Interactive Ray Tracing Using a SIMD Reconfigurable Architecture

Manuel L. Anido[1], Nozar Tabrizi[3] , Haitao Du[3] , Marcos Sanchez-Elez M.[2], Nader Bagherzadeh[3]

[1]Federal University of Rio de Janeiro, NCE
mlois@nce.ufrj.br
[2]Universidad Complutense de Madrid
marcos@fis.ucm.es
[3]University of California, Irvine, CA 92697
nader@ece.uci.edu

Abstract

This paper presents an architecture for running interactive ray tracing applications on portable devices such as cell phones, PDAs, and head mounted displays and discusses the main issues related to the mapping of this graphics algorithm using fixed-point arithmetic. The paper shows that a floating-point arithmetic unit, with its associated power and area consumption, can be avoided by using appropriate fixed-point arithmetic and block floating-point operations. It is also shown that a computation intensive graphics method like ray tracing can be used to generate simple images at interactive rates on portable devices. This can be achieved by employing a reconfigurable SIMD architecture on a chip, which trades parallelism for frequency of operation, thus providing significant benefits in power saving, which is essential in portable devices.

1. Introduction

New phones for next-generation networks and other types of handheld devices such as Personal Digital Assistants (PDAs) will soon be able to download (or even upload) games, video, audio and many other types of applications. Some companies have already started offering camera-equipped cell phones to snap and send photos wirelessly[1]. This capability will open a new range of applications in many areas, such as in education, entertainment, news, security, where support for multimedia is highly desirable or mandatory. In particular, e-commerce applications would benefit from a significant higher level of realism than it is possible to achieve with the current graphics hardware.

Although the games available in cell phones are still far simpler than what is currently available in computer games or in recent game products such as Nintendo Game Cube and Microsoft's X-Box, they are more detailed and responsive than previous versions. Besides games, companies are starting to add more entertaining features into the basic functions of phones, including its sounds, its screens and the messages it sends[2].

Dr. M. Anido acknowledges the financial support by CNPq. This work was supported by DARPA (DoD) under contract F-33615-97-C-1126 and the National Science Foundation under grant CCR-0083080.

To cater for the computation demand that CGI and multimedia applications require, some recent microprocessors have incorporated SIMD instructions in their execution units and it has been shown that significant improvements can be achieved. This has been driven by the need to accelerate multimedia and DSP applications because these applications have become increasingly important for computer systems as well as for wireless-based devices as a dominant computing workload.

The "MultiMedia eXtentions"(MMX) and Streaming-SIMD Extensions(SSE) from Intel [4], the 3Dnow! Extension from AMD[5], and AltiVec technology from Motorola [6] are examples of SIMD signal processing instruction set extensions on general-purpose processors.

There are also special-purpose multimedia processors (ASICs and DSPs) such as the Trimedia VLIW processor from Philips[7] and nVidia's nForce/geForce GPUs used in Microsoft's Xbox [8]. These processors usually have hardware add-on units in the form of peripherals for one or more of the multimedia decoding functions. A third category of architectures also used for DSP and multimedia applications are reconfigurable SIMD systems-on-chip, which can be advantageous in many applications, when compared with general purpose processors, ASICs or DSPs. Examples of such chips are MorphoTech's MorphoSys SIMD system-on-chip[9] ClearSpeed's FUZION architecture[10]. The market for such special-purpose multimedia processors is primarily in low-cost low-power embedded applications such as set-top boxes, cell phones, PDA devices, wearable devices, and standalone entertainment devices like DVD players.

2. Ray Tracing and SIMD architectures

2a. Ray Tracing

The main emphasis of this paper is on the architecture of a SIMD system-on-chip for a ray tracing algorithm, although the SIMD architecture to be described also provides proper support for the rasterization approach. However, this will not be addressed in this paper. The possibility of having an

IEEE COMPUTER SOCIETY

architecture for both graphics approaches is feasible because the SIMD system-on-chip to be described is programmable and reconfigurable and it can easily adapt to different types of irregular data parallel applications, as will be discussed in section 2b.

The term ray tracing is commonly associated with highly realistic images but certainly not with interactive or real-time graphics. However, with the increasing power of hardware resources, interactive and real-time ray tracing is becoming a reality and offers a number of benefits over the traditional rasterization pipeline. Today, interactive rendering is almost exclusively the domain of rasterization-based algorithms that have been put into highly integrated and optimized graphics chips. Their performance has improved dramatically over the last few years and now it even exceeds that of what used to be graphics supercomputers. Moreover, this hardware can be purchased at such low cost that most of today's PCs already come equipped with state-of-the-art 3D graphics devices[8]. However, most of these devices are non-programmable application-specific circuits that operate at very high frequencies and consume a significant amount of power, which is not compatible with the low power requirements of portable devices[8].

Ray tracing projects light rays from the viewer through each pixel in an image using the constraints of a camera model, which consists of the position of the object, position of the camera, viewing angle of the camera and viewer's position[11,12]. The ray tracer sets the pixel to the color of the object that the ray hits, also taking into consideration the contributions of reflections, shadows, highlights, and various surface properties of the object. This mechanism is illustrated by figure 1.

