Procedia Computer Science

Volume 29, 2014, Pages 95–101

ICCS 2014. 14th International Conference on Computational Science

# Comparison of High Level FPGA Hardware Design for Solving Tri-Diagonal Linear Systems

David J. Warne[1], Neil A. Kelson[2], and Ross F. Hayward[3]

[1] High Performance Computing and Research Support, Queensland University of Technology, Brisbane, Queensland, Australia
david.warne@qut.edu.au

[2] High Performance Computing and Research Support, Queensland University of Technology, Brisbane, Queensland, Australia
n.kelson@qut.edu.au

[3] School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane, Queensland, Australia
r.hayward@qut.edu.au

**Abstract**

Reconfigurable computing devices can increase the performance of compute intensive algorithms by implementing application specific co-processor architectures. The power cost for this performance gain is often an order of magnitude less than that of modern CPUs and GPUs. Exploiting the potential of reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs) is typically a complex and tedious hardware engineering task. Recently the major FPGA vendors (Altera, and Xilinx) have released their own high-level design tools, which have great potential for rapid development of FPGA based custom accelerators. In this paper, we will evaluate Altera's OpenCL Software Development Kit, and Xilinx's Vivado High Level Sythesis tool. These tools will be compared for their performance, logic utilisation, and ease of development for the test case of a tri-diagonal linear system solver.

*Keywords:* Reconfigurable Computing, Field Programmable Gate Arrays, High Level Synthesis, Open Computing Language, Numerical Linear Algebra, Tri-diagonal Linear Systems

## 1 Introduction

A huge portion of compute time in scientific applications is spent solving numerical linear algebra problems. Reconfigurable computing devices can be used to implement custom accelerators, and implementations of numerical linear algebra routines on reconfigurable hardware is an active area of research [7, 16, 13, 18].

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014


doi:10.1016/j.procs.2014.05.009

## 1.1 Reconfigurable Computing

The most common reconfigurable computing devices are Field-Programmable Gate Arrays (FPGAs) which are considered here. Theses are made up of configurable logic blocks (CLBs), along with other elements such as distributed block RAMs (BRAM), and digital signal processors (DSPs) all interconnected by a programmable interconnect fabric. CLBs consist of a number of look-up tables (LUTs), logic gates and multiplexers which can be configured to implement custom logic. The massively parallel architecture of the FPGA provides a platform for the development of low power and power efficient (i.e., FLOPS/Watt) application specific accelerators [14, 1] allowing the development of High Performance Computing (HPC) with a low power footprint. Power efficient HPC has become an important challenge of leading HPC systems in realising Exa-scale computing [12, 5, 9].

## 1.2 High Level FPGA Design

Traditional methods for development on FPGAs require a hardware architecture to be designed using a Hardware Description Language (HDL). This requires the developer to design data paths, control logic, interface with Intellectual Property (IP) hardware cores, and deal with timing constraints. This is a tedious and complex development process which can inhibit their adoption in many practical applications. Alternatively, high level design tools attempt to remove some of the tedium in FPGA development. Using these, a developer may instead specify an architecture's behaviour at the algorithmic level in some high level programming language. The Compilation process will then extract data paths, state-machines, process schedules etc., and automatically generate an HDL architecture specification.

Recently the two major FPGA vendors (Altera and Xilinx) have released their own high level development tools. Both vendor tools present some new possibilities for more rapid hardware/software co-design, with the potential to expose FPGAs to a wider community of both software and hardware developers.

Xilinx have released the Vivado High Level Synthesis (HLS) tool as part of their 7-series development software stack. This tool enables the development of IP cores using standard C code [6], which is subsequently mapped to an RTL architecture (either VHDL or Verilog). Internal C function code is synthesised into RTL statements which operated over a number of clock cycles. The function signature is mapped to RTL ports using standard protocols such as Buses, FIFOs, and RAM [17]. The use case for the Vivado HLS is the rapid development of IP cores that typically need to be integrated into a larger system design. As a result, the Vivado HLS is primarily targeted at improving development abilities of hardware engineers who have sufficient expertise in both software and hardware design [10].

