique is called, Single Instruction Single Data (SISD). When applying the same instruction on multiple data streams is a popular method to increase performance. This technique is called Single Instruction Multiple Data (SIMD). Having multiple processors operate in parallel is even more common, as this parallelism is the most easy one to program. This is called Multiple Instruction Multiple Data (MIMD). There also exists Multiple Instruction Single Data (MISD), but this architecture is only useful in very specific cases, and therefore not so common.

Each architecture can be specified as a 4-tuple $(I, O, D, S)$, where $I$ is the issue rate (instructions per cycle), $O$ the number of (basic monadic or dyadic) operations specified per instruction, $D$ the number of operands or operand pairs to which the operation is applied, and $S$ is the superpipelining degree. The latter is introduced by Jouppi [6], and defined as

$$S(\text{architecture}) = \sum_{\forall Op \in \text{I.set}} f(Op) \times lt(Op) \quad (4.1)$$

where $f(Op)$ is the relative frequency with which operation $Op$ occurs in a representative mix of applications, and $lt(Op)$ is the latency of operation $Op$; $lt(Op)$ indicates the minimal number of cycles after which operations, dependent on $Op$, have to be scheduled in order not to cause pipeline stalls (in case the architecture supports dependence
locking), or cause semantic incorrect results (in case the architecture does not lock on dependencies). It is related to the number of delay slots \( d \) of an operation by:

\[
lt(Op) = 1 + d(Op)
\]

For some architectures no clear numbers on the superpipelining could be found, we estimated these as being equal or close to 1.

![Diagram](https://example.com/diagram.png)

Figure 4.1: Four dimensional representation of the architecture design space.

Delay slots are invisible at the architectural level if dependence locking is used; in that case filling them with independent operations is semantically not required. However, filling them results in higher resource utilization and therefore better performance; this is part of the compiler’s job. The latency of non-pipelined operations is defined as one; \( S(CISC) \) is therefore one. The superpipelining degree indicates how many delay slots have to be filled (on average) in order to keep the processor busy.

RISC architectures are very close to the center (1,1,1,1) of the architectural design space. RISCs have the potential of issuing one instruction per cycle \( (I = 1) \), where each instruction specifies one operation \( (O = 1) \), each operation applies to a single or single pair of operands \( (D = 1) \), and the superpipelining degree is slightly larger than one \( (S \approx 1) \).
4.2 Examples

In this section we will describe some different architectures and how they achieve the level of parallelism they provide.

Typical values of \((I, O, D, S)\) for the to be discussed architectures and several processors described below are found in Table 4.1. \(K\) indicates the number of FUs or processor nodes available. This table also indicates the amount of parallelism \(M_{\text{par}}\) offered by the architecture; this number corresponds to the average number of operations in progress, at least when the hardware is kept busy, and is defined by:

\[
M_{\text{par}} = I \times O \times D \times S
\]  

(4.3)

When taking the Emotion Engine and Graphics Synthesizer from the PlayStation 2 and analyze them for the level of parallelism present, we get the values as presented in Table 4.2. When adding the values, we get a total level of 56 for parallelism, which is quite impressive. Because it is impossible to issue 2 instructions in the MIPS and one in the FPU at the same time, the FPU line is left out from the totals.

4.2.1 CISC

A Complex Instruction Set Computer (CISC) is a microprocessor instruction set architecture (ISA) in which each instruction can indicate several low-level operations, such as a load or store from memory, and an arithmetic operation, all in a single instruction. The Intel 8086 is a CISC processor. While the instruction set was rich, the instructions differed strongly in size, the number and type of operands.

```plaintext
ADD CX, DX ; Add contents of register DX to register CX
ADD AX, [VARIABLE] ; Add value pointed to by VARIABLE
                   ; to register AX
```

4.2.2 RISC

The Reduced Instruction Set Computing (RISC) evolved from the CISC. RISC processors have a much smaller and more simple (reduced) instruction set with a very limited number of addressing modes. With most instructions having the same size and performing more or less the same amount of work, they fit well in a simple pipelining scheme. An example of a RISC processor is a MIPS, which has a 5 stage pipeline.

4.2.3 Pentium 4

The Pentium 4 can be categorized as a modern "CISC" processor. It supports all the instructions from its predecessors in the x86 architecture, translating these into microcode that much more resembles RISC instructions that are executed internally. The P4 has a
very deep pipeline of more than 20 (shorter) stages. While instructions have a higher latency, because they have to go through more stages, the clockspeed can be increased. Because the average number of instructions per cycle remains the same, the actual performance increases along with the clockspeed.