Ray tracing is known not only for its ability to generate high-quality images but also for its long rendering times due to its high computational cost. This happens because of the need to traverse a scene with many rays, intersecting each ray with the geometric objects (which are usually represented by three-dimensional parametric equations), shading the visible surface samples, and finally sending the resulting pixels to the screen. Nonetheless, ray tracing offers a number of benefits over rasterization-based algorithms[11,12,13], such as Flexibility; Occlusion Culling and Logarithmic Complexity; Efficient Shading; Simpler Shader Programming; Correctness by Default; Parallel Scalability and Coherence.

It is due to the advantages listed above that ray tracing is an interesting alternative even in the field of interactive or real-time 3D graphics. The challenge is to improve the speed of ray tracing to the extent that it can compete with rasterization-based algorithms. It is worth noting that the use of ray tracing algorithms goes well beyond graphics applications such as rendering. Ray casting is a fundamental task that is at the core of a large number of algorithms from other fields.

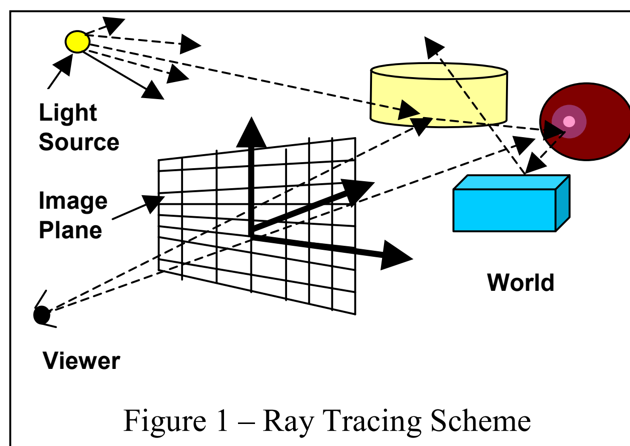

Figure 1 – Ray Tracing Scheme

2b. SIMD Architectures for Ray Tracing and Rasterization

The field of computer graphics has seen both architectures of varying programmability and architectures that use custom special-purpose hardware. For many years, designers have taken advantage of the inherent parallelism in ray tracing and rasterization approaches. Due to their scalability and programming model, several SIMD architectures were developed for computer graphics. Pixar's Chap[16] was one of earliest architectures to explore a programmable SIMD computational organization, on 16-bit integer data. There have also been recent SIMD architectures and chips for ray casting and volume rendering such as PAVLOV[17] and FUZION [10]. The first ray tracing chip was developed by Advanced Rendering Technology in 1997 – the AR250[14]. Recently, the company has announced the AR350. These chips employ expensive hardware intersection pipeline (e.g. ray-triangle intersection) and 32 32-bit floating point units. Therefore, there has been a trend to integrate systems on a chip and SIMD architectures are a natural candidate for such integration because of their scalability and regularity.

The general SIMD computing model lets one instruction operate at the same time on multiple data items. Operations that normally require a repeated succession of instructions can be performed in one instruction. For that purpose, SIMD architectures incorporate an array of processing elements that are centrally controlled by a general-purpose microprocessor called the *master processor (or array control unit)*. Thus, it is possible to have only one copy of the program and a master processor that fetches the instructions and broadcasts them to all of the processing elements. The master processor is also responsible for performing *global branches* that affect the entire array of PEs[18].

MorphoSys-I [9,19,20,21] is a system-on-chip which consists of a reconfigurable ASIMD-like array of reconfigurable cells (64 cells or RCs) on the same chip with a general-purpose RISC processor and a high performance streaming memory interface. Its architecture is illustrated by Figure 2. Each reconfigurable cell (RC) is an individual 16-bit processing

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)
0-7695-1772-2/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

element which also includes some 32-bit operations. Besides allowing reconfigurable connections between RCs (PEs), MorphoSys-I has the characteristic of being able to execute different non-communicating instruction streams in different columns (or rows) under a central control. Such a characteristic can be used to compute irregular data-parallel algorithms such as ray tracing, thus facilitating the emulation of MIMD behavior. Mapping several algorithms on the MorphoSys-I architecture showed that very high performance can be achieved, when compared with other approaches regarding various algorithms like MPEG encoding, automatic target recognition, ciphering using the IDEA algorithm and several DSP algorithms[9,19,20,21].
This paper describes an on-going work about using a new version of the MorphoSys chip for graphics applications (MorphoSys-II-G), which includes novel features to support irregular data parallel computations[DSD2002], fast access to individual data within the internal RAM of each reconfigurable cell (RC or processing element) and new instructions to support for fixed-point arithmetic with scaling. Some of these features will be described in the next sections.

