Implementing Closed-Form Expressions on FPGAs Using the NAL, with Comparison to CUDA GPU and Cell BE Implementations

Robin Bruce[1,3], Javier Setoain[2], Richard Chamberlain[3], Malachy Devlin[3], Rosa M. Badia[4]
[1]Institute for System Level Integration, [2]Universidad Complutense de Madrid, [3]Nallatech Ltd, [4]Barcelona Supercomputing Center
robin.bruce@sli-institute.ac.uk, jsetoain@pdi.ucm.es, r.chamberlain@nallatech.com, m.devlin@nallatech.com, rosa.m.badia@bsc.com

Abstract

This paper outlines the Nallatech Accelerator Layer (NAL) and its relationship to Intel's Accelerator Abstraction Layer. The NAL is looked at in its academic context. Hardware platforms that support the NAL are discussed: the Nallatech H101, the Intel FSB-FPGA Module and the BenOne PCIe. The Intel QuickAssist Technology initiative and its associated Accelerator Abstraction Layer (AAL) are introduced.

To demonstrate the NAL system, two closed-form expressions are implemented. These functions are single-precision floating-point, and make use of arithmetic operations and elementary functions. The functions selected were the probability density function (PDF) and the Black-Scholes-Merton options pricing formula (BSM). These functions were implemented on a dual-core Opteron, a Nallatech H101 card using the NAL, a Nallatech BenOne PCIe card, an NVIDIA G80 using CUDA and a Cell BE system. The GPU system showed the best silicon performance for the implementation of these kernels. Including data transfer times, the BenOne PCIe had the highest performance.

1. Introduction

A general-purpose reconfigurable computer uses programmable logic for its principal processing elements. The concept of a reconfigurable computer is credited to Gerald Estrin who first described the concept in 1960 [1]. Reconfigurable computers today use field-programmable gate arrays (FPGAs) as their programmable logic element. Though not designed with reconfigurable computers in mind, SRAM-based FPGAs from Xilinx and Altera provide the most practical mix of configurability, density, performance, availability and affordability. Silicon architectures intended to be better suited to reconfigurable computing have been conceived, an example being Ambric's processor array [2]. However, their long-term commercial viability is not proven. FPGAs are capable of extraordinarily high performance in certain areas of computation when compared to conventional commodity microprocessors. El-Ghazawi et al give an overview of the state of the art in a recent paper [3]. Speedups of between 1 and 4 orders of magnitude are reported in the areas of DNA and protein sequencing and in cryptographic secret-key deduction for a range of ciphers. These applications operate at the bit-manipulation level. FPGAs are undoubtedly the most capable commodity devices for such parallel, bit-manipulation applications. Similarly, FPGAs can lag commodity microprocessors by orders of magnitude in applications that are inherently serial and iterative in execution, given the commodity microprocessor's ~15x advantage in clock rate, and its unparalleled on-chip SRAM memory structure.

Figure 1 below gives a graphical representation of the relative advantages of FPGAs when compared to stored-program processors, of which commodity microprocessors are a subclass. This representation is the authors' reinterpretation of Cantle's view on the subject [4]. Cantle's view was itself influenced by a project of the US Air Force Research Laboratory's Advanced Computing Technology Branch (IFTC), which looked at advanced computing technologies for novel information processing paradigms [5]. The diagram shows the efficiency of a selection of modern processing technologies for application data types that range from bit-level processing to symbolic processing. Efficiency is an admittedly subjective composite of size, weight, energy consumption, absolute performance and time to solution. The diagram demonstrates how, with each generation, stored-program processors and FPGAs are evolving from their respective symbolic and bit-level roots to become ever more capable vector/streaming processors.

