Accelerating Implicit Finite Difference Schemes Using a Hardware Optimized Tridiagonal Solver for FPGAs

Samuel Palmer
ucabsdp@ucl.ac.uk

Department of Computer Science, University College London, Gower Street, London, WC1E 6BT

Abstract—We present the design and implementation of the Thomas algorithm optimized for hardware acceleration on an FPGA. The hardware based algorithm combined with the custom data flow and low level parallelism available in an FPGA reduces the overall complexity from 8N down to 5N serial arithmetic operations and almost halves the overall latency by parallelizing the two costly divisions. Combining this with a data streaming interface, we reduce memory overheads to only 2 N-length vectors per N-tridiagonal system to be solved. The Thomas Core developed allows for multiple independent tridiagonal systems to be continuously solved in parallel, providing an efficient accelerator for many computations. Finally we present applications for derivatives pricing problems using implicit finite difference schemes on an FPGA accelerated system and investigate the use and limitations of fixed-point arithmetic in our algorithm.

I. INTRODUCTION

A. FPGAs

Field programmable gate arrays (FPGAs) provide an integrated circuit that can be reconfigured on the fly or ‘in the field’ in the form of a chip. FPGAs provides a flexible and cost effective way to develop and implement custom hardware designs. The core component that allows an FPGA to be reconfigurable is a look-up table (LUT). A LUT produces an output/outputs as a function on the digital inputs, these functions are determined when the FPGA is configured and provides the desired logic by the inputs controlled via the programmable cells. The other key component that helps to increase the performance of FPGAs are on-chip block memory (BRAM) which can provide fast local memory caches. Some FPGA chips may also offer other additional features such as high speed digital signal processors (DSP) and multipliers. The FPGA chip is then usually placed on a circuit board and connected to additional peripherals such as DDR memory, USB ports, ethernet, PCI express and VGA ports to provide the complete heterogeneous computing system.

B. Finite Difference Schemes and Tridiagonal Systems

Finite difference (FD) schemes are an important tool for solving parabolic partial differential equations (PDEs) numerically. In financial engineering FD methods are commonly employed to solve PDEs that are used to model derivatives, such as the famous Black-Scholes equation (BSE) [1].

Finite difference schemes begin by discretising the problem domain into a mesh/grid over the time interval [0, 1] and in basic cases, the asset price interval [0, Smax]. The domain is discretised into N asset price steps and M time steps, given by:

$$\Delta S = \frac{S_{max}}{N}$$

$$\Delta t = \frac{1}{M}$$

The spatial derivative terms are approximated using central difference and backward difference for the time derivative. These discretizations are then substituted into the PDE to produce the discrete difference equation, for example the BSE equation gives:

$$V^m_n = a_n V^m_{n-1} + b_n V^m_{n-1} + c_n V^m_{n+1}$$

with problem dependant stencil coefficient values $a_n$, $b_n$, $c_n$. This is the basic implicit scheme used for one dimensional problems, the resultant system of equations can be written in matrix form and needs to be solved for the price vector at the current time-step $V^{t-1}$, where the vector $V^t$ is known from the previous implicit step.

$$AV^{t-1} = V^t$$

or more generally written as the matrix inversion problem $Ax = y$, where the coefficient matrix $A$ takes on the banded tridiagonal form shown below:

$$A = \begin{bmatrix}
b_0 & c_0 & 0 & 0 & 0 & \cdots \\
a_1 & b_1 & c_1 & 0 & 0 & \cdots \\
0 & a_2 & b_2 & c_2 & 0 & \cdots \\
0 & 0 & a_3 & b_3 & c_3 & 0 & \cdots \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \cdots & 0 & a_N & b_N &
\end{bmatrix}$$

Further to this when introducing numerical schemes for pricing multidimensional derivatives, such as basket options...
or under stochastic volatility, another class of finite difference schemes known as alternating-direction-implicit schemes [2] may be used. These schemes solve the PDE in an implicit manner within multiple dimensions. These methods can be computational challenging as they require the solution to multiple tridiagonal systems at each time step, thus a lot of effort has gone into creating fast parallel solvers on devices such as GPUs [3] [4].