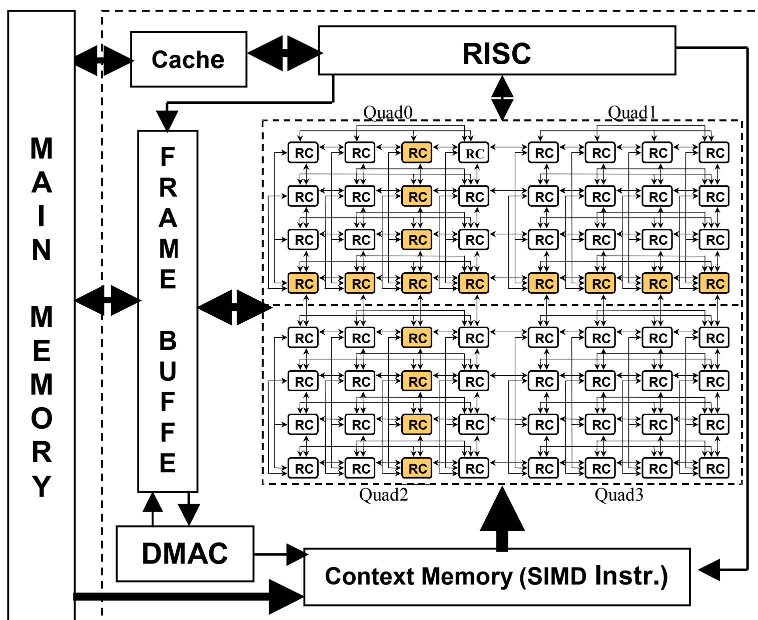

Figure 2 – MorphoSys I Architecture

MorphoSys operation is as follows: The host computer loads the sequential program code for the RISC control unit from main memory. The RISC control unit programs the internal DMA controller to transfer contexts (SIMD instructions) from main memory to the Context Memory and also transfer data from main memory into the Frame Buffer. Then, the RISC controller executes ordinary sequential RISC instructions or coordinates the execution of SIMD instructions in the RC array. The execution of one SIMD instruction means executing 64 simultaneous contexts (or RC instructions). The context memory stores the contexts that the SIMD array has to execute. The context memory stores up to 8 different contexts (instructions for each RC) in each context memory plane. Henceforth, up to eight different contexts can be executed simultaneously on rows or columns, which implies that there can be different non-communicating instruction streams being executed simultaneously. One of the key aspects of the architecture is that the Frame Buffer employs two banks, allowing DMA data transfer concurrently with processing, providing excellent support for the processing of instruction streams.

3. Arithmetic Operations

Floating point arithmetic units are normally used in general-purpose microprocessors to support fast arithmetic operations for floating point data types. However, these units demand a considerable silicon area and consume significant power. These characteristics are incompatible with the low power and low area requirements of portable devices. Most DSP chips do not use floating-point units because of these restrictions.

Fixed point arithmetic is a type of arithmetic that uses the integer unit of the processor, where operands may have an integer part and a fractional part. When two operands are added or subtracted, a carry (borrow) bit propagates normally between the fractional part and the integer part as if the two operands were just integers[23]. Fixed point arithmetic with scaling, together with *block floating point* arithmetic [24] can provide similar image quality to floating point arithmetic, at an equivalent speed for basic functions such as addition, subtraction and multiplication. Figure 3 illustrates a 16-bit fixed point multiplication with scaling that is implemented in one clock cycle in each of the 64 MorphoSys SIMD processing elements. Floating point arithmetic may be necessary for graphics applications that employ a large database where the world coordinates may require a large dynamic data range. However, this is not the case for the targeted applications envisaged for the architecture described in this paper.

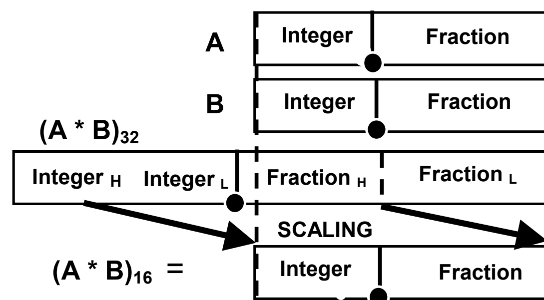

Figure 3 – Fixed Point Multiplication with Scaling

We use *block floating point* arithmetic [24] in sections of the code where intermediate computations may generate results that do not fit into the processor word and also when it is

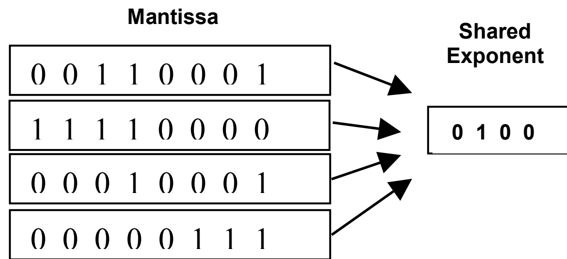

Figure 4 – Block Floating Point Operation

necessary to minimize truncation errors during some operations. *Block floating point* provides some of the benefits of floating point representation, and it works by scaling blocks of numbers rather than each individual number. This approach was used to make the most of the mantissa of the operands and also to minimize loss of significant bits during arithmetic operations with scaling (truncation). As illustrated by figure 4, if a number in a block of numbers becomes too large for the available word length, the programmer scales down all the numbers in the block, by shifting them to the right (similar for small numbers).

To calculate the intersection of a ray with some common types of objects like spheres, cylinders and ellipsoids, two complex functions are usually required: division and square root. The implementation of these functions require careful analysis because of the number of cycles required and also because of the possible loss of significant bits during intermediate computations, which can affect image accuracy and quality.