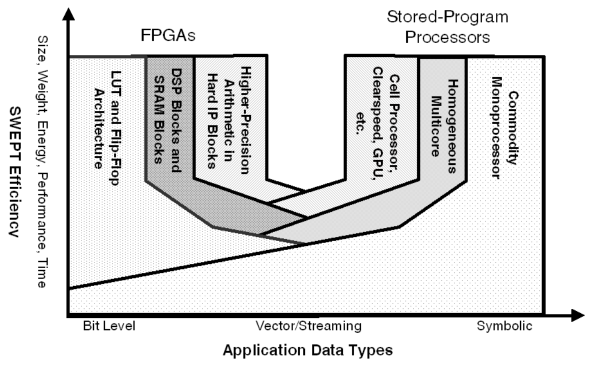
FPGAs
Stored-Program Processors
LUT and Flip-Flop Architecture
DSP Blocks and SRAM Blocks
Higher-Precision Arithmetic in Hard IP Blocks
Cell Processor, Clearspeed, GPU, etc.
Homogeneous Multicore
Commodity Monoprocessor
Size, Weight, Energy, Performance, Time
SWEPT Efficiency
Bit Level
Vector/Streaming
Symbolic
Application Data Types

Figure 1: FPGAs versus stored-program processors

Researchers who have worked with reconfigurable computers over the last two decades have reported on the perennial difficulties in programming them. Many papers have been published on these difficulties and in the efforts and progress being made to overcome them [6],[7]. The fundamental problem in programming reconfigurable computers is in how to abstract the complexities of designing hardware. The aim is to have a process that resembles software development in its program development and debug, without excessively sacrificing performance.

This paper looks at the Nallatech Accelerator Layer (NAL). The NAL is a set of C++ classes that functions as a system-level design environment for Nallatech reconfigurable computing platforms. The NAL was designed to complement Intel's Accelerator Abstraction Layer (AAL) as a programming environment for the FSB-FPGA accelerator. The FSB-FPGA is a reconfigurable computing platform that has been developed in a partnership between Intel, Nallatech and Xilinx, and a separate partnership between Intel, XtremeData and Altera. The FSB-FPGA and the AAL are components in Intel's wider QuickAssist technology intitiative. The Intel AAL provides consistent libraries and a consistent development interface to hardware accelerators, to allow software portability between accelerators. In addition to Intel AAL compatibility, the Nallatech Accelerator Layer was designed with backwards compatibility to Nallatech's FUSE runtime services API and Nallatech's hardware products such as the H101 FPGA-based accelerator, and BenOne PCIe. Nallatech's high-level language compiler DIME-C is a prominent component of the NAL. It allows for the compilation of ANSI C code to VHDL targeted at Xilinx FPGAs. DIME-C is discussed in greater detail in a previous paper [8].

2. Related Work

The design goals of the Nallatech Accelerator Layer (NAL) are mirrored by those of the USURP project [9]. Holland et al recognize that "the current reliance on custom hardware wrappers and proprietary software APIs is detrimental to the efficiency of FPGA-accelerated application development". To overcome the custom hardware wrappers problem, the NAL automatically generates the HDL code that bridges the gap between an algorithm implementation and the hardware-driver boundary. The proprietary software API problem is addressed firstly by abstracting away the underlying runtime service APIs within the NAL, and secondly by the development of the Intel AAL runtime service API, which aims to provide a vendor-neutral API for accelerators in Intel-based platforms. The USURP framework is intended by its developers to help lay the foundation for a standard API for the FPGA industry. This would place the AAL API layer below the USURP software API in the abstraction hierarchy.

Andrews *et al* have developed hthreads [10]. The authors advocate managing the co-operation of software and hardware in reconfigurable computers by having asynchronous software and hardware threads that interact as peers, with shared global memory. POSIX threads (pthreads) are the basis for their work to which they have added hardware threads and named the result hthreads. The authors are looking to take operating systems and middleware layer abstraction across the CPU/FPGA boundary and into the FPGA itself. The hthreads system is limited to a single chip, with the microprocessor located on the FPGA die. In the systems the NAL targets, the host microprocessor or microprocessors are separated from the FPGA fabric by a communications channel, such as PCI-X or the Intel FSB. There are similarities in approach to hthreads, and in many ways a NAL-implemented accelerator functional unit (AFU) behaves as a hardware thread. Certainly, hardware and software threads in the NAL communicate through shared memory, in a zero-copy fashion.

