Using Mitrion-C to Implement Floating-Point Arithmetic on a Cray XD1 Supercomputer

Kevin K. Liu, Charles B. Cameron, and Antal A. Sarkady
Department of Electrical & Computer Engineering, US Naval Academy (USNA), Annapolis, MD
liu.kevin.k@gmail.com and {cameronc, sarkady}@usna.edu

Abstract

Field-Programmable Gate Arrays (FPGAs) are of interest to the high performance computing (HPC) computing community because they offer lower power consumption and higher throughput compared to traditional processors. Recently, the implementation of floating-point operations on FPGAs has become possible as the amount of memory available on FGPAs has increased. Unfortunately, advances in technology have also increased the complexity of creating hardware designs for FPGAs. In this project, we describe our experiences using the Mitrion-C high-level language to implement floating-point calculations on a Cray XD1. We report resource consumption, throughput, and power consumption and conclude that Mitrion simplifies the hardware design process while successfully harnessing the computational power of FPGAs at little additional cost to power consumption.

1. Introduction

The scientific community is interested in using field-programmable gate arrays for scientific computations because Field-Programmable Gate Arrays (FPGAs) can be targeted for specific applications and achieve greater throughput at a lower power cost.[1–3] However, these gains can usually only be achieved by a user with expert knowledge of hardware design. Therefore, despite improvements in FPGA technology that have allowed their use to become attractive for a wider range of applications, inexperience with hardware design remains a barrier for many.

High-level languages use a variety of approaches to reduce the complexity of hardware design. We chose to use Mitrion-C for this project because it was readily available at the Naval Research Laboratory, where this work was done, and because it is a commercial product with fast and effective support services. Mitrion-C makes hardware design more accessible in two ways. First, algorithms are described in the Mitrion-C programming language, which uses “C-like” syntax and structures, such as functions and loops. Second, the Mitrion Integrated Development Environment (IDE) packages together a user interface, compiler, and simulator. Figure 1 shows the necessary steps of hardware and highlights the steps that Mitrion IDE executes.

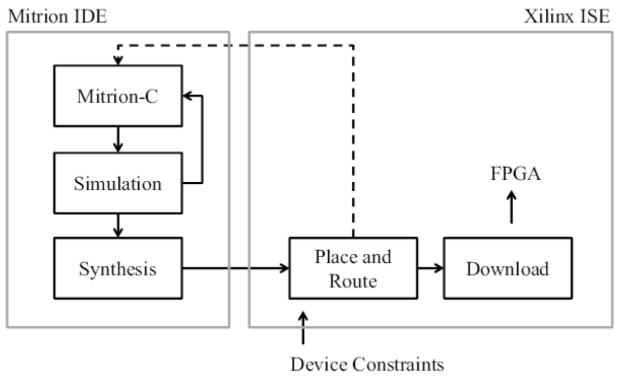

Figure 1. Hardware design flow

In hardware design using a traditional hardware description language (HDL) such as Very High Speed Integrated Circuit HDL (VHDL), both simulation and synthesis are time consuming and synthesis can often fail, requiring modification of the code. The Mitrion IDE simulates and generates VHDL in one step and also estimates whether a design will fit, based on the target hardware’s limitations. Therefore, as long as there are no syntax errors in the Mitrion code, the VHDL synthesis will most likely be successful, with the exception of cases where resource consumption exceeds the resources of the FPGA by a very small margin.

One downside of using a high-level language is that the hardware designer loses a level of control. Although Mitrion-C offers explicit options for pipelining, how it achieves its optimizations is opaque to the user. We sought, therefore, to not only measure the performance of designs using Mitrion-C, but also to predict future performance based on our results.