C. Thomas Algorithm

**Algorithm 1** Thomas Algorithm (a,b,c,y) Pseudo Code

```plaintext
d[0] = b[0]
z[0] = y[0]
for i = 1 to N do
  prev = i - 1
  li = a[i]/d[prev]
  d[i] = b[i] - li*c[prev]
  z[i] = y[i] - li*z[prev]
end for
z[N] = z[N]/d[N]
for i = N-1 to 0 do
  x[i] = (z[i] - c[i]*x[i+1])/d[i]
end for
return x[i]
```

The Thomas algorithm [5] is the simplest method used to solve a tridiagonal system of equations and is commonly employed on serial devices such as a CPU. The Thomas algorithm is a specialised case of Gaussian elimination and can be derived from the LU decomposition of the matrix A. This reduces the system down to the solution of two bi-diagonal systems which can then be solved via Gaussian elimination. The first system is solved via forward substitution and the second system is solved via backward substitution. These two stages will be referred to as the forwards and backwards iterations. The Thomas algorithm is given in algorithm 1, it has a complexity of $O(N)$ and requires a total of $8N$ arithmetic operations to solve an $N$-tridiagonal system.

In parallel computing the Thomas algorithm is usually less favoured than algorithms such as recursive-doubling [6], cyclic-reduction [7] and parallel cyclic-reduction [8], since although these algorithms have a larger number of arithmetic operations some of the operations be can parallelised on devices such as GPUs [9] resulting in an overall lower algorithmic complexity. With a recent increased interest in FPGA acceleration attempts have been made to port tridiagonal solvers onto FPGAs [10][11][12][13]; in this application the simplicity of the Thomas algorithm makes it well suited to the task when compared to cyclic-reduction which maybe too complex for efficient data flow FPGA implementation.

II. ALGORITHMIC OPTIMIZATION AND LOW LEVEL PARALLELIZATION

Figures 1 and 2 depict the data dependency of the Thomas algorithm; it can be observed that in the forward iteration there are two separate branches of computation, one for calculating $d_n$ and the other for $z_n$ and hence this provides the first level of parallelism extracted. A similar approach has been taken by both Oliveira et al [10] and Warne et al [11]. This optimisation reduces the effective serial arithmetic operations down from $8N$ to $6N$.

The problem with this simple optimisation is that although there is a reduction in serial operations, it has only reduced a multiply and a subtract which are computational cheap when compared to divisions. Consequently a competitive speed-up over faster clocking devices such as CPUs may not be obtained [11]. We thus introduce a simple algorithmic rearrangement that can allow for the two divisions from the backwards and forward iterations to be effectively parallelised. Equation 6 shows the factorisation of the backwards iteration calculation where we now treat the divisions of $z_n$ and $c_n$ by $d_n$ individually.

$$\frac{z_n - c_n V_{n+1}}{d_n} = \frac{1}{d_n} z_n - \frac{1}{d_n} c_n V_{n+1}$$

(6)

In a serial implementation this would add an extra division to the total number of arithmetic operations, which is not usually desirable, but as shown in figure 3 the data dependence is in fact here reduced, we can now treat these two divisions in
parallel with each other whilst also performing them in parallel with the forward iteration calculations. This reduces the serial arithmetic operations down from the original 8N to 5N.

For FPGA implementations this rearranged algorithm has two advantages:

1) Total latency of the algorithm is almost halved by parallelising the two lengthy divisions.
2) Memory requirements are reduced from the need to save three intermediate data vectors (c, z and d) to two (c/d and z/d).

A. Pipelining

Further to the low level algorithmic optimisations higher level parallelism can be achieved in two ways: pipelining the data through the computation of the forward and backward iterations; and pipelining the sets of data between the forward and backward iterations, which has commonly been implemented for multiple CPU versions to parallelise the Thomas algorithm [14].

Firstly the computational units themselves can be deeply pipelined, an approach used by Olivera et al [10], which allows for multiple independent tridiagonal systems to be computed in the same iteration cycle. For example, if the forward iteration computational unit has $P_F$ pipeline stages then throughout one iteration it is possible to fill each stage of the pipeline with a computation allowing for $P_F$ independent tridiagonal systems to be computed.

The second type of pipelining of the algorithm means that given a set $T$ of pipelined tridiagonal systems for either iteration, as discussed above, we can simultaneously compute the forward and backwards iterations of the two different sets (given the first set has already been through the forward iterations) independently in one Thomas solver. In [12] they look at using OpenCL and Xilinx HLS to build the Thomas solvers, but do not obtain this level of parallelism due to the complex scheduling involved for the pipelines. As such this design has been made in VHDL allowing this desired low level control to be obtained.

B. Hardware Architecture

The input to the solver core consists of 5 data items $a, b, y$ and $id$ where $id$ is the local id of the system to be solved, this acts as a thread id and is important for addressing the correct memory stacks in the solver. The hardware architecture consists of four main components, the forward iteration core, the $d$-divider, the backwards iteration core and the stack array. The forward and backward cores contain the pipelined arithmetic for the stages of the algorithm, and the $d$-divider consists of two dividers to carry out the $c/d$ and $z/d$ computations. The stack array is used for storing the intermediate variables $c/d$ and $z/d$, a stack can be used due to the nature of the problem which is that the backwards iterations first require the last values calculated by the $d$-divider, this saves unnecessary complications with memory addressing.