A team of researchers, based primarily at Northeastern University, has developed Vforce [11] (VSIPL++ for Reconfigurable Computing). Vforce is designed to allow the same application code to run on different reconfigurable computing platforms, and to permit the runtime binding of applications to hardware. The authors believe that application-level code needs to be separated from platform-specific code, so that no hardware-specific code is required in the application code. Vforce is based on VSIPL++, the Vector, Signal and Image Processing Library, which provides an object-oriented library of commonly used signal and image processing algorithms via a C++ API. VSIPL++ is a standard API designed to provide application-level portability between microprocessor-based platforms. NAL complements the Vforce approach in that it is primarily aimed at providing a single source environment for the development of reconfigurable computing accelerated kernels, inclusive of bitstream generation. There is however significant overlap between the aims of Vforce, and the aims of the Intel AAL. Vforce remains unique in its links to VSIPL++, and its neutrality with respect to microprocessor vendor.

3. NAL-Compatible Hardware Platforms

The Nallatech H101 card is shown below in Figure 2. The card accommodates either a Xilinx Virtex-4 LX100 or LX160 as the user FPGA. The card connects with a host computer via the PCI-X interface. The NAL supports the H101 card. The non-expert version of the NAL will allow a user to interface DIME-C compatible ANSI C code with a host C++ program. The DIME-C either runs in emulation mode on the host processor, or it is implemented on the FPGA hardware, with communication occurring via the PCI-X interface. When communicating across the PCI-X interface, the NAL abstracts away the details of the underlying FUSE Application Programming Interface (API). The FUSE API handles all aspects of finding, registering and binding the H101 accelerator card as well as managing the DMA transfers across the PCI-X interface.

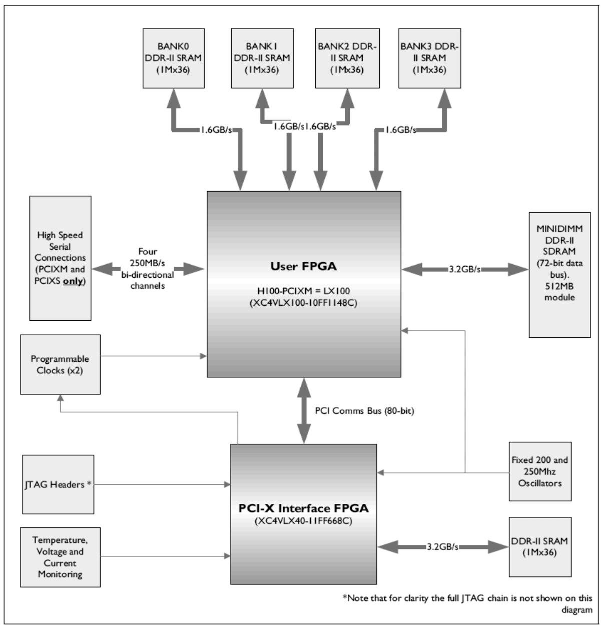
BANK0 DDR-II SRAM (1Mx36)
BANK1 DDR-II SRAM (1Mx36)
BANK2 DDR-II SRAM (1Mx36)
BANK3 DDR-II SRAM (1Mx36)
1.6GB/s
1.6GB/s
1.6GB/s
1.6GB/s
High Speed Serial Connections (PCIXM and PCIXS only)
Four 250MB/s bi-directional channels
User FPGA
H100-PCIXM = LX100 (XC4VLX100-10FF1148C)
3.2GB/s
MINIDIMM DDR-II SDRAM (72-bit data bus). 512MB module
Programmable Clocks (x2)
PCI Comms Bus (80-bit)
JTAG Headers *
PCI-X Interface FPGA (XC4VLX40-11FF668C)
Fixed 200 and 250Mhz Oscillators
Temperature, Voltage and Current Monitoring
3.2GB/s
DDR-II SRAM (1Mx36)
*Note that for clarity the full JTAG chain is not shown on this diagram
Figure 4: H101-PCIXM Functional Diagram