Altera have released an OpenCL Software Development Kit (SDK) for certain supported Stratix V PCIe cards [1, 2, 3]. The Open Computing Language (OpenCL) is an open standard for parallel programming on heterogeneous compute systems [8] comprising of a device side parallel programming language, and a host side C Application Programming Interface (API). While OpenCL implementations primarily exist for CPU and GPU architectures, the Altera OpenCL SDK is an implementation of the OpenCL embedded profile for FPGAs [2].

Device side OpenCL compute kernel code is translated to a Verilog IP core which is then used to generate a FPGA configuration image. The Host side OpenCL API can then be used to configure and control the FPGA for data processing. In contrast with the Vivado HLS the OpenCL SDK permits the development of FPGA-based applications by software engineers with minimal hardware experience [1, 10].

With the advent of high-level design tools, there is opportunity to rapidly increase the rate

at which research can progress on reconfigurable hardware-based numerical linear algebra designs. In this work, a tri-diagonal linear system solver was selected as the target application for comparison of the current generation of high level FPGA design tools. This was selected due to it's requirement of floating-point arithmetic, it's simple implementation, and previous development we have performed in HDL [15]. The focus of this comparison will be logic utilisation, performance, and accuracy. This work is intended to be the initial steps in a more detailed comparison across a broader range of numerical linear algebra algorithms.

# 2 Tri-Diagonal Linear System Solver Designs

## 2.1 The Tri-Diagonal Matrix Algorithm

A tri-diagonal linear system has the form

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 &= b_1 \\ a_{i,(i-1)}x_{i-1} + a_{i,i}x_i + a_{i,(i+1)}x_{i+1} &= b_i,\ i \in [2, 3, \ldots, n-1] \\ a_{n,(n-1)}x_{n-1} + a_{n,n}x_n &= b_n. \end{aligned}$$

Such a system can be represented compactly as four arrays of $n$ elements,

$$\begin{aligned} \mathbf{U} &= \left[0, a_{1,2}, a_{1,3}, \ldots, a_{(n-1),n}\right] \\ \mathbf{D} &= \left[a_{1,1}, a_{2,2}, \ldots, a_{n,n}\right] \\ \mathbf{L} &= \left[a_{2,1}, a_{3,2}, \ldots, a_{n,(n-1)}, 0\right] \\ \mathbf{b} &= \left[b_1, b_2, \ldots, b_n\right] \end{aligned} \tag{1}$$

where **L**, **U**, and **D** are the lower, upper, and diagonal bands of the coefficient matrix and **b** represents the right hand side vector.

The LU-decomposition, forward substitution, and backward substitution of tri-diagonal systems reduce to $\Theta(n)$ operations. This results in the Tri-Diagonal Matrix Algorithm (TDMA) which can be expressed simply in terms of the vectors given in Equation (1). The LU-decomposition/forward substitution step is given as Algorithm 1 and the backward substitution step is given as Algorithm 2.

$\mathbf{L}(1) \leftarrow \mathbf{L}(1)/\mathbf{D}(1)$;
**for** $i \leftarrow 2$ **to** $n$ **do**
  $\mathbf{D}(i) \leftarrow \mathbf{D}(i) - \mathbf{L}(i-1) \times \mathbf{U}(i)$;
  $\mathbf{L}(i) \leftarrow \mathbf{L}(i)/\mathbf{D}(i)$;
  $\mathbf{b}(i) \leftarrow \mathbf{b}(i) - \mathbf{L}(i-1) \times \mathbf{b}(i-1)$;
**end**

**Algorithm 1:** The TDMA LU-decomposition/Forward Substitution

Note that Algorithms 1 and 2 are the usual *sequential* forms of the TDMA. Rather than exploiting parallelism by applying more complex methods such as cyclic reduction, we have chosen to work with the sequential TDMA with pipelining of factorisation and backward substitution steps [11]. This is primarily to enable comparison against a co-processor which we have developed previously [15].