Connecting the forward iterations to the backwards iterations is a queue. This queue allows the problem index to be passed onto the backwards core for computation once the forward iterations have finished, this system allows for efficient independent operation of the forwards and backward iterations. The backwards core checks for space in the pipeline, and reads in the problem to begin computing if there is space, otherwise it remains queued.

In addition to the main Thomas algorithm core, the core has been placed in a wrapper allowing for easy usability. The wrapper consists of fifo queues for the input data and output results, allowing for variable write and read times to and from the core, as well as the option for floating-point to fixed-point converters for input data and vice-versa for results.

When changing the arithmetic only the arithmetic cores are changed, and the architecture remains constant. The only variability with the arithmetic cores is the pipelining due to the differing latencies, but this is managed via adjustable parameters within the solver VHDL.

III. Design Analysis

Here we theoretically analyse the performance of the solver for solving multiple independent tridiagonal systems $T = \{T_1^{N_1}, T_2^{N_2}, \ldots, T_M^{N_m}\}$, where $M$ is the total number of independent tridiagonal systems to be solved and $N_m$ is the size of the $m$th system. The notation is used in this work is as follows:

- $T_m^{N_m}$ is the size of the $m$th tridiagonal system to be solved of size $N_m$.
- $T^N$ is a special case where for all $T_m \in T$, $N_m = N$.
- $C_D^{+, -, /, \times}$ is the number of clock cycles taken for that arithmetic operation using data format $D$. 
• $C_{F,B,A}$ is the number of clock cycles taken for a single forwards, $F$, and backwards, $B$, iteration and administration costs $A$.

• $f$ is the clock frequency of the FPGA system.

The number of cycles taken for the iteration stages are:

\[ C_F = C_P^0 + C_P^D + C_X \]  \hspace{1cm} (7)

\[ C_B = C_P^0 + C_P^+ + C_P^D \]  \hspace{1cm} (8)

with $C_A$ being a constant determined by the programming of the algorithm.

To fully harness the power of the pipelined design it is desired that maximal throughput should be achieved by scheduling groups of independent computations.

**Definition 1.** The number of computational blocks, $B$, is defined as the number of subsets of independent tridiagonal systems to be solved. The set of of blocks given by $B = \{b_1^{m_1}, b_2^{m_2}, \ldots, b_B^{m_B}\}$, where $b_i^{m_i} \subset T$ of size $m_i$ such that:

\[ \cup_{i=1}^B b_i^{m_i} = T \quad \cap_{i=1}^B b_i^{m_i} = \emptyset \]  \hspace{1cm} (9)

Thus for a given $M$ the time to compute $T^N$ is then given by:

\[ t_{F\times N} = \frac{NB(C_F + C_A) + BC_I + NC_B + 2 \sum_{b=1}^B (m_b - 1)}{f} \]  \hspace{1cm} (10)

The partitioning of $T$ into the set of blocks $B$ can be effected by the data transfer rate $r_d$ between the solver and the host system. The rate of computation, $r_c$, of the Thomas solver is given by:

\[ r_c = \frac{5D}{f} \]  \hspace{1cm} (11)

where $D$ is the number of bits used to represent a number in the given format. This value is the rate at which data can be processed by the Thomas solver, the solver requires 5 inputs, $a, b, c, y$ and $id$ and can process a row every clock cycle.

The optimal number of blocks $B$ can be obtained if the rate of transfer is quicker than the rate of computation, i.e. the solver can receive all the 5 values in one clock cycle or less:

\[ B_{opt} = \text{floor}(\frac{M}{C_P}) \quad ; \quad r_d \geq r_c \]  \hspace{1cm} (12)

It maybe the case that the data transfer rate is slower than the rate of computation and hence the solver has to be stalled whilst waiting for the data. It is therefore desirable to compute the maximum number of tridiagonal systems in a block $b$ in the pipeline without stalling for data. The number of blocks $B$ is given by:

\[ m_{opt} = \text{ceil}(\frac{r_c}{r_d}) \quad ; \quad r_d < r_c \]  \hspace{1cm} (13)

\[ B = \text{floor}(\frac{M}{m_{opt}}) \]  \hspace{1cm} (14)

Maximum throughput for the solver can be obtained if the set of tridiagonal systems to be solved completely fills the pipeline:

\[ M\%C_F = 0 \]  \hspace{1cm} (15)

\[ B > 1 \]  \hspace{1cm} (16)