Figure 2 – Nallatech H101 card

The NAL is also compatible with the Nallatech BenOne PCIe platform. The BenOne can host a number of Nallatech DIME-II modules, giving users a range of FPGA, memory and I/O options in building a reconfigurable computing platform. The BenOne PCIe has a full duplex communications bandwidth of 1 Gbyte/s
Figure 3 below shows an abstraction model of the FSB-FPGA accelerator module, connected by a front-side bus (FSB) connection to its Intel Xeon host processor via shared system memory. In the diagram, elements that are developed by the end user or a third-party library developer are shown in white boxes. Light grey denotes elements for which Intel is responsible, or Intel and Xilinx in the case of the Accelerated Hardware Module (AHM) core that manages communication across the FSB connection. Dark grey shows elements for which Nallatech is responsible. Elements in dashed-line boxes are optional elements in the construction of an FPGA-accelerated application. Section 5 gives more detail of the hierarchy on the FPGA in the case of NAL-implemented acceleration.

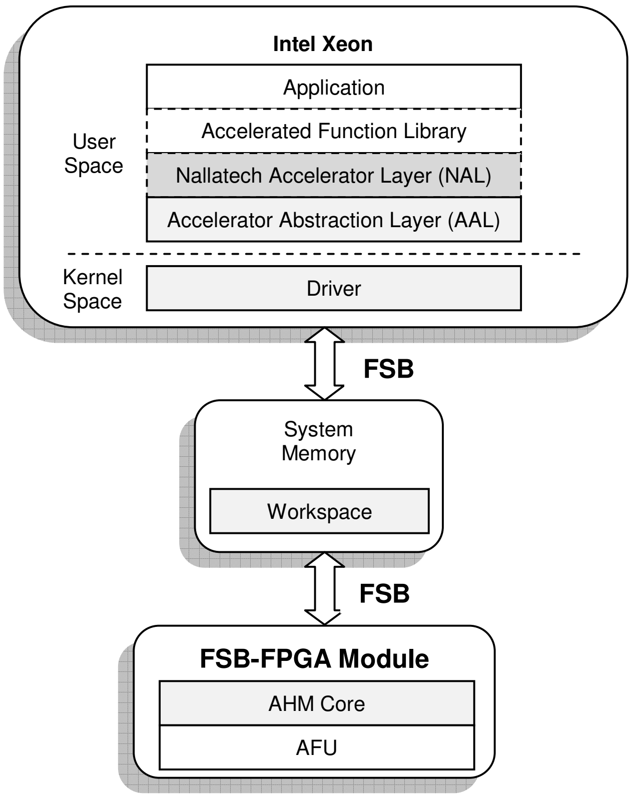

Figure 3 – FSB-FPGA abstraction model

Figure 4 below shows the common and distinguishing elements of the FSB-FPGA and H101 hardware platforms.

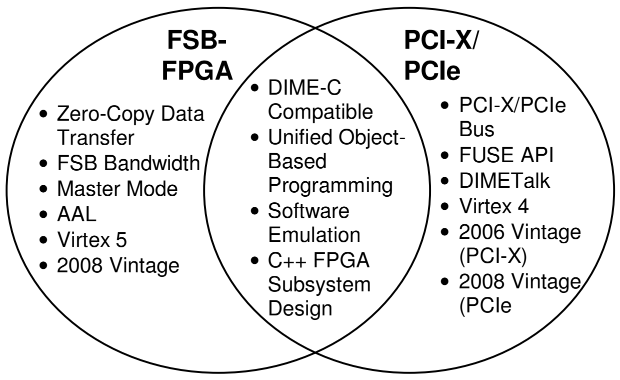

Figure 4 – Comparison of FSB-FPGA and H101 platforms

4. Intel Accelerator Abstraction Layer (AAL)

This section summarizes publicly-available material on the AAL [12],[13].