$\mathbf{b}(n) \leftarrow \mathbf{b}(n)/\mathbf{D}(n)$;
**for** $i \leftarrow n-1$ **to** 1 **do**
  $\mathbf{b}(i) \leftarrow \mathbf{b}(i) - \mathbf{b}(i+1) \times \mathbf{U}(i+1)$;
  $\mathbf{b}(i) \leftarrow \mathbf{D}(i)$;
**end**

**Algorithm 2:** The TDMA Backward Substitution

## 2.2 Direct Hardware Design

We have previously presented a custom co-processor for solving tri-diagonal linear systems [15] using the hardware description language VHDL, and was implemented on a Xilinx Virtex-4 LX200 FPGA. This design was targeted to applications requiring the solutions to many independent tri-diagonal systems, such as cubic spline interpolation [11, 4]. Briefly, the VHDL design implemented a pipelined TDMA solver that enabled the LU-decomposition of the next system to begin without the need to wait for the backward substitution to complete. As a result, the number of cycles $c$ required to solve $M$ tri-diagonal systems with $N$ unknowns each was $c = N \times (1 + M)$.

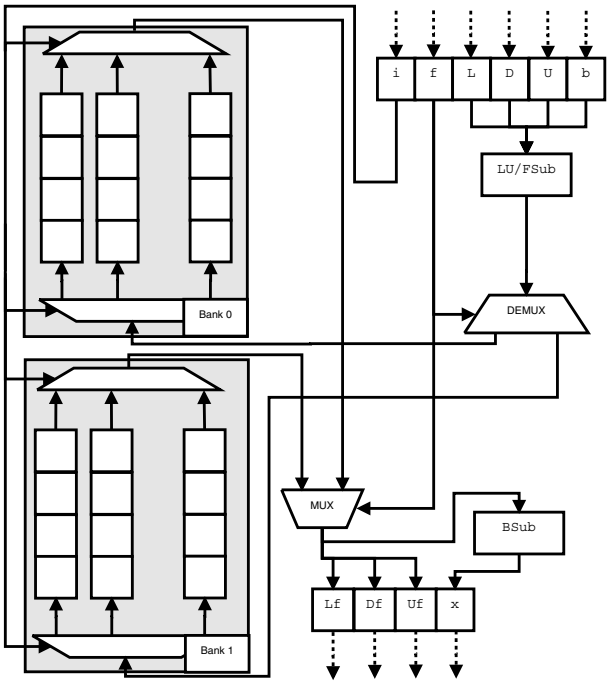


Figure 1: TDMA solver pipeline

A overview of our design is given in Figure 1. The flag `f` specifies which memory bank the TDMA forward loop writes to, consequently the backward substitution loop reads from the other memory bank. The interested reader should look to our previous paper for more details on this design [15]. This design will be the hardware benchmark we will compare the High level implementations against.

Table 1: Device Utilisation

| Logic Utilisation | VHDL | Altera OCL | Xilinx HLS |
|---|---|---|---|
| LUT | 211,297 | 22,255 | 3,765 |
| FF | 262,504 | 39,465 | 2,310 |
| DSP | - | 46 | 10 |
| BRAM (Kb) | - | 557 | 180 |

### 2.3 High Level Language Designs

In this section, we will describe the implementation of the TDMA solver using the High level development tools supplied by the leading FPGA vendors. Namely Xilinx's Vivado HLS, and Altera's OpenCL SDK.

Using Vivado HLS to generate an IP core is a straight forward task. This was achieved by directly implementing Algorithms 1 and 2 in ANSI C. The result of compilation is a TDMA RTL IP core. The C function input arguments are mapped to external RAM interfaces (though other options are possible such as FIFO's), and BRAM is utilised to store the factorised system before the backward substitution step.