4.2.4 TriMedia

The Trimedia [9] is a multimedia processor for high-performance multimedia applications that deal with high-quality video and audio. The Trimedia core is a powerful 32-bit DSPCPU. It implements a 32-bit linear address space and has 128, fully general-purpose 32-bit registers. The registers are not separated into banks. While this offers full flexibility for register usage, any operation can use any register for any operand, the complexity of such a large register file has a negative impact on the cycle time.

The core uses a VLIW instruction-set architecture and is fully general purpose. Up to five operations can be issued per cycle. These operations can target any five of the 27 functional units in the DSPCPU, including integer and floating-point arithmetic units and data-parallel multimedia operation units.

4.2.5 Imagine Stream Architecture

The Imagine Stream Architecture is a novel architecture that executes stream-based programs. It provides high performance with 48 floating-point arithmetic units and a area- and power-efficient register organization. A streaming memory system loads and stores streams from memory. A stream register file provides a large amount of on-chip intermediate storage for streams. Eight VLIW arithmetic clusters perform SIMD operations on streams during kernel execution. That is, the clusters can only perform the same instruction in parallel, so one could call it an SIMD of VLIW clusters. Kernel execution is sequenced by a micro-controller. A network interface is used to support multi-Imagine systems and I/O transfers. Finally, a stream controller manages the operation of all of these units.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>$K$</th>
<th>$I$</th>
<th>$O$</th>
<th>$D$</th>
<th>$S$</th>
<th>$M_{par}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CISC</td>
<td>1</td>
<td>0.2</td>
<td>1.2</td>
<td>1.1</td>
<td>1</td>
<td>0.26</td>
</tr>
<tr>
<td>RISC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>&gt;1</td>
<td>1.2</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>&gt;1</td>
<td>12</td>
</tr>
<tr>
<td>TriMedia</td>
<td>27</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>&gt;1</td>
<td>20</td>
</tr>
<tr>
<td>Imagine</td>
<td>48</td>
<td>1</td>
<td>6</td>
<td>8</td>
<td>&gt;1</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 4.1: Typical values of $K$, the number of FUs or processor nodes, and $(I, O, D, S)$ for different architectures.
4.3 Conclusion

In this chapter we have discussed several methods to achieve parallelism in architectures and illustrated these techniques with various existing processors. When looking at the architecture of the PlayStation, most of the processing power of this architecture comes from the high level of parallelism.

<table>
<thead>
<tr>
<th>PlayStation 2</th>
<th>$K$</th>
<th>$I$</th>
<th>$O$</th>
<th>$D$</th>
<th>$S$</th>
<th>$Mpar$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>&gt; 1</td>
<td>8</td>
</tr>
<tr>
<td>FPU</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>&gt; 1</td>
<td>2</td>
</tr>
<tr>
<td>Vector Unit (2)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>&gt; 1</td>
<td>16</td>
</tr>
<tr>
<td>Graphics Synthesizer</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>&gt; 1</td>
<td>16</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7</strong></td>
<td><strong>7</strong></td>
<td><strong>6</strong></td>
<td><strong>28</strong></td>
<td><strong>1</strong></td>
<td><strong>56</strong></td>
</tr>
</tbody>
</table>

Table 4.2: PlayStation 2 values of $K$, the number of FUs or processor nodes, and $(I, O, D, S)$.
Chapter 5

Mappings, experiments, results

In this chapter we investigate the actual performance that can be expected from the PlayStation 2. We will use two applications as a test case. The first is a JPEG decoder, because block based image-decoding is a fundamental part of most used image and video decoding and multimedia applications, like MPEG-1, 2 and 4, but also h.261 – h.264 and Windows Media Video.

Secondly we use a typical 3D rendering kernel for comparison. Since the PlayStation 2 was designed with a strong focus on 3D graphics and games, we expect to see a top performance here.

In Section 5.1 we introduce the test applications we have used in our experiments. We discuss the experiments and preset the results in Section 5.2. We draw conclusions based on these results in Section 5.3.

5.1 Test applications

In Sections 5.1.1 and 5.1.2 we introduce the application we have used in our experiments.