We implement a signed 32-bit by 16-bit fixed point division by multiplying the numerator by the reciprocate of the divisor, that is, Q = N*1/D, where N and D can be considered integers or fractions represented using n-bit numbers in the 2's complement form. The method for computing 1/D is based on the Newton-Raphson iteration, which leads to the recurrence[22,23]:

$$x^{(i+1)} = x^{(i)} * (2 - x^{(i)}D).$$

If D is normalized to the interval [1/2,1), and choosing $x^{(0)} = 1.5$, it limits the initial error to a maximum of 0.5, guaranteeing quadratic convergence. Because 1/D is linear for $D \in [1/2, 1)$, we do not use a lookup table for the first approximation $x^{(0)}$ as indicated in most computer arithmetic books. Instead, we take the one's complement of the four most significant bits of D as the first approximation, as illustrated by figure 5. The number of iterations of the algorithm to accomplish the 32-bit by 16-bit division is two(2) and the total number of instructions (clocks in this case) is 30. The RC array of MorphoSys has 64 processing elements, which means that up to 64 divisions can be computed simultaneously on different data.

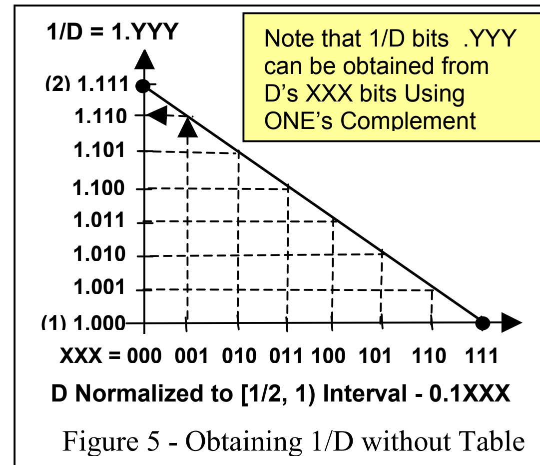

Figure 5 - Obtaining 1/D without Table

Like division, square-rooting can be formulated as a sequence of shift and subtract operations and several methods are described in the literature [22,23]. Common binary restoring shift/subtract algorithms and binary nonrestoring algorithms usually require between 70 and 100 instructions to extract the square root of a 32-bit number. Additionally, the implementation of these algorithms in DSPs or general-purpose microprocessors makes use of branch instructions to speedup the algorithm by branching code sections that are not needed depending on certain data values. However, the processing elements of SIMD machines do not have a program counter that supports the implementation of branches. Therefore, the implementation of the square root algorithm in the RC unit (PE) of MorphoSys requires the use of a convergence method that always executes the same number of iterations, irrespective of the data value. We use the Newton-Raphson's convergence method as described in [22,23].

To use the Newton-Raphson method to compute $\sqrt{z}$, the function $f(x) = x^2 - z$, with a root at $x = \sqrt{z}$ is usually chosen. Thus, the function $f(x) = x^2 - z$ leads to the following convergence scheme for square rooting:

$$x^{(i+1)} = 0.5(x^{(i)} + z/x^{(i)})$$

Unfortunately, this function employs a division operation in each iteration and the division itself requires 30 cycles for completion. A variant of the approach, which has been used in high-performance processors[Cray], is based on computing the reciprocal of $\sqrt{z}$ and then multiplying the result by z to obtain $\sqrt{z}$. The function $f(x) = 1/x^2 - z$ which has a root $x = 1/\sqrt{z}$ can be used for this purpose, resulting in the following recurrence:

$$x^{(i+1)} = 0.5\, x^{(i)} (3 - z(x^{(i)})^2),$$ which can be rewritten (to avoid subtraction from 3) as:

$$x^{(i+1)} = 1.5\, x^{(i)} - 0.5\, z(x^{(i)})^3$$

In the first iteration, the values $1.5x^{(0)}$ and $0.5(x^{(0)})^3$ can be read out from a table to reduce the number of operations. Thus, each iteration now requires five multiplications, two shifts and one subtraction. To keep the intermediate results within 16 bits of precision and also to perform 16 x 32 multiplications to avoid some loss of significance, a few additional shift and mult instructions are required. The mapping of the 32-bit square root function described requires 65 clock cycles (or RC instructions) and makes use of two small tables of 16 data each. It is possible to have 64 simultaneous square root operations, which means that, on average, one square root takes one clock cycle for completion in MorphoSys SIMD RC array.

4. Improving the Operation Autonomy of SIMD Processing Elements

Both the ray tracing algorithm and the rasterization method are irregular data parallel applications, which means that a single program or function that is applied to different data can follow different paths for different data. SIMD architectures do not normally support this type of parallel behavior because it implies that different processing elements could take totally different paths in the program flow (or would have total operation autonomy). Most SIMD architectures support a very limited form of operation autonomy which usually consists of executing, or not, an instruction. In previous papers [25,26] we presented new solutions to this problem, allowing the processing elements of SIMD architectures to execute programs with complex nested *if-then-else* constructs, which translate into conditional branches. These solutions were implemented and verified in the MorphoSys II-G simulator.

The PE(or RC) autonomy described in [25,26] is not total autonomy because there are no individual control units like MIMD machines. However, it allows the MorphoSys RC array to execute different non-communicating program streams in different columns, or rows. This facilitates the parallel mapping of several applications. For example, in ray tracing, instead of associating one ray (or pixel) to one processing element (or RC) it may be efficient to have all the PEs of one column or row calculating the intersection of some rays with one type of object, while another intersection calculation is done for a different object in a neighboring column or row. Ideally, all RCs should have the same contexts (program) because this is the situation in which the utilization of the RC array is the highest. However, this is hard to guarantee in most algorithm mappings. Therefore, different columns or rows may execute different instruction streams. If different instruction streams need to communicate, then the programmer or compiler has to take care of the data dependencies between these streams. MorphoSys architecture also offers support for 2D neighboring RCs to cooperate in the solution of a problem, that is, RC associations other than column or row are also supported[9,19,20,21].