At present, accelerator vendors must develop their own platform-level services. This adds to the cost of developing accelerators, and prevents inter-accelerator migration by users. The AAL defines common protocols for communicating data and instructions to and from FSB-FPGA accelerators, as well as common policies for managing memory and dealing with exceptions. The AAL handles *canonical functionalities.* Discovery, registration, and binding are examples of canonical functionalities. The AAL is intended to remain constant in the face of unforeseen developments in hardware accelerators. The AAL programming model will also allow an accelerator to be shared among multiple applications. In addition, the AAL does not place limits on the number of threads or cores, meaning that it is thread and multi-core safe. The AAL has two major components: a common Unified Accelerator Interface (UAI) and Accelerator Abstraction Services (AAS). Intel aim to provide compatibility for the AAL with the most commonly used operating systems and programming environments. Applications will have access to a range of acceleration technologies without requiring any changes at the application level. For FSB-coupled accelerators Intel will provide register transfer logic (RTL) and drivers necessary to develop FSB solutions. The key features of the AAL are:

- A high-performance, low-level software framework to support communications between the CPU and the accelerator
- Optimal management of shared memory
- Zero-copy memory transfer between host and FPGA module
- Shared memory blocks are mapped and locked into user space and accessible to the accelerator module
- Accelerator developers freed from developing the optimal software stack for each OS
- Compatible with the popular programming environments and languages
- Accelerated application is insulated from the acceleration technology, so it is possible to migrate from one technology to another
- Allows for portable application domain-specific APIs to abstract away hardware complexities and specificities
- Allows lower level service API access, for lower level interactions with hardware
- AAL configuration database manages bitstreams and associated metadata of accelerated functional units
- First hardware platform for the AAL is the FSB-FPGA range of accelerators

5. Nallatech NAL

The NAL is a set of C++ classes that functions as a system-level design environment for Nallatech reconfigurable computing platforms. The NAL approach permits the system-level modeling of multiple DIME-C blocks, something that in the previous DIMETalk-based system could not be reliably modeled at the software level. Figure 5 shows how the NAL sits atop the AAL. In this diagram, the AAL is split into its two components, the Accelerator Abstraction Services (AAS) and the Accelerator Interface Adaptor (AIA). This diagram shows the shim layer that sits between the user side of the FSL interface. The shim layer consists of VHDL generated at compile time that interfaces between the FSL interface and the DIME-C interface buffers that vary with the function of the DIME-C block.

The NAL allows for mixed software-hardware applications to be written from a single source environment. The computational kernel to be accelerated on the FPGA is written in the DIME-C compatible subset of ANSI C. Code that represents the system-level design on the FPGA is written using the relevant NAL C++ classes. The system-level design process connects computational kernels to each other and to memory resources such as FIFOs and memory blocks. All other aspects of communication from host computer to attached FPGA accelerator are also written using the NAL's C++ classes. This same C++ environment is where the user can write conventional application code to run on the microprocessor, in parallel to the FPGA accelerator.

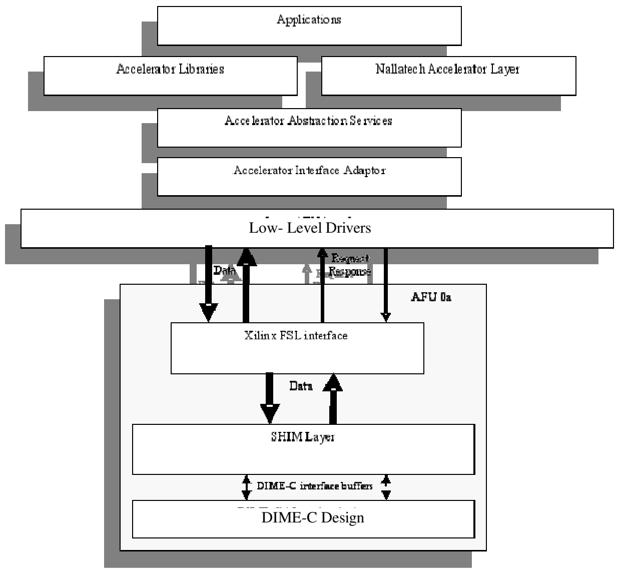

Figure 5: System overview of an application using an FSB-FPGA accelerator module

Table 1 below shows the elements that makes up the Nallatech Accelerator Layer non-expert tool flow.