5.1.1 JPEG overview

A JPEG [8] bit-stream consists of general image information, table specifications and the actual compressed image data. The general information describes the information such as the dimensions of the image and the colour space, e.g. Y, YCrCb, CMYK. The tables specify the Huffman codes used for the entropy coding and the quantization levels. The remaining part consists of the compressed image data. The image is divided in blocks of 8 by 8 pixels. Most JPEGs have their colour information stored at half the resolution of luminance (black and white) information, so for every 4 luminance blocks there are 2 colour blocks. These 6 blocks together form a Macro block (MB) with a resolution of 16 × 16.
5.1 CHAPTER 5. MAPPINGS, EXPERIMENTS, RESULTS

Figure 5.1: Block diagram of a JPEG decoder.

Figure 5.1 shows a block diagram of a JPEG decoder. The input of a JPEG decoder is a bit-stream connected to the demultiplexer block (DMX). The DMX block extracts the tables and general information necessary for decoding the image from the input stream. The remaining part is the actual compressed data, which is passed on to the variable length decoder (VLD), which performs Huffman decoding followed by runlength decoding to generate 64 image coefficients. These coefficients need to be dequantized by the Inverse Quantizer (IQ) and reordered by the Inverse Zigzag Scan (IZS) to rebuild the original block of 8 by 8 coefficients. This block is then transformed into the original colour information by applying a two dimensional Inverse Discrete Cosine Transform (IDCT). The resulting colour channels need to be transformed in the desired colour channels, e.g. YCbCr → RGB conversion (CCV).

5.1.2 3D Perspective Matrix Transformation

To compute the view of a 3D model, the 3D world coordinates of the vertices in the model must be converted to 2D screen coordinates, which determine which screen pixel represent each vertex. This conversion depends on the position and orientation of the virtual camera in the 3D space and on the type of projection desired.

Figure 5.2 shows the basic configuration for a perspective viewing transformation. The 3-dimensional objects which are inside the view frustum are projected onto the display window. This frustum, which is coloured grey in the figure, is the volume bounded by the near and far planes and the other for sides defined by (top, left) and (bottom, right). To transform the 3D coordinates \( V_{obj} = (v_x, v_y, v_z) \) of these objects to 3D coordinates which are suitable for displaying on the screen, they must be transformed using the perspective matrix \( M_{\text{perspective}} \) defined in Equation 5.1.

\[
M_{\text{perspective}} = \begin{pmatrix}
2\text{near} & 0 & \text{right-left} & 0 \\
0 & 2\text{near} & \text{top-bottom} & 0 \\
0 & 0 & \text{top+bottom} & 0 \\
0 & 0 & -1 & 0 \\
\end{pmatrix}
\]

(5.1)
CHAPTER 5. MAPPINGS, EXPERIMENTS, RESULTS

5.2 After the transform in Equation 5.2 the coordinates need to be converted back to homogeneous coordinates which can be used by the Graphics Synthesizer, shown in Equation 5.3.

\[
\begin{pmatrix} v_x' & v_y' & v_z' & v_w' \end{pmatrix}^T = M_{\text{perspective}} \cdot \begin{pmatrix} v_x & v_y & v_z & 1 \end{pmatrix}^T
\]

(5.2)

\[
V_{gs} = \begin{pmatrix} v_x' & v_y' & v_z' & v_w' & v_w' \end{pmatrix}^T
\]

(5.3)

5.2 Experiments

To achieve performance improvements, we focused on both computation and memory operations.

5.2.1 DTSE

For the memory improvements, we applied Data Transfer and Storage Exploration (DTSE), a technique that has been researched at IMEC. These techniques aim at improving the locality and reducing the lifetime of temporary variables during processing. If more operations are to be performed on a set of data, it is more efficient to apply all of them in one pass over the data instead of applying them all in separate passes. This reduces the need for temporary load/store operations, or at least improves cache usage.

Because the JPEG standard was written with limited memories in mind, there is little to gain in applying DTSE techniques. At most one macroblock totalling 384 bytes of
active data needs to be in memory. In our 3D kernel long streams of coordinates get processed, where DTSE won’t be of any help also. While we cannot reduce the number of load/store operations any further, we did investigate the possibilities of using the scratchpad memory instead of the main memory. However, explicitly moving data to the scratchpad memory did not give any improvements. The cache controller performs very good in prefetching consecutive data, and offers sophisticated methods of control as described in Section 3.1.6 when necessary.