For the Altera OpenCL version, Algorithms 1 and 2 were implemented as an OpenCL kernel function. In this case, the explicit use of OpenCL local memory buffers is required to enforce the compiler to generate a local cache of BRAM to store the factorised system, otherwise everything would be stored via on-board (off-chip) SRAM. The TDMA kernel function compiles to an RTL TDMA solver processor, and is then integrated into a larger OpenCL device top-block.

Both high level designs differ in one aspect to the direct VHDL implementation; that is, they are not double buffered. The Vivado HLS and Altera OpenCL versions will stall the forward loop while the backward substitution is being performed. As a result, performance times only take into account the solving of a single tri-diagonal linear system.

## 3 Comparison of Designs

In this section, we will compare the RTL code generated by Vivado HLS and Altera's OpenCL SDK against our direct VHDL implementation.

### 3.1 Logic Utilisation

The FPGA logic fabric utilisation is dependent on the maximum sized tri-diagonal linear system that is required to be solved. This is due to the dependency that the back substitution step has on the factorisation step. Table 1 shows the logic fabric utilisation of each design for a maximum tri-diagonal linear system size of 1024 unknowns. In all cases, the utilisation numbers are for the inner most TDMA solver unit.

Our VHDL implementation utilises significantly more LUTs and FFs (Flip-Flops). There are several reasons for this. Firstly, BRAM modules were intentionally not used, as it simplified the VHDL. Secondly, the VHDL design is double buffered, so two tri-diagonal systems are stored. Finally, custom floating point units were also designed and did not fit appropriately with the DSP blocks.

The Vivado HSL uses much less of the logic fabric than the Altera OpenCL. At this stage, we have not analysed the RTL code in detail, so the exact reasons for this are unclear. The

Table 2: Run-time Comparison and Solution Error

| | VHDL | Altera OCL[1] | Xilinx HLS |
|---|---|---|---|
| Run-time (secs) | 0.000671 | 0.002752 | 0.000465 |
| $\|\|\mathbf{x}-\mathbf{b}\|\|_\infty$ | 1.79e-04 | 6.60e-05 | 4.40e-05 |

interfacing of the OpenCL kernel compute unit to the complete OpenCL device top block probably contributes to this.

### 3.2 Performance and Accuracy

The time taken to solve a single 1024 equation tri-diagonal linear system was recorded for all three designs. Of all the designs, only the Altera OpenCL SDK design was actually executed on the board (The Bittware S5PH-Q PCIe board [3]). Both the VHDL design, and the Vivado HLS design could only be performed in simulation due to the extra development time required to design an appropriate top-block and host side communication functions.

The solution error was also computed. This error was against the solution vector $\mathbf{x}$ which was used to generate the tri-diagonal linear system. The error metric used was the infinity norm $||\mathbf{x}||_\infty = \max|x_i|$.

As shown in Table 2, the accuracy the Vivado HLS design achieves the best result. The custom floating point designs utilised in the VHDL design do not fully conform to IEEE–754 standard rounding, which is most likely for the reduced accuracy here. Altera also utilises a fused floating point design in which multiple floating point modules are chained together with the de-normalisation and normalisation steps occuring only at the start and end of the chain [1]; this can be a source of floating point error.

In terms of performance, the VHDL and Vivado HLS perform have roughly equivalent performance, with the Vivado HLS being slightly better. However it is worth noting that in our VHDL design, the double buffering allows multiple solves to be pipelined; this is not reflected in the numbers in Table 2 as only one system ins being solved. Also it should be emphasised that despite the Altera OpenCL design being slower according to Table 2, the entire OpenCL framework enabled a very quick design cycle to real execution on the device.

## 4 Conclusions

High level hardware design is opening FPGA-based high-performance reconfigurable computing to a wider research community. The purpose of this paper has been to evaluate the leading high level design tools, namely Xilinx's Vivado HLS and Altera's OpenCL SDK, against a design hand-coded in VHDL. The TDMA solver was select due to it being a common linear algebra routine, and the pre-existence of an in-house design from our prior work [15].