Table 1: Elements of non-expert tool flow

Nallatech Accelerator Library	Nallatech accelerator libraries required for creating a new AFU.
Board package libraries	Nallatech Accelerator Library board packages to support all available accelerators.
Xilinx FSB Interface	AFU FSB interface netlist provided by Xilinx.
Shim Layer	The shim interface provides a more convenient interface to the FSB.
Makefile	Makefile for creating bitstream for appropriate accelerator technology.
DIME-C	DIME-C compiler.
XML to VHDL binary	Converts system XML description to equivalent VHDL.
AAL & Drivers	The AAL will be used to control the host interface software and includes the device drivers for communication to the FSB module.

6. Closed-Form Expression Implementation

As examples of the type of application that can be implemented using only the NAL high-level design environment and its associated low-level libraries now follows.

Two closed-form mathematical expressions were implemented using DIME-C, the *Probability Density Function* and the *Black-Scholes-Merton* options pricing formula. These functions are *closed form* in the sense that the function outputs depend only on their input and they can be computed using standard arithmetic operators and elementary functions. The performance results for the implementation of these functions is given for two Nallatech cards compatible with the NAL, the PCI-X H101 card, with a Xilinx Virtex-4 LX160 FPGA and the PCI Express BenOne card with a BenADC-3G module, also with an LX160 FPGA. To implement the functions on the FPGA, the Nallatech math library was used [14]. The same kernel source code used for the DIME-C code was recompiled for three other architectures.

The first is a pure software compilaton running on a dual-core AMD Opteron 280 2.4 GHz with 2GiB of RAM running Windows XP with the gcc 3.4.2 compiler. The implementation is multithreaded using pthreads to make best use of the two cores on the processor die. The programs ran with real-time priority to increase performance and reduce OS interference, and the results were averaged over multiple runs. To the authors' best knowledge the kernel code is not structured in manner that would disadvantage the microprocessor. In fact, it has been the authors' experience that kernel code restructured with the DIME-C compiler in mind has actually improved software performance. When code is written in a manner that benefits a vectorizing compiler such as DIME-C, it seems logical that it would also improve caching

The second alternative platform was an NVIDIA G80 GPU platform. CUDA was used to program the device. The GPU card was attached via the PCI Express bus, providing a nominal 1.6 GBytes/s data transfer bandwidth. The card had 768MiB of RAM. The G80 GPU used supports CUDA capability 1.0, meaning that overlapped memory transfer and computation were not possible, as is the case with newer capability 1.1 cards. Given that data transfer time dominated in calculations, this had minimal effect on the results.

The third and final architecture targeted was an IBM Cell blade with 4 Cell BE processors and 1 GiB of RAM, which was programmed using the CellSs high-level language framework [15].

In all the implementations presented here, a single processing die was used, i.e. one LX160 FPGA, one dual-core processor, one Cell BE processor and one G80 GPU. All implementations used single-precision IEEE754 *format* floating-point numbers and operators. Although all used the IEEE754 format, the GPU, Cell and FPGA implementations are not fully compliant to the IEEE754 standard. It is not known to the authors if the Opteron carried out the single-precision operations to the full IEEE754 specification, though support for denormal numbers is presumed.

The scenario for the results presented here is that there is a fictional application running on the host processor(s) that has a constant stream of input values for which its need output values from the closed-form function implemented on the attached accelerator. The throughput measurement in terms of mega operations per second (MOPS) is desired. Results are presented both taking bandwidth limitations into account, to show the performance the host application would experience, and without bandwidth limitations to show the silicon potential that could be tapped with higher bandwidth interconnect. For both of the functions implemented, there are inputs that are considered to be *parameters,* as they require updating so irregularly as to make a negligible impact on the data transfer bandwidth consumption.

The first closed-form expression implemented was the probability density function, shown below in equation (1). On the LX160 FPGAs targeted in both the H101 and BenOne implementations, six probability engines could fit, running at 100 MHz. The units are fully-pipelined, giving an aggregate throughput potential of 600 MOPS. However, given the bandwidth of the PCI-X and PCI express connection between the FPGA boards and the microprocessor system, the engines cannot be kept busy. For each result for this function, 4 bytes of input must be transferred from the host to the attached accelerator, and four bytes of output must be transferred back to the host.