### 5.2.2 JPEG

For the JPEG decoder we started using the JPEG decoder software written by the Independent JPEG Group (IJG) [5]. It is a high quality widely available and used implementation in C of the JPEG standard. As a reference for high performance, we use the Image Processing Unit. There are no specifications available on decoding speed, but since this unit is used for playback of DVDs, it should at least be fast enough for decoding an MPEG-2 PAL Standard Definition stream. MPEG-2 block decoding differs from JPEG only in using fixed Huffman tables for the Variable Length Decoding.

Because the JPEG decoder mostly works only at one block at a time, there is little to gain from DTSE.

PAL Video has a resolution of $720 \times 576$ at 25 frames per second (fps), or $45 \times 36 = 1620$ macroblocks at 25 fps gives a minimum throughput of 40500 macroblocks per second for the IPU.

We use a photograph with a resolution of $1536 \times 1152$, consisting of 6912 macroblocks. Three blocks in the JPEG decoder use the majority of processing power, VLD, IDCT and CCV, as can be seen from the relative execution time (relative to the total) shown in Table 5.1.

In Table 5.2 we presents the results of the different mappings which are described below. The two blocks where the most gain could be achieved, were the Colour Conversion and the IDCT. For our measurements we used for these blocks the original implementation, a highly optimized implementation and a null implementation (do nothing). We’ve measured the different possible combinations locate the best possible implementation. Because we did not even get close to the hardware-accelerated IPU decoder, we’ve used the null implementations (configuration 5 and 6)to find lower bounds on execution time.
Variable Length Decoding

In the VLD step codes varying in length from 1 to 16 bits are decoded to 8 bit values. Because most (> 95%) codes are less than 9 bits in size, they are looked up in a table by using 8 bits of the bitstream as the index. If the code is longer than 8 bits, the next 8 bits are used for a lookup in a second table. These tables contain (value, length) pairs, meaning the decoded value and the length of the code used. The bitstream is then forwarded the number of bits specified by the length, which involves bitshifting the current codeword and appending an also shifted part of the bitstream to it. For almost (except when the encoded value is exactly 8 bits long) every decoded value involves a lot of sub-byte level aligned operations. Only the MIPS can perform these operations reasonably well. The VLD inside the Image Processing Unit can unfortunately not be used for generic Huffman decoding, because the codetables cannot be changed. They are fixed in the MPEG-2 standard and the IPU strictly follows the standard strictly here, simplifying the hardware design.

Colour conversion

The CCV colour conversion is a more likely candidate for a different implementation. For every pixel the three colour components (R, G, B) are computed from the luminance (Y) and colour difference (Cr and Cb) components, using the following formulas:

\[
R = Y + 1.40200 \times C_r \\
G = Y - 0.34414 \times C_b - 0.71414 \times C_r \\
B = Y + 1.77200 \times C_b
\]

(5.4)  
(5.5)  
(5.6)

For every pixel we need 4 multiplications and 4 additions/subtractions. Because both the input as the output are integers, the multimedia SIMD instructions are the perfect candidate for implementing the colour conversion, enabling the processing of 8 consecutive pixels at a time. This results in a speedup by a factor of 4 for the colour conversion, and an increase in the total decoding by a factor of 1.3.

When using a null conversion, which simply copies the components into the output buffer, the execution time actually gets worse. This is being caused by stalling load/store operations and 'dirty' interleaving of data at offsets of 3 (every red component is separated by green and blue component and so on). In the standard conversion routine the RGB values are written in one pass (without jumping through the array). The optimized version uses multimedia instructions for the interleaving and writes 8 pixels at a time as 3 double words.

Inverse DCT

The IJG gives a choice between three implementations for the IDCT: one floating point version and two integer versions (a normal and a fast). The normal integer version is
as specified by the standard, which performs equally fast as the floating point version since the instructions have the same throughput on the MIPS. The fast version sacrifices some accuracy for speed. We choose to move the floating point version to a vector unit.

The two dimensional IDCT can be separated into two one dimensional IDCT's, first on each column and then on each row. The vertical IDCT can easily be transformed to a vector unit. The 128 bit registers which can hold four 32 bit floats enable the processing 4 floats in parallel instead of 1. A speedup by a factor of four is gained almost for free. The horizontal IDCT is a completely different story however. The Vector Unit instruction set has no support for data reorganization within a register, e.g. exchanging the middle two floats inside a register \((ABCD \rightarrow ACBD)\). The only way to change a position of a float is using so called broadcast instructions. These broadcast instructions can assign the single float result of an operation to any 4 possible locations inside a register. Because the butterfly operations inside the IDCT algorithm require two position changes each, almost all possible speedup through parallelism in the implementation is lost. Using the Vector Unit we can go from 16 1D IDCTs to 2 (four parallel) vertical and 8 horizontal 1D IDCTs. The expected speedup is a factor of \(16/(8+2) = 1.6\) using one Vector Unit. When distributing the IDCT over both units, the speedup is dictated by the slowest of the two, being the horizontal IDCT resulting in a factor of \(16/8 = 2\).