In [26] it is shown that by combining guarded instructions[27] and pseudo branches it is possible to achieve higher operation autonomy and higher instruction level parallelism than in previous SIMD/ASIMD architectures. The paper showed that it is feasible to avoid most branches and it is also possible to emulate conditional execution on the processing elements, either by using guarded instructions or by using pseudo branches, thus avoiding unnecessary intervention by the array control unit in data-dependant computations. Pseudo branches are used when it is not possible to use guarded instructions. Additionally, they also support the implementation of complex nested if-then-else constructs, improving the execution of irregular data-parallel applications. It was also shown that pseudo branches can be used to control the power saving of those processing elements that have instructions nullified.

We employ a simplified form of predication for most of the SIMD instructions of the MorphoSys' architecture, which consists of *guarded execution* based on combinations of the ALU flags (Carry, Zero, Sign and Overflow). There were not enough bits in the instruction field to implement full predication, which would include a field to specify a predicate register. Instead, we decided to have more registers in the register file. Additionally, the implementation of a full predication scheme requires a considerable area, which may not be feasible for fine-grain SIMD/ASIMD architectures.

Although guarded instructions are essential to minimize the number of branches, they are not able to solve all possible situations. Additionally, there are instructions that do not have sufficient bits in the instruction word to specify the guard condition. In these cases, previous microprocessors have used *Guard* and *Wakeup* instructions to support *if-then-else* constructs. However, no previous scheme has supported *nested* if-then-else sentences. If PE's in a SIMD array had a program counter (allowing them to perform branches) there would be no problem. However, as it is not possible to have true branches, a new scheme to solve this problem had to be devised. We introduced *pseudo Branches* in SIMD machines, which are another form of a *Guard* instruction, but with the novel feature of adding a destination tag(label) field in the instruction to support *nested if-then-else* constructs.

The main reason to emulate real branches in PE's is to be able to conditionally (or unconditionally) nullify blocks of code starting at the "branch" instruction and ending at the *target* position, independently of the instructions inside that block of code. This is how real branch instructions work and by doing so complex nested *if-then-else* constructs can be emulated. In normal microprocessors, the *target* address is loaded into the program counter (PC), before branching. One possible solution to specify a target instruction in a stream of SIMD instructions is to include an extra tag(label) field in the instruction word. However, this increases the instruction word width, which is not feasible normally. The solutions of these problems are described in [26].

IEEE COMPUTER SOCIETY

5. MorphoSys II-G Reconfigurable Cell Architecture

The RC array is composed of 64 reconfigurable cells. Each cell can be connected to its neighbors as illustrated by figure 2 and is described in detail in [9,18]. The ALU in each reconfigurable cell of MorphoSys II-G is 32-bit wide and the communication with neighboring RCs is accomplished by using 16-bit buses. The major blocks of each reconfigurable cell are illustrated by figure 6 and are: an internal ALU, a 16x16-bit parallel multiplier, input multiplexers to select the input operands, 32-bit combinational shifter, sixteen 16-bit general-purpose registers which can be also addressed as register pairs forming eight 32-bit registers, and a 512x16-bit RAM.

The main objectives of the RC RAM are to store data that may be addressed differently by each RC (e.g. tables for sine, cosine, square root, image slices, etc) and to provide a mechanism to save temporary data when the number of registers is not enough. Another fundamental function of the internal RAM is that it provides support for a pipelined operation between
MorphoSys II-G cores, i.e., output data from one pipeline stage does not have to be moved back to the frame buffer. Instead, it can be transferred via DMA to the next element in the pipeline. Fast access to this internal RAM is supported by special instructions which use Base+Index with auto-increment addressing mode.

It is important to mention that constants or data whose address is specified by the RISC control unit program (when data address does not depend on computations within each RC or, in other words, it is not data-dependant) is stored in the frame buffer (see figure 2). It is more efficient to have this type of data centralized in the frame buffer because data transfers to RAM units within RCs can be avoided.