978-0-7695-3515-9 2008
U.S. Government Work Not Protected by U.S. Copyright
DOI 10.1109/DoD.HPCMP.UGC.2008.40

Report Documentation Page

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUL 2008**	2. REPORT TYPE	3. DATES COVERED **00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE **Using Mitrion-C to Implement Floating-Point Arithmetic on a Cray XD1 Supercomputer**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **United States Naval Academy,Department of Electrical & Computer Engineering,Annapolis,MD,21402**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Proceedings of the DoD HPCMP Users Group Conference (HPCMP-UGC'08), pp. 391?395, 14?17 July 2008

14. ABSTRACT
Field-Programmable Gate Arrays (FPGAs) are of interest to the high performance computing (HPC) computing community because they offer lower power consumption and higher throughput compared to traditional processors. Recently, the implementation of floating-point operations on FPGAs has become possible as the amount of memory available on FGPAs has increased. Unfortunately, advances in technology have also increased the complexity of creating hardware designs for FPGAs. In this project, we describe our experiences using the Mitrion-C high-level language to implement floating-point calculations on a Cray XD1. We report resource consumption, throughput, and power consumption and conclude that Mitrion simplifies the hardware design process while successfully harnessing the computational power of FPGAs at little additional cost to power consumption.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**5**	

2. Methodology

The simulation of the interaction of a ray of light with an optical element—assuming that the element is a conic surface—requires several calculations. We chose to look at two in particular: the intersection point of a ray with an element, and the vector normal to the element's surface at the point of intersection. These two calculations are illustrated in Figure 2.

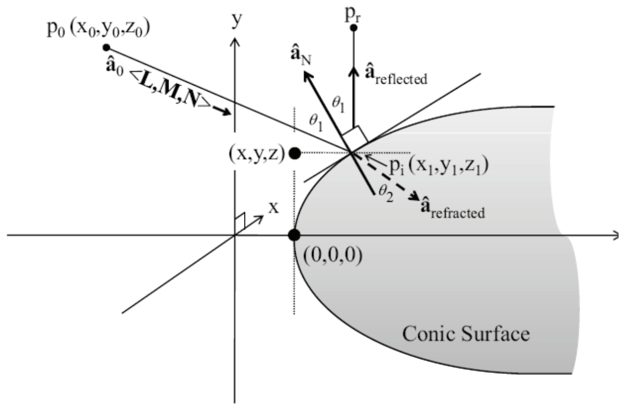

Figure 2. Interaction of a ray of light with an optical element

For our purposes, the two calculations can be reduced to a system of arithmetic operations, as described by Spencer and Murty.[4] For the ray-intersection problem, they are:

$$g = N - c\left(x_0 L + y_0 M + (k+1.0) z_0 N\right) \tag{1a}$$

$$h = c\left(x_0^2 + y_0^2 + (k+1.0) z_0^2\right) - 2z_0 \tag{1b}$$

$$f = c\left(1 + kN^2\right) \tag{1c}$$

$$u = \frac{h}{g + \sqrt{g^2 - fh}} \tag{1d}$$

$$x_1 = uL + x_0 \tag{1e}$$

$$y_1 = uM + y_0 \tag{1f}$$

$$z_1 = uN + z_0 \tag{1g}$$

The ray-intersection calculation requires 11 floating-point additions, 3 subtractions, 19 multiplications, 1 division, and 1 square root.

The system of equations for the normal-vector calculation is presented next.

$$v = u\left(x^2 + y^2\right) \tag{2a}$$

$$a = \sqrt{1 - v} \tag{2b}$$

$$p = 1 + a \tag{2c}$$

$$q = ap \tag{2d}$$

$$r = pq \tag{2e}$$

$$s = 2q \tag{2f}$$

$$w = c / r \tag{2g}$$

$$b = w\left(s + v\right) \tag{2h}$$

$$\frac{\partial q}{\partial x} = bx \tag{2i}$$

$$\frac{\partial q}{\partial y} = by \tag{2j}$$

$$e = \sqrt{\left(\frac{\partial q}{\partial x}\right)^2 + \left(\frac{\partial q}{\partial y}\right)^2 + 1} \tag{2k}$$

$$f = 1 / e \tag{2l}$$

$$\hat{\mathbf{a}}_N = \left(\frac{\partial q}{\partial x}, \frac{\partial q}{\partial y}, f\right) \tag{2m}$$

In this calculation, the term u is defined as u=(1+k)c^2. The normal-vector calculation requires 5 additions, 1 subtraction, 13 multiplications, 2 divisions, and 2 square roots. The result is given as the three components of the normal vector, $\hat{\mathbf{a}}_N$, as shown in Figure 2.