The maximum gain we can expect is not performing an IDCT at all. When doing so, we achieve a throughput of 15710 macroblocks, or 0.44 seconds for decoding the image.

Using both Vector Units for the IDCT leads to an estimated execution time of \(0.44 + (0.76 - 0.44)/2 = 0.60\) seconds, or a throughput of 11529 macroblocks per second. A performance increase by a factor of 1.65 compared to the original C implementation, but this still leaves us a factor 3.5 slower than the IPU.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>IDCT</th>
<th>CCV</th>
<th>Execution time (s)</th>
<th>MB/s</th>
<th>compared to IPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>normal</td>
<td>normal</td>
<td>0.97</td>
<td>7126</td>
<td>0.176</td>
</tr>
<tr>
<td>2</td>
<td>fast</td>
<td>normal</td>
<td>0.90</td>
<td>7680</td>
<td>0.190</td>
</tr>
<tr>
<td>3</td>
<td>normal</td>
<td>mmi</td>
<td>0.76</td>
<td>9095</td>
<td>0.225</td>
</tr>
<tr>
<td>4</td>
<td>fast</td>
<td>mmi</td>
<td>0.67</td>
<td>10316</td>
<td>0.255</td>
</tr>
<tr>
<td>5</td>
<td>normal</td>
<td>null</td>
<td>1.10</td>
<td>6284</td>
<td>0.155</td>
</tr>
<tr>
<td>6</td>
<td>none</td>
<td>mmi</td>
<td>0.44</td>
<td>15710</td>
<td>0.388</td>
</tr>
<tr>
<td>7 (estimated)</td>
<td>VU0, VU1</td>
<td>mmi</td>
<td>0.60</td>
<td>11530</td>
<td>0.285</td>
</tr>
</tbody>
</table>

Table 5.2: Experimental results of different JPEG mappings.

5.2.3 Conclusions

Clear bottleneck is the Huffman decoding, which is quite complex for a processor due to the many conditional branches and sub-byte level aligned operations, while being quite easy to implement efficiently in hardware. While the Vector Units excel at performing
the same operation on multiple data (as being a true SIMD processor), the lack of field exchanging operations as found in the Multimedia Extensions in the MIPS reduces there performance to a normal FPU in such cases.

5.2.4 3D Perspective Matrix Transformation

The transformation requires 1 division, \(7 + 3 = 10\) multiplications (there are 7 coefficients in the matrix, and 3 are needed for normalizing the x, y and z values of the calculated result) and 3 additions. When combining multiple transformation matrices, there number of zeros in the matrix decreases, so in the worst case there need to be performed \(4 \times 4 + 3 = 19\) multiplications and \(4 \times 3 = 12\) additions and 1 division. The additions can be done using the multiply-add operation, so we can leave the additions out in both cases. Since the Vector Unit can perform four multiplications and additions in parallel, we expect a performance increase by a factor of four.

On the FPU we implement both the minimal implementation, which computes exactly the calculations in Equations 5.2 and 5.3. The full implementations compute a complete matrix multiplication, not exploiting the presence of zeros in the matrix. We count the number of cycles used in the innerloop, discarding initialization such as the loading of \(M_{\text{perspective}}\) into registers, since those values do not change during the rendering of a frame.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>#innerloop cycles</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPU Full</td>
<td>29</td>
<td>1.0</td>
</tr>
<tr>
<td>FPU Minimal</td>
<td>20</td>
<td>1.5</td>
</tr>
<tr>
<td>VU Full</td>
<td>8</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Table 5.3: Innerloop cyclecount of 3D Perspective Matrix Transformation

As we can see in Table 5.3, the full implementation on the Vector Unit is more than twice as fast than the Minimal FPU implementation. When compared to the full implementation we gain a speedup by a factor of 3.6! When doing typical 3D matrix manipulation it is in fact possible to achieve a speedup of almost 4 on the Vector Unit when comparing to the MIPS.