6. Instruction Format and Instruction Set

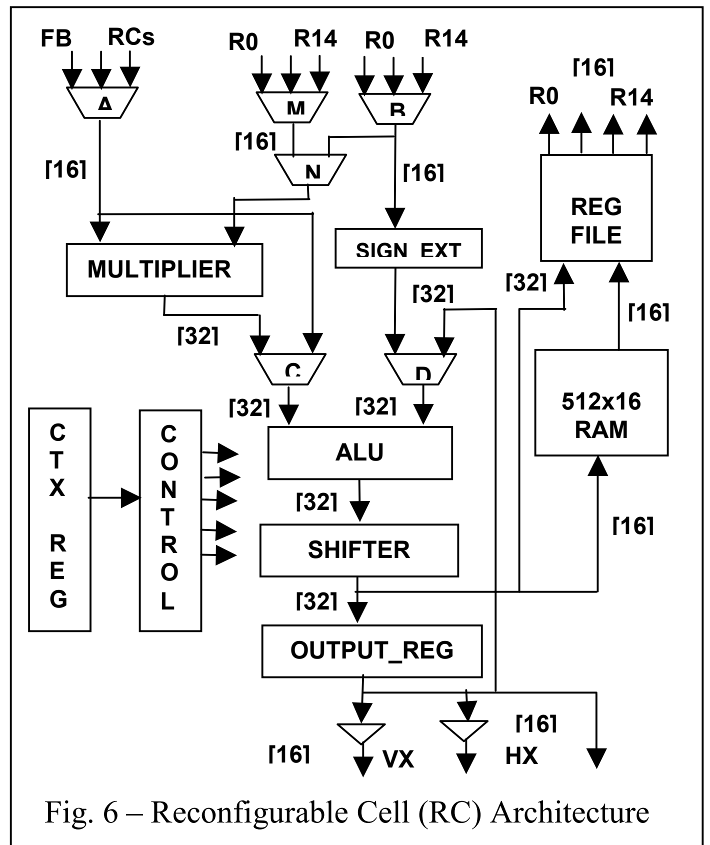

Fig. 6 – Reconfigurable Cell (RC) Architecture

Each RC unit in MorphoSys II-G architecture supports a small, yet powerful, set of contexts to support most operations required by CGI applications using fixed-point arithmetic. The context width is 32-bit wide and all contexts execute in a single clock cycle. Most operands are 16-bit wide but there are also 32-bit operations which usually make use of two concatenated 16-bit registers. The main purpose of having 32-bit operations is to support 32-bit operands in intermediate calculations.

MorphoSys II-G contexts differ from MorphoSys I [9] contexts because graphics applications have different requirements from DSP applications. To implement some of

Table 1 – Some MorphoSys SIMD Context (Instruction) Types - 32 bit Instruction Word

ALU Type	OPCODE	DST	Shift Dir	#SHIFTS	MUXA	MUXB	S/N	G. COND
MAC “	OPCODE	DST	Shift Dir	# SHIFTS	MUXA	MUXB	SRC_MAC	Rg32
LDIMM “	OPCODE	DST	xxxxxxxx	xxxxxxxx	CONSTANT			
MEMOP “	OPCODE	DST or SRC	xxxxxxxx	xxxxxxxx	xxxxxxx	xxxxxxxx	xxxxxxxx	(Reg)
SHIFT16	OPCODE	DST	Shift Dir	xxxxxxxx	MUXA	MUXB	Reg=#Shifts	G.COND
MISCEL	OPCODE	DST	Shift Dir	#SHIFTS	MUXA	MUXB	BASE	G.COND

Context 0	Context 1	Context 2	Context 3	Context 4	Context 5	Context 6	Context 7

Figure 7 - Context Memory (SIMD Program) – width = 8 x 32 bits = 256 bits

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD’02)
0-7695-1772-2/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

the new instructions, it was necessary to make changes in the internal architecture of the RC. Instruction formats are illustrated by figure7 and table 1.

Among the most powerful instructions are single-cycle MAC (Multiply-Accumulate) operations with scaling. MAC instructions multiply two 16-bit numbers and generate a 32-bit result, which can be shifted if necessary. After shifting, the result can be added with a previous 32-bit result in the accumulate unit. Other important instructions are: Multiple-bit Shift instructions; Arithmetic and Logic operations that can specify neighboring cells (in addition to normal registers) followed by shift operations; Load and Store operations to/from the internal RC's RAM using base+index mode with auto-increment; pseudo Branch[26] instruction and Count Leading Zeros instruction to support normalization operations required by arithmetic functions. A Multiple Load instruction is available and it can schedule the loading of up to 3 operands from RC's RAM, to be performed concurrently with ALU operations.

Most of the instructions are *guarded*, that is, they can be executed conditionally, depending on the evaluation of the guard condition. Guard conditions are obtained by testing the usual ALU flags (C,S,N,V) and they were included to support pseudo branches, providing each RC with more operation autonomy than previous SIMD architectures had provided[26].

7. Evaluation, Performance and Resource Estimates

7a. Evaluation

To demonstrate the feasibility of generating good quality images (using a ray tracing algorithm) without employing floating point arithmetic and also to prepare the source code to be mapped onto MorphoSys II-G, we programmed a ray tracing algorithm using only 16-bit fixed-point arithmetic with scaling. We made use of *block floating point* arithmetic when necessary, as described in section 3.

Figures 8 and 9 (300 x 300 x 32K color pixels) illustrate the results obtained from a ray tracing algorithm encoded in C, which is integrated to the simulator of the MorphoSys SIMD system-on-chip architecture. Several functions of the ray tracing algorithm were mapped onto the SIMD architecture and were used to generate these images. A different work in our research group also mapped the polygon rasterization approach onto MorphoSys II-G using fixed point arithmetic.

7b. Performance

A VHDL simulator of the RC architecture, synthesized using Synopsys tools and a 0.13μm cell library provided a pre-placement and routing clock cycle of 8 ns or 125 MHz clock cycle. After placement and routing the frequency of operation will be lower.

Aiming at assessing the performance of the mapping of a ray tracing algorithm onto the MorphoSys SIMD architecture for a simple game running on a cell phone, we evaluated the number of clock cycles required to perform each of the operations. The results are shown on Table 2.