One might observe that in the ray-intersection calculation, the term k+1.0 is used twice—once in Eq. 1a and again in Eq. 1b. It would be expected, then, that Mitrion would simply use the same result twice rather than perform two identical calculations. However, the number of floating-point units reported reflects the output of the Mitrion simulator. We also wrote a separate program to isolate this issue and found that separate additions were in fact implemented. Therefore, the count of 11 additions for the ray-intersection calculation is accurate.

3. Implementation

We used Mitrion-C version 1.4 to implement the two calculations. Figure 3 shows the data flow between the Mitrion-C and host programs. Each of the Quad-Data Rate (QDR) memories directly available to the Virtex-II Pro contains 4 MB of space for input/output, for a total of 16 MB of input and output. Since many scientific applications require more than 16 MB of input and output, a host program is needed to mar-shall data between the

FPGA's memory and host memory present on the same compute node.

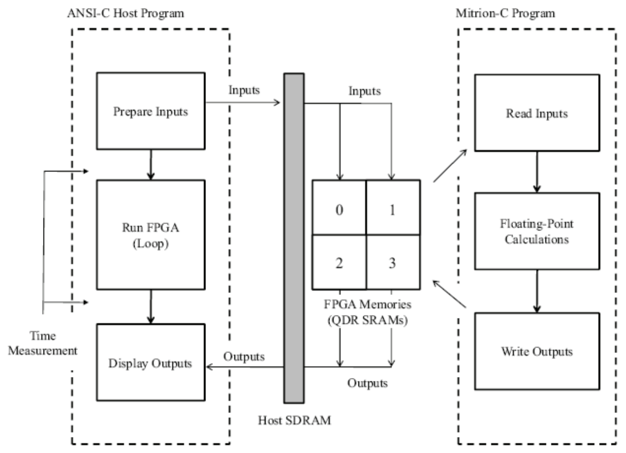

Figure 3. Data flow between host and FPGA programs

We wrote the host program using the American National Standards Institute's standard for C (ANSI-C) and ran it on one of the Advanced Micro Devices (AMD) Opteron 275 processors on the same compute node as the FPGA. The Cray XD1 uses an interconnect system that allows data transfer between the FPGA and host RAM at a rate of 3.2 GB/s.[5] Mitrion-C uses the full bandwidth provided by Cray.

In the host program, each of the FPGA's QDR memories is treated as an array. The host program loads values into the arrays, sends the FPGA a start signal using a function provided by Mitrionics, and reads the results after it receives a *don*e signal back from the FPGA.

The Mitrion-C program was split into three functions that: 1) read the inputs from QDR memory, 2) performed floating-point calculations, and 3) wrote the results to a different QDR memory. We stored our data in a `list` data structure and ran the program in a `foreach` loop. This combination explicitly instructs the Mitrion compiler to automatically pipeline the design, as stated in the Mitrion-C documentation[6].

4. Results

As a benchmark we compared the performance of the Mitrion-C implementations of the ray-intersection calculation and normal-vector calculation to ANSI-C programs. The power and throughput measurements isolated the calculation-intensive portions of each of the programs.

4.1. Throughput

Each of the 4 MB memories available to the the Virtex-II Pro has a bit-width of 64 bits. We implemented our calculations using the 32-bit width IEEE single-precision floating-point representation. This means that each memory can hold 2^{20} or 1 048 576 floating-point numbers. We initially used two QDR memories for input and two for output. In the case of the normal-vector calculation, which requires four inputs, one set of four inputs could be read each clock cycle. However, in the case of ray-intersection calculation, using only two memories for input required two clock cycles to read each set of eight inputs. Therefore, we wrote a second version of the ray-intersection calculation that used four memories for input and observed a doubling in throughput, as shown in Table 1.