	Probability Density Function		Black Scholes Merton	
	Without Data Transfer (MOPS)	With Data Transfer (MOPS)	Without Data Transfer (MOPS)	With Data Transfer (MOPS)
Opteron	N/A	**4.75**	N/A	**3.4**
H101 PCI-X (LX160)	**600**	**74.4**	**200**	**49**
Cell BE	**195**	**189**	**27.7**	**26.2**
G80	**5959**	**205**	**1276**	**110**
BenOne PCIe (LX160)	**600**	**250**	**200**	**125**

$$P(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1)$$

The second function implemented was the Black-Scholes-Merton options pricing formula shown below in equation (2). The formula gives the price *C* of a European call option with exercise price *K* on a stock currently trading at price *S*, *i.e.*, the right to buy a share of the stock at price *K* after *T* years. The constant risk-free interest rate is *r*, and the constant stock volatility is σ.

$$C(S,T) = S \cdot \Phi(d_1) - K \cdot e^{-rT} \cdot \Phi(d_2)$$
$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2) \cdot T}{\sigma \cdot \sqrt{T}} \quad (2)$$
$$d_2 = d_1 - \sigma \cdot \sqrt{T}$$

Φ is the standard normal cumulative distribution function, shown in equation (3). The error function, though not theoretically closed form, can be adequately evaluated in single-precision arithmetic by means of a Taylor expansion, making it closed-form from a computational perspective.

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \quad (3)$$

In the implementations we assume that there is a large portfolio of options on a relatively small number of stocks. This is realistic, as there are only a limited number of openly-traded securities (e.g. IBM common stock) on which a large number European-style options can be written. For the FPGA implementation, two such engines could fit onto the LX160 FPGA, each fully pipelined running at 100MHz, for a total potential throughput of 200 MOPS. For each execution of the function, 8 bytes of input data are written from the host to the accelerator, and 4 bytes of output data are written from the accelerator to the host.

7. Results

The results are shown below in Table 2. They show that the Opteron had the weakest performance for the functions. The GPU had the strongest *silicon performance*, the performance discounting data transfer. When taking into account the data transfer, then the outcome depended on the method of interconnect used. Amongst the accelerators, the PCI-X H101 FPGA accelerator card had the lowest overall, transfer-inclusive performance, followed by the Cell Processor and the GPU, with the BenOne implementation coming out on top. Cell had the lowest silicon potential of the accelerators, but was most balanced in terms of silicon potential and data transfer bandwidth.

Table 2 – Peak Streaming Performance for Various Architectures

8. Conclusions

The Nallatech Accelerator Layer (NAL), a high-level programming framework for Nallatech FPGA accelerator platforms, has been presented. The relationship of the NAL to Intel's Quickassist initiative and its Accelerator Abstraction Layer (AAL) have been detailed. Relevant comparisons have been made to existing academic efforts in this area.

The results indicate that NVIDIA GPU silicon outperforms FPGA fabric for the computation of closed-form mathematical expressions on large datasets. Both outperformed the Cell BE processor in this case, though the authors do not feel confident to make generalized conclusions about this result. All three accelerator technologies significantly outperformed the Opteron processor. For both the FPGA and the GPU, the performance of the accelerators was not limited by the silicon but by the interconnect. It is reasonable to expect this situation to persist with evolutions in GPUs, FPGAs and interconnects. In the results presented here, FPGAs have outperformed GPUs, in the case of the BenOne. This can be attributed to the higher data transfer bandwidth of the BenOne card versus the G80 graphics card used. In concluding on the relative merits of FPGAs and GPUs for the implementation of closed-form expressions such as are presented here, one may wish to consider power consumption (significantly lower for FPGAs), compile time (significantly higher for FPGAs), cost (lower for GPUs), ability to connect to varied I/O (a strength of the FPGA) and context switch times (significantly higher for the FPGA). The functions implemented here exhibit high levels of data-level parallelism and suit the GPU well. FPGAs and the Cell Processor have functional parallelism advantages not addressed in this example.

9. Future Work

It would be useful to compare performances between FPGA, GPU, Cell and Multicore processors with respect to their power consumption, and the error of the results when compared to a benchmark implementation. FSB-FPGA results are to be added in due course. Future work might investigate improving the Cell results, with the possible use of alternate development environments.