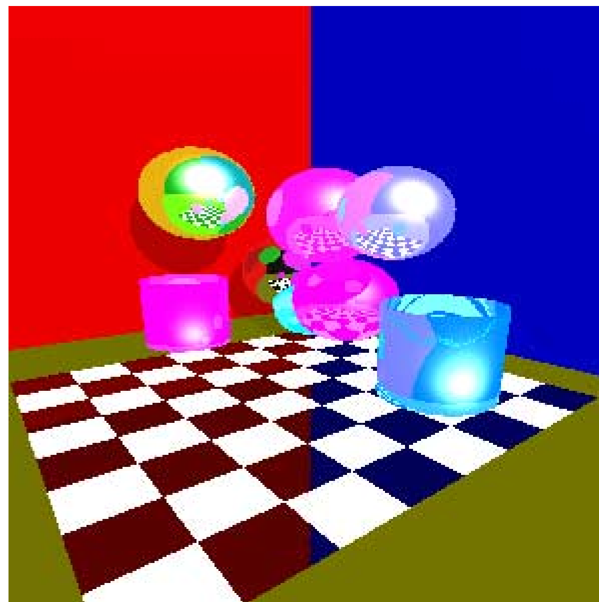

Figure 8 – Image Without Texture

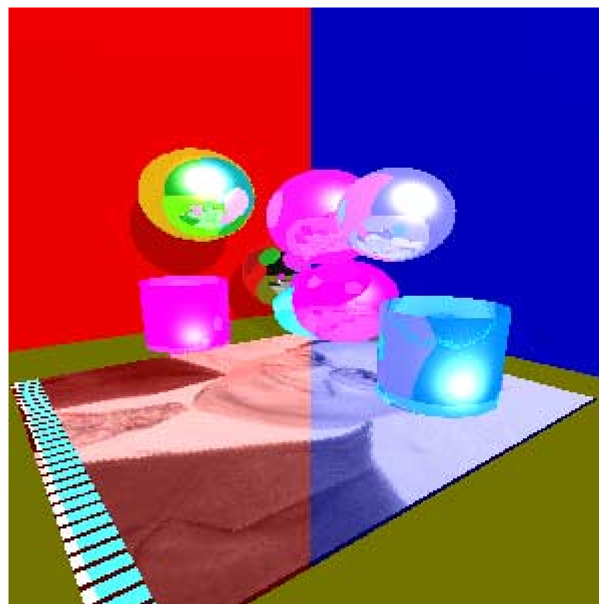

Figure 9 – Image With a Rectangle-Mapped Texture

IEEE COMPUTER SOCIETY

Let us consider a handheld device with a screen size of 200 x 200 pixels (4 x 10^4 pixels), where a clever algorithm can reduce the number of rays to be traced by at least 50% of the number of pixels (2 x 10^4 rays). In our analysis, a game with a small number of objects (approximately 20 objects) will require approximately 8 * 10^6 clock cycles to draw a picture.

Running on a 100 MHz MorphoSys SIMD chip, approximately 12.5 frames per second can be drawn on the screen, which is above game interaction requirements. Running on a 200 MHz MorphoSys chip it is possible to achieve 25 frames per second (real-time) for a scene of the complexity described above.

It is worth reiterating that our main objective is to design and implement a programmable low-power, low-area integrated circuit for simple graphics applications on portable devices. Therefore, we restricted ourselves to a small screen size (with a smaller number of pixels than normal screens) and also to a smaller number of objects than are usually required for highly realistic images.

Table 2 – Clock Cycles per Function

Object	Clock Cycles (per RC)
Plane	50
Rectangle	350
Sphere	500
Illumination & Shading	700
Data Transfer	300
Scaling	200
TOTAL	2100

7c. Resource Estimates

To estimate the VLSI resources required by the targeted reconfigurable cell (RC) of MorphoSys II-G system-on-chip, the entire RC unit was synthesized and technology-mapped to the target cell library (using a CMOS 0.13 µm process) . The area of each block is shown on table 3. From table 2 it can be seen that the RAM and the parallel multiplier are the largest blocks, followed by the register file. Adding the areas of all cells gives a total area of approximately 0.125 mm^2 , which does not include the routing area. Nevertheless, it provides fundamental information to estimate the area resources needed by each RC, and to assess the area of the RC array, which is the largest area of the MorphoSys II-G system-on-chip. The area of the RC unit of MorphoSys I (after routing) using a 0.35 µm was 2mm^2 [18,19].

8. Conclusions

This paper presented an architecture for running interactive ray tracing applications on portable devices such as cell phones, PDAs, and head mounted displays and discusses the main issues related to the mapping of this graphics algorithm using fixed-point arithmetic.

Table 3 – Area of RC Blocks (CMOS 0.13um Process)

CELL	CELL AREA (μm^2)
RAM	56300
ALU	6240
Control	6350
Multiplier	25270
Mux M	2500
Mux C	320
Mux A	4700
Mux B	2280
Mux D	430
Register File	11950
Shift Unit	6950
Rout	1200
TOTAL	124490

The paper showed that a floating-point arithmetic unit, with its associated power and area consumption, can be avoided by using appropriate fixed-point arithmetic and block floating-point operations. It also showed that a computation intensive graphics method like ray tracing can be used, by employing a SIMD architecture, which trades parallelism for frequency of operation, providing significant benefits in power saving, which is essential in portable devices.