Table 1. Throughput measurements

	Operton 275	**Virtex-II Pro**
Rays traced	1 073 741 824	
Ray-intersection calculation		
Time (s)	219.54	21.49
Throughput (rays/s)	4.891×10^6	4.996×10^7
Ray-intersection, using 4 inputs		
Time (s)	—	10.75
Throughput (rays/s)	—	9.988×10^7
Normal-vector calculation		
Time (s)	114.79	10.75
Throughput (rays/s)	9.354×10^6	9.988×10^7

Although all four of the FPGA's memories were used for input, two of the memories had to be used for output as well. Mitrion-C provides memory synchronization commands that enable bidirectional use of the FPGA's memories with no effect on throughput. We also checked a representative set of data to ensure no data corruption or overlapping had occurred.

4.2. Resource Consumption

The resource consumption reported in Table 2 was taken from the report generated by the Xilinx Integrated Synthesis Environment (ISE) after the place-and-route step. The amount of resources consumed by each design gives insight into how much additional optimization is possible.

In the case of the normal-vector calculation, the measured throughput was 99.88×10^6 rays/s, which corresponds to approximately one ray calculated for every clock cycle, given a 100MHz clock. Since the QDR memories had a bit-width of 64 bits, or 8 bytes, the throughput of each memory was about $8\,\frac{\text{bytes/ray}}{\text{memory}}\times\left(99.88\times10^{6}\right)\frac{\text{rays}}{\text{s}}=799.04\,\frac{\text{MB/s}}{\text{memory}}$. This result indicates that the memory was used at very near its maximum theoretical bandwidth of 3.2 GB/s, or 800 MB/s per memory. Therefore, the only way to improve throughput would have been to use additional QDR memories as both inputs and outputs.

The normal-vector calculation consumed over 70% of slices, the term Xilinx uses to refer to the basic reconfigurable logic unit within an FPGA. Had we used additional QDR memories, requiring additional floating-point logic, we would likely have exceeded the resources of the FPGA. Low-level customization beyond the capabilities of Mitrion-C would have been required to implement more floating-point calculations without making the design too large.

In contrast, the floating-point logic implemented in the ray-intersection calculation was capable of producing one calculation per clock cycle because it was implemented within a `foreach` loop and so throughput was only limited by the fact that the input memories could provide one set of inputs every two clock cycles. Using four memories for input instead of two did not affect the resources needed to implement the floating-point logic, but removed the bottelneck imposed by the input memories. Table 2 shows that using four memories for input instead of two cost a small amount of resources and did not exceed the resources of the FPGA.

Table 2. Resource consumption comparison

Resource (Total)	Implemented (Percent)
Ray-intersection calculation	
Slices (23616)	19 044 (81%)
Flip Flops (47 232)	26 508 (56%)
4-input LUTs (47 232)	26 250 (56%)
Block RAMs (232)	25 (11%)
Multipliers (232 18×18)	72 (31%)
Ray-intersection, using 4 inputs	
Slices (23616)	20 593 (87%)
Flip Flops (47 232)	26 688 (56%)
4-input LUTs (47 232)	26 579 (56%)
Block RAMs (232)	25 (11%)
Multipliers (232 18×18)	72 (31%)
Normal-vector calculation	
Slices (23616)	16 571 (70%)
Flip Flops (47 232)	21 670 (46%)
4-input LUTs (47 232)	20 466 (43%)
Block RAMs (232)	23 (11%)
Multipliers (232 18×18)	72 (31%)

4.3. Power Consumption

We measured power with Cray's Hardware Supervisory Subsystem (HSS), software that runs on the management processor of each chassis within the Cray XD1 and monitors the health of the system. Table 3 reports our results. Our measurements showed that a node with an FPGA will consume $\frac{130.94-102.65}{102.65}=27.56\%$ more power than a node without an FPGA consumes while idling. However, we also found that in the case of a node with an FPGA present, using the FPGA for processing requires at most $\frac{143.66-139.84}{139.84}=2.73\%$ more power than implementing an equivalent calculation on the Opteron 275 processor alone.