The Vivado HLS certainly seems superior in terms of logic utilisation, performance, and accuracy. Since we have not executed the Vivado HLS solution on real hardware, it may be that these results are artificially skewed, future work should focus a real execution comparison. The Altera OpenCL implementation allows a developer to design a complete system with host-processor communication, process scheduling, and data transfer all without looking at a single line of RTL code; this property is very attractive for rapid prototyping of a custom processor.

[1]The VHDL and Xilinx HLS results were simulated using iSim, whereas the Altera OpenCL results are taken from a real compute run of the Bittware S5PH-Q PCIe board.

Our results show that both these tools can produce RTL designs which are comparable in performance and logic utilisation with our VHDL design. By using these high level tools the development time is significantly reduced. These tools make it feasible to rapidly design high performance/low-power compute accelerators to speed up linear algebra processing for scientific applications.

# References


[1] Altera. Implementing fpga design with the opencl standard. Technical report, Altera, Inc., November 2012.

[2] Altera. Altera sdk for opencl programming guide. Technical report, Altera, Inc., June 2013.

[3] Bittware. S5ph-q user's guide. Technical report, Bittware, Inc., 2012.

[4] Biswa N. Datta. *Numerical Linear Algebra and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 2010.

[5] Yong Dou, Yuanwu Lei, Guiming Wu, Song Guo, Jie Zhuo, and Li Shen. Fpga accelerating double/quad-double high precision floating-point applications for the exascale computing. In *The Proceedings of the 24th ACM International Conference on Super Computing*, pages 325–336, 2010.

[6] Tom Feist. Vivado design suite. Technical report, Xilinx, Inc., June 2012.

[7] Juan Gonzalez and Rafael C. Nùñez. Lapackrc: Fast linear algebra kernels/solvers for fpga accelerators. *Journal of Physics: Conference Series*, 180(1):012042, 2009.

[8] Khronos OpenCL Working Group. The opencl specification version 1.0. Technical report, Khronos Group, October 2009.

[9] David W. Jensen and Arun F. Rodrigues. Embedded systems and exascale computing. *Computing in Science and Engineering*, 12(6):20–29, 2010.

[10] Kevin Morris. Hls versus opencl: Xilinx and altera square off on the future. `http://www.eejournal.com/archives/articles/20130312-highlevel/`, 2013. Accessed: 13-01-2014.

[11] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in FORTRAN 77:The art of Scientific Computation*. Cambridge University Press, Cambridge, England, 2nd edition, 1992.

[12] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In José M. L. M. Palma, Michel Daydé, Osni Marques, and João C. Lopes, editors, *High Performance Computing for Computational Science VECPAR 2010*, volume 6449 of *Lecture Notes in Computer Science*, pages 1–25. Springer Berlin Heidelberg, 2011.

[13] Sam Skalicky, Sonia Lopez, and Marcin Lukowiak. Performance modeling of pipelined linear algebra architectures on fpgas. *Computers and Electrical Engineering*, 2013.

[14] Prasanna Sundararajan. High performance reconfigurable computing. Technical report, Xilinx, Inc., 2010.

[15] David J. Warne, Neil A. Kelson, and Ross F. Hayward. Solving tri-diagonal linear systems using field programmable gate arrays. In *Proceedings of the 4th International Conference on Computational Methods*, 2012.

[16] Guiming Wu, Yong Dou, Junqing Sun, and Gregory D. Peterson. A high performance and memory efficient lu decomposer on fpgas. *IEEE Transactions on Computers*, 61:366–378, 2012.

[17] Xilinx. Vivado design suite user guide high-level synthesis. Technical report, Xilinx, Inc., December 2012.

[18] Ling Zhou and Viktor K. Prasanna. High-performance designs for linear algebra operations on reconfigurable hardware. *IEEE Transactions on Computers*, 57(8):1057–1071, 2008.