10. Acknowledgements

The lead author's research is sponsored by Nallatech, and partially funded by the UK Engineering and Physical Sciences Research Council. The Institute for System Level Integration and Strathclyde University, both in Scotland, provide academic and logistical support. This work has been also supported by the Spanish government through the research contracts CICYT-TIN 2005/5619 and Ingenio 2010 Consolider CSD00C-07-20811

11. References

[1] G. Estrin, "Organization of Computer Systems—The Fixed Plus Variable Structure Computer," *Proc. Western Joint Computer Conf.*, New York, 1960, pp. 33-40.

[2] M. Butts, A.M. Jones, and P. Wasson, "A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing", *Proc. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, Napa, CA, USA, 23-25 April 2007, pp. 55-64

[3] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V.V. Kindratenko, and D. Buell, "The Promise of High-Performance Reconfigurable Computing", *Computer, Volume 41 , Issue 2*, IEEE Computer Society Press Los Alamitos, CA, USA, February 2008, pp. 69-76

[4] A. Cantle, "Is it Time for Von Neumann and Harvard to Retire?", *Reconfigurable Systems Summer Insitute, National Centre for Supercomputer Applications*, Urbana-Champaign, Illinois, USA, July 12th 2005

[5] C. Thiem, S. Drager, C. Flynn, D. Burns, and T. Renz, "Advanced Computer Technology for Novel Information Processing Paradigms", *Journal of Aerospace Computing, Information, and Communication 2004, vol.1 no.7*, American Institute of Aeronautics and Astronautics, pp. 308-317

[6] V.V. Kindratenko, C.P. Steffen, and R.J. Brunner, "Accelerating Scientific Applications with Reconfigurable Computing: Getting Started", *Computing in Science & Engineering, Volume 9, Issue 5*, Sept.-Oct. 2007, pp. 70-77

[7] V.V. Kindratenko, R.J. Brunner, and A.D. Myers, "Mitrion-C Application Development on SGI Altix 350/RC100", *Proc. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, Napa, CA, USA, 23-25 April 2007, pp. 239-250

[8] G. Genest, R. Chamberlain, and R. Bruce, "Programming an FPGA-Based Super Computer Using a C-to-VHDL Compiler: DIME-C", *Proc. Second NASA/ESA Conference on Adaptive Hardware and Systems*, Edinburgh, Scotland, 5-8 Aug. 2007, pp. 280-286

[9] B.M. Holland, J. Greco, I.A. Troxel, G. Barfield, V. Aggarwal, and A.D. George, "Compile- and Run-time Services for Distributed Heterogeneous Reconfigurable Computing", *Proc. of the 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA 2006),* Las Vegas, Nevada, USA, June 26-29, 2006, pp. 33-41

[10] D. Andrews, R. Sass, E. Anderson, J. Agron, W. Peck, J. Stevens, F. Baijot, E. Komp, “Achieving Programming Model Abstractions for Reconfigurable Computing”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 16, Issue 1, San Francisco, CA, USA, Jan. 2008, pp.34-44

[11] N. Moore, A. Conti, M. Leeser, and L.S. King, "Writing Portable Applications that Dynamically Bind at Run Time to Reconfigurable Hardware," *Proc. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007),* Napa, CA, USA, 23-25 April 2007, pp. 229-238

[12] N. Palaniswamy, “Intel QuickAssist Technology”, *Technology@Intel Magazine, Volume 4, Issue 10*, Intel Corporation, May 2007

[13] I. McCallum, “Intel QuickAssist Technology Accelerator Abstraction Layer (AAL)”, *unpublished,* 2007

[14] R Bruce, M Devlin, S Marshall, “An Elementary Transcendental Function Core Library for Reconfigurable Computing”, RSSI, Urbana-Champaign, July 17-20 2007

[15] Pieter Bellens, Josep M. Perez, Rosa M. Badia, Jesus Labarta, "CellSs: a Programming Model for the Cell BE Architecture," *sc*, p. 5, ACM/IEEE SC 2006 Conference (SC'06), 2006