5.3 Conclusion

When mapping a program on a specific architecture, an important factor in the performance one can achieve depends on how well the application 'fits' on the architecture. The implementation of a 3D matrix transformation on the PlayStation 2 which targets specifically at 3D applications gives almost a factor 4 speed improvement when using vector coprocessors.
The mapping of the more generic JPEG decoding application, which still is close to the domain of gaming and multimedia applications, gives a slightly less astonishing speed improvement of 1.65.

Because the bus provides a very high bandwidth low latency, data transfers and load/store operations did not introduce any problems. An extra benefit is the excellent cache controller. Because both test applications require only a small amount of local data and operate on larger consecutive amounts of data, the cache controller does a remarkable job. Because the cache was big enough, we did not benefit from using the scratchpad ram.
Chapter 6

PS2b: a better PS2

While the PlayStation 2 is a very successful platform with over 70 million units sold and still is the leading console even after 6 years of its introductions, it is not perfect. In this chapter, we present a better version of the PlayStation 2, or how the PlayStation 2 really should have been. We will discuss both improvements in hardware (Section 6.1) and in software (Section 6.2). Based on these findings, we present our conclusions in Section 6.3.

6.1 Hardware

In our experiments, we came across some irritating (at least for us) limitations of the architecture.

1. The Vector Units could be used for many more functions than just vector processing when having some operations added. Most notably the ability to reorganize data within registers. The multimedia instructions in the MIPS do support these operations, and are indeed very useful. They are quite common in most current architectures, such as in the PowerPC Altivec and the Intel SSE.

2. The Image Processing Unit is very fast at what it is designed for, namely decoding MPEG-2 streams. It is however also very inflexible. The VLD Huffman decoder would be very useful if the decoding tables could be defined in software.

3. While adding hardware accelerators, a cryptographic accelerator would come in handy. Both for secure internet transactions, as online gaming is expected to finally become big, and Digital Rights Management (DRM) protection for content\footnote{Some will argue this would be a reason to leave it out}.

4. Typical vertex processing operations on the Vector Units can produce 15 to 20 million vertices per second. This is far below the limit of 75 million the Graphics Synthesizer can render, unless all the nice features such as shading, fogging and...
anti-aliasing are turned on simultaneously. Besides, we want to use the VUs also for other purposes, such as Artificial Intelligence (AI) processing and simulating physics. AI algorithms such as reinforcement learning are nothing more than large vector multiplications.

5. A larger memory inside a VU would be nice for this, but not essential. Because the memory inside the VU can be accessed by both the the VU and the DMAC at the same time, one can set up the memory for double-buffer processing. That is, the VU operates on one part of the memory and the DMA reads/writes the other part. After the VU has finished processing both memory areas can be swapped. However when enlarging the memory, more registers are more than welcome. This way complex operations can be combined in larger loops, without the need of loading the same coefficients every iteration again because of register starvation.

6. Even more important would be equal memory sizes for the vector units. Currently, every program that can be executed on VU0 can also be executed on VU1, the other way around is not always true. Not only does VU0 lack some instructions present in VU1 (although they are not that often required), more important is the difference in memory size by a factor of four. Once a decision has been made to locate a program on VU1, it is often impossible to move it to VU0.

7. When generating complex (i.e. high polygon count) scenes, all processing capabilities of the vector units are required to create the scene. As a result little AI and physics processing can be done. As a solution to this situation more main memory could be installed, so behaviour or scenes can be precalculated and loaded from memory instead of being regenerated every time. A much more flexible solution is of course to install more vector units. When using some of the options like shading and anti-aliasing in the graphics synthesizer, two vector units will generally be sufficient to generate complex scenes at full framerate (i.e. 50fps for a PAL version). On the other hand, one vector unit can be used for so capable of simulating relatively complex and realistic behaviour through AI processing, while the other VU simulates object movements and behaviour through physics processing and collision detection. Instead of having to choose between both situations, it would generally be nice to do all these operations in parallel and to be forced to make a trade-off. An additional two vector units (so a total of four) will be sufficient for both realistic intelligence, object movements and pushing the graphics more to the limit and actually achieving full framerate. Many games run only at half the framerate, so 25 fps for a PAL version. This results in visual artifacts such as motion judder and less smooth animations [2].