To support fixed-point arithmetic and block floating-point operations, some new instructions were required, particularly instructions that support the fast normalization of operands, such as count leading zeros and multiple-bit shift instructions. Equally important was the addition of *guarded instructions* (or *guarded contexts* in MorphoSys terminology) and pseudo branches. These instructions supported the implementation of *if-then-else* constructs which occur in irregular data-parallel algorithms, such as ray tracing and rasterization.

Overall, the paper showed that it is possible to use a system-on-chip reconfigurable programmable device, instead of an application-specific device, trading frequency of operation for area (for lower power consumption), and achieve sufficient performance to run simple interactive graphics applications on portable devices.

9. References

[1]http://www.nandotimes.com/technology/story/393710p-3134641c.html, http://www.idg.net/idgns/2000/10/13/TOKYOEDGECellphonesGetIn.shtml

[2] http://arcadetones.emuunlim.com/;

[3] http://www.virtualresearch.com/; http://www.iisvr.com/products_VR_main.html

[4] B. Patwardhan, "Introduction to the Streaming SIMD Extensions in the Pentium III – Parts I, II and III", Dr. Dobb's Journal, January, 2002.

0-7695-1772-2/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY

[5] M. Fomithchev, "AMD 3Dnow", Dr. Dobb's Journal, Vol. 25, No. 8, pp.40-42, August, 2000.
[6] Motorola, "AltiVec Technology", http://www.mot.com/SPS/PowerPC/AltiVec/index.html
[7] Philips, "Trimedia VLIW processos", http://www.semiconductors.philips.com/markets/industrial/trimedia/
[8] Nvidia, "Nvidia GeForce4 Graphics Processors Family", http://www.nvidia.com/
[9] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. M. C. Filho, - "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications", IEEE Transactions on Computers, pp. 465-481, Vol. 49, No. 5, May, 2000.
[10] M. Meiβner, S.Grimm, W. Straβer, J. Packer and D.Latimer, "Parallel Volume Rendering on a Single-Chip SIMD Architercture", IEEE 2001 Symposium on Parallel and Large-Data Visuallization and Graphics, pp.xxx-yyy, San Diego, Ca, October, 2001.
[11] Foley, J, van Dam, A., Feiner, S.K., Hughes, J.F., *Computer Graphics – Principles and Practice*, Addison-Wesley Publ., 2nd. Ed., 1996.
[12] Watt, A., *3D Computer Graphics*, Addison-Wesley Publ., 1993.
[13] Wald, I., and Slusallek, P., "State of the Art in Interactive Ray Tracing", EUROGRAPHICS 2001.
[14] Advanced Rendring Technologies. The AR250 – a new architecture for ray traced rendering. In Proceedings of the Eurographics/SIGGRAPH workshop on graphics hardware – Hot Topics Session, pages 39-42, 1999.
[15] Weghorst, H., Hooper, G., and Greenberg D.P., "Improved Computational methods for Ray Tracing", *ACM Trans. on Graphics,* 3 (1), pp.52-69, 1984.
[16] Levinthal, A., and Porter, T., "Chap – a SIMD graphics processor", Computer Graphics (Proceedings of SIGGRAPH84), 18(3): 77-82, July, 1984.
[17] Kreeger, K. and Kaufman, A., "PAVLOV: A Programmable Architecture for Volume Processing", SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, Lesbon, Portugal, 1998.
[18] Hwang, K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.
[19] Lee, Ming-Hau, "Design and Implementation of the High-Performance Low-Power MorphoSys Reconfigurable Processor", Ph.D. thesis, Electrical and Computer Engineering Dept, Univ. of California, Irvine, 2000.
[20] Lu, Guangming, "Modeling, Implementation and Scalability of the MorphoSys Dynamically Reconfigurable Computing Architecture", Ph.D. thesis, Electrical and Computer Engineering Dept, Univ. of California, Irvine, 2000.
[21] Singh, Hartej, "Reconfigurable Architectures for Multimedia and Data-Parallel Application Domains", Ph.D. thesis, Electrical and Computer Engineering Dept, Univ. of California, Irvine, 2000.
[22] Ercegovac, M.D. and Lang, T., *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Press, 1994.
[23] Parhami, B., *Computer Arithmetic*, Oxford University Press, 2000.
[24] Frisch, R., Isestein, B., and Stein, S., "Arithmetic Methods trade off precision for numerical range", Electronic Product, 27(21), pp. 75-78,80, April, 1985.
[25] Paar, A., Anido, M.L. and Bagherzadeh, N., "A Novel Predication Scheme for a SIMD System-on-Chip", ACM/IFIP conf., Euro-Par'2002, pp. --,--, Paderborn, Germany, August 2002.
[26] Anido, M.L., Paar, A. and Bagherzadeh, N., "Improving the Operation Autonomy of SIMD Processing Elements by Using Guarded Instructions and Pseudo Branches", DSD'2002, Proc. EUROMICRO Symposium on Digital System Design, North Holland Publ., pp ---,---, Dortumond, Germany, September 2002.
[27] Advanced RISC Machines. *ARM610 RISC Processor*, 1993. Document #: ARM DDI 0004C.

IEEE COMPUTER SOCIETY