Table 3. Power measurements

Node Type	Implementation	Total Power (watts)
No FPGA	Idle	102.65
FPGA	Idle	130.94
Ray-intersection calculation		
No FPGA	Sequential Only	110.87
FPGA	Sequential Only	139.57
FPGA	FPGA	141.13
Normal-vector calculation		
No FPGA	Sequential Only	111.84
FPGA	Sequential Only	139.84
FPGA	FPGA	143.66

We only measured the power consumed when running the version of the ray-intersection calculation that used two memories for input. However, the version that used four memories is unlikely to draw significantly more power, judging by the similarity in resources consumed.

5. Discussion

As mentioned before, the maximum bandwidth of the interconnect, between the FPGA's QDR memories and the host memories, is 3.2 GB⁄s. This means that each of the four QDR memories makes up 800 MB⁄s of that total. Since each FPGA memory can read or write 64 bits (8 bytes) every clock cycle, the 100 MHz clock used by Mitrion makes use of the maximum 800 MB⁄s bandwidth of the memories.

Our measurements confirmed that a throughput very near the limit of the memories—799.04MB/s in the case of the normal-vector calculation—could be maintained over a large sample of data. We conclude that Mitrion-C is a straightforward way to achieve the maximum throughput allowed by the memory bandwidth, given that the intended design fits on the target FPGA.

6. Conclusion

In this paper, we explored the ability of the high-level language Mitrion-C to simplify the implementation of floating-point operations on FPGAs. We found that Mitrion-C was different enough from ANSI-C to require a significant investment of time to be able to use it effectively, but that Mitrion-C significantly reduces the time spent in the hardware design cycle. In terms of throughput, we found that Mitrion-C could achieve the maximum theoretical throughput allowed by memory bandwidth in cases where the design easily fit on the FPGA and memory operations could be completed in one clock cycle. However, we observed that low-level programming would still be needed to make small tradeoffs between throughput and resource consumption. Finally, we found that maintaining FPGAs requires roughly a constant 30% increase in power consumption, but that in cases where FPGAs are present on a compute node, using them for processing requires roughly only 3% additional power over using a sequential processor alone. We recommend Mitrion-C as a tool to exploit the processing power of FPGAs, given that the intended application does not exceed the resource limits of the target hardware.

Acknowledgments

This work was supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program at the Naval Research Laboratory (NRL). It was also supported by Kenneth Sarkady, Head of the Infrared Countermeasures Systems Section. In addition, the authors would like to thank the many scientists including Jeanie Osburn, Wendell Anderson, Rick Hurd, and Ray Yee at NRL, who willingly shared their time and expertise.

References

1. Koo, J., A. Evans, and W. Gross, “Accelerating a medical 3D brain MRI analysis algorithm using a high-performance reconfigurable computer.” *Field Programmable Logic and Applications, 2007, FPL 2007. International Conference*, pp. 11–16, 27–29 Aug. 2007.

2. Kindratenko, V.V., R.J. Brunner, and A.D. Myers, “Mitrion-C Application Development on SGI Altix 350/RC100.” *Field-Programmable Custom Computing Machines, FCCM 2007, 15th Annual IEEE Symposium*, pp. 239–250, 23–25 April 2007.

3. Mitrionics AB, *Accelerate your applications—unleash the massive performance of FPGAs*, available online at http://www.mitrion.com/press/Mitrion product_brief.pdf.

4. Spencer, G.H. and M.V.R.K. Murtry, “General ray-tracing procedure.” *J. Opt Soc. Ameri.*, vol. 52, no. 6, pp. 652–678, June 1962.

5. Cray, Inc., “Cray XD1 datasheet.” *Cray Inc., Tech. Rep., June 2005*, available online at http://www.cray.com/downloads/Cray XD1_Datasheet.pdf.

6. Mohl, S., The Mitrion-C programming language, Mitrionics Inc., 2006, available online at http://www.mitrionics.com/.