The resulting figures for parallelism achieved in the PS2b, are presented in Table 6.1. When adding up the numbers to calculate the total, again the FPU line is left out because a maximum of two instructions can be issued every cycle when using the FPU and MIPS together. The total level of parallelism is 88, which is 1.57 times higher than the parallelism of the original PS2, which is 56.
6.3 Conclusion

The improvements we propose for the hardware can be divided into two categories:

- Create a more flexible architecture that can be used in more application domains.
- Add more processing elements and more memory.
The first category requires relatively small modifications (e.g., addition of instructions to the processing elements), but is only interesting if the architecture is to be deployed in different application domains than game processing. The second category not only increases the processing power and possibilities of the architecture, but also increase the costs a lot (mainly due to the chip size). Now smaller production techniques are available, these additions (such as extra vector units) are feasible.

The main point about software is that there are no decent tools available to use the architecture to the max. While maybe not available at the time of introduction, more sophisticated development tools (such as a compiler for the vector units) should have been made available.
Chapter 7

PS3: overview of the Cell architecture

The successor of the PlayStation 2, labelled PlayStation 3, has a big hype around it's new Cell architecture. Most details about the Cell in this chapter are taken from the overview presented in [7]. We will first give an overview of the Cell architecture in Section 7.1. We will compare the Cell architecture in Section 7.2 with the improved PS2b we proposed in Chapter 6.

7.1 The Cell Processor

At the heart of the PlayStation 3 is a Cell processor, a cooperative design by IBM, Sony and Toshiba. Besides the new PlayStation, the chip will also be used in upcoming Toshiba television sets. The Cell will operate at a clockspeed of 4GHz, but can be used at 3GHz using a lower voltage, which would reduce the power consumption considerably. The PlayStation 3 is expected to run at the lower 3GHz clockspeed. Figure 7.1 shows a block diagram of the Cell architecture.

The Cell consists of three main units, supported by two Rambus interfaces. There is a single Power PC core that acts as the main host processor, eight SIMD processors (SPU), and a highly programmable DMA controller (MFC).

7.1.1 Synergistic Processor Unit

The Cell processor contains eight Synergistic Processor Units (SPU). The SPU is the successor of the VU in the PlayStation 2. Each SPU is a four-way SIMD unit optimized for single-precision floating point. It contains 128 128-bit wide registers, which reduces the need for load store operations. The instruction set is a superset of the PlayStation 2 Vector Unit instruction set including multiply-add, with a flavour of Altivec instructions added (as found in other PowerPC CPU’s, such as the Power G4). At a clockspeed of 4GHz, the eight SPUs will be capable of a peak performance of 256MFlops.

The Local Storage (LS) SRAM memory has a size of 256kB, which is used for both instructions and data. The LS memory is mapped in the global memory space,
however it is not cache coherent with respect to modifications made by the SPU to which it belongs.

7.1.2 Memory Flow Controller

The Memory Flow Controller (MFC) is a highly programmable DMA controller. The MFC is responsible for transferring data between main memory, the SPU local storages. For time-critical operations the MFC has the ability to transfer data from an SPU local store directly into the L2 cache of the PowerPC. To keep the Cell processor running and the SPU utilization high, the MFC supports 12 simultaneous transaction flows. When transferring data to an SPU, the MFC can instruct the SPU to begin processing instructions after the transfer has finished.

The MFC supports DMA-list command schemes. The list of DMA commands is placed in the local store and then processed asynchronously by the MFC.

7.1.3 Memory and I/O

The memory and I/O interfaces are based on Rambus technology. The I/O Controller (IOC), consists actually of two independent controllers for flexibility. The memory interface can support a bandwidth of 25.6GB/s. Two Cells can be connected using the RRAC connection.
CHAPTER 7. PS3: OVERVIEW OF THE CELL ARCHITECTURE

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>O</th>
<th>D</th>
<th>S</th>
<th>Mpar</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerPC</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>&gt;1</td>
<td>8</td>
</tr>
<tr>
<td>SPU</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>&gt;1</td>
<td>16</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>18</td>
<td>17</td>
<td>36</td>
<td>&gt;1</td>
<td>136</td>
</tr>
</tbody>
</table>

Table 7.1: Parallelism figures for the Cell processor

7.1.4 Element Interconnect Bus

All the elements are connected to each other via the Element Interconnect Bus (EIB). The EIB is actually not a bus, but 4 unidirectional data-ring structures, 2 running clockwise, the other two counter-clockwise. This reduces the worst-case transport latency to half the length of the ring. Every ring supports up to 3 simultaneous transactions, leading to a maximum of 12 transactions of 8 bytes per cycle, or a total transfer rate of 96 bytes per cycle. Operating at a clockspeed of 4GHz, this equals a bandwidth of 38.4GB/s.

7.1.5 Parallelism

In Table 7.1 the parallelism values for the Cell are presented (not the complete PS3, for which we did not have any figures available). However, even when only comparing a single Cell to a complete PS2 or PS2b gives interesting result. The high parallelism level of the Cell of 136 is a factor of 1.55 above the PS2b, and even 2.43 times as high as the PS2. This means that the Cell can perform almost 2.5 times as many operations as the PS2 at the same clockspeed.

7.2 Comparison between the Cell and the PS2b

The Cell is a big improvement over the Emotion Engine, while having a familiar design. In the following sections we will compare the Cell architecture with our proposed improved PlayStation 2.

7.2.1 Synergistic Processor Units vs. Vector Units

The SPU, the Cell equivalent for the VU supports a superset of the VU instruction set. The added Altivec inspired instructions suggest that there will be support for general instructions such as data reorganization, like we propose for the PS2b.

The SPU has a local store of 256kB, which is used for both instructions and data. Smaller programs leave more space for data. The number of registers has been doubled twice. There are 128 128bit registers in every SPU.
While we suggest to have 4 Vector Units in the PS2b, there are 8 SPUs in the Cell. There are two possible reasons for this. With a more powerful graphics processor between the Cell and the screen, it is likely to be beneficial to have more SPUs. Besides that, the SPUs can be regarded as flexible (fully programmable) accelerators, as we discuss in Sections 7.2.2, 7.2.3.

7.2.2 Image Processing Unit
While being practical for decoding MPEG-2 streams but inflexible for almost everything else, the IPU is not found in the Cell. Maybe parts of it will reappear inside the graphics processor (in PC's it's quite common to have IDCT and colour conversion been done by the video processor), but on the other hand MPEG-2 decoding is no challenge at all for the PowerPC.

7.2.3 Crypto accelerator
The generic Huffman accelerator we proposed didn't find it's way into the Cell, but the cryptographic accelerator did, sort of. SPUs can be fenced of from each other through hardware protection features. A separated SPU can the safely be used for security processing operations.

7.2.4 Programming
The programming toolchain for the Cell is built on PowerPC Linux. There is also a C(++) compiler for the SPU. This will definitely ease the process of moving functions away from the PowerPC to one of the SPUs. If there are good support libraries available is not clear yet, but the availability of a C compiler for the SPUs is a positive sign.

7.3 Conclusion
The Cell/PS3 has very impressive specifications, and most limitations we encountered in the PS2 design have been removed in the PS3. Adding a high clockspeed (estimated 4GHz), this will indeed be a very powerful platform. How well it will perform in the field will depend a lot on the quality of the supplied development tools, as this platform takes complexities to yet another level where manual design becomes less and less viable.
Chapter 8

Conclusions and future work

8.1 Conclusions

The results presented in Chapter 5 show that we can achieve a higher decoding speed for JPEG images. By using other instructions and using more processing units we've increased the processing speed by a factor of 1.6. This is however, still 3.5 times slower than the MPEG-2 decoding speed delivered by the Image Processing Unit. Main reasons are the Huffman decoder that cannot be moved away from the MIPS and the lack of data reorganization functions in the Vector Units.

The mapping of a 3D Perspective Transform, gives a speed improvement of 3.6, which is close to our expected improvement by a factor of 4. Clearly, processing vector and matrix multiplications at high speed is one of the key performance points of the PlayStation 2.

Not only are most of the improvements we suggest in Chapter 6 are actually present in the Cell processor, that will be used inside the PlayStation 3. The Cell processor goes further than the PS2b with even more processing elements and an expected decent toolchain.

8.2 Future work

While not all details of the Cell are yet revealed, the most interesting research topic is the development of a programming paradigm to program the different units easy, even when connecting more Cells together. Use of a library such as Yapi as being used for Spacecake comes to mind.
Appendix A

Glossary

AI  Artificial Intelligence.
DRM  Digital Rights Management.
DTSE  Data Transfer and Storage Exploration.
EE  Emotion Engine. CPU of the PlayStation 2.
IJG  Independent JPEG Group.
IPU  EE image processor unit.
SIMD  Single Instruction Multiple Data. Perform the same operation on a multiple of small operands which are packed together into one register.
SPR  ScratchPad Ram. Quick-access data memory built into the EE Core.
VPU  Vector processing unit. EE contains 2 VPUs: VPU0 and VPU1.
VU  VPU core operation unit.
VIF  VPU data decompression unit.
Bibliography