**IV. FIXED-POINT ARITHMETIC ANALYSIS**

**A. Numerical Bounds**

To maximise performance it maybe required that custom data formats are used in the FPGA design. Fixed-point arithmetic often provides faster and smaller FPGA designs, for example [15][16], but at the cost of the loss of some precision in the results and a higher risk arithmetic overflow. Therefore it is important to know the range of values the solver is expected to use in the algorithm to allow for the custom data formats to be optimised for the problem. The preceding theorems presented require additional conditions that the data formats to be optimised for the problem. The preceding theorems presented require additional conditions that the data formats to be optimised for the problem. The preceding theorems presented require additional conditions that the data formats to be optimised for the problem. The preceding theorems presented require additional conditions that the data formats to be optimised for the problem.

**Theorem 1.** Given $A$ is diagonally dominant by row or columns, and let $A$ have LU factorisation $A = LU$. Then $\|d\|_\infty \leq 3\|b\|_\infty$

**Proof:** Given in the proof of $|L||U| \leq 3|A|$ found in [17] pg.175 the following result is used:

\[ |l_n c_{n-1}| + |d_n| \leq 3|b_n| \]  \hspace{1cm} (17)

Simple rearrangement and the observation that the max will occur at the maximum absolute value gives the result of theorem 1

**Theorem 2.** Given $A$ is diagonal dominant by row, and let $A$ have LU factorisation $A = LU$ then $|d_n| > |b_0 - \frac{a_1}{b_0}||c||_\infty| \forall n$, given that $b(n)$ is a positive monotonically increasing function of the row index, $i$ and $\Delta b \leq ||c||_\infty$.

**Proof:**

\[ d_0 = b_0 \]  \hspace{1cm} (18)

\[ d_1 = b_1 - \frac{a_1}{b_0}c_0 \]  \hspace{1cm} (19)

\[ b_1 - \frac{a_1}{b_0}||c||_\infty \leq d_1 \]  \hspace{1cm} (20)

Under the assumption that $b(n)$ is a positive monotonically increasing function then

\[ b_0 - \frac{a_1}{b_0}||c||_\infty \leq b_1 - \frac{a_1}{b_0}||c||_\infty \leq d_1 \]  \hspace{1cm} (21)

finally for this to hold over all cases it must enforced that $\|l\|_\infty \leq \frac{||a||_\infty}{|b_0|}$, which implies that:

\[ b_0 \leq b_1 - \frac{||a||_\infty}{|b_0|}||c||_\infty \]  \hspace{1cm} (22)
thus for this hold $\Delta b \leq \|c\|_{\infty}$, given that $\|a\|_{\infty} < |b_0|$.

A more general theorem for all functions of $b(n)$ will be investigated, though theorem 2 currently suffices for this work as will be seen later. The approach for finding a more general theorem will look at conditions for when a certain sequence of $b$ provides a minimum for $d_0$ when compared to all other possible permutations.

**Theorem 3.** Given $A$ is diagonal dominant by row, and let $A$ have LU factorisation $A = LU$ then $\|l\|_{\infty} < \frac{\|a\|_{\infty}}{|b_0|} |c|_{\infty} \leq \frac{\|a\|_{\infty}}{|b_0|}$, given that $b(n)$ is a positive monotonically increasing function of the row index, $n$, and $\Delta b \leq \|c\|_{\infty}$.

Proof: Using theorem 2, the maximum value of $l$ must be achieved when the largest value of $a$ is divided by the smallest value of $d$.

In fact, although theorem 1 provides an upper bound for the value of $d$, using the previous theorems a tighter more accurate bound can now be defined.

**Theorem 4.** Given $A$ is diagonal dominant by row, and let $A$ have LU factorisation $A = LU$ then $d \leq \|b\|_{\infty} + \|l\|_{\infty} \|c\|_{\infty}$, given that $b(n)$ is a monotonically increasing function of the row index, $n$.

1) Bounding the Thomas Algorithm:

The first section describes various bounds for the LU decomposition of a matrix $A$. This forms the basis of the well known Thomas Algorithm used for solving tridiagonal inversion problems of the form $Tx = y$, where $T$ is a tridiagonal matrix. The first stage of the algorithm is to apply LU decomposition to the matrix and then solve to auxiliary equations using forwards and backwards substitution.

**Theorem 5.** Given a tridiagonal matrix $T$ is diagonally dominant by row and let $T$ have LU factorisation $T = LU$ with $\|l\|_{\infty} < 1$, then solving the first auxiliary equation of the inversion problem $Lz = y$ ; $z = U \cdot x$, then $\|r\|_{\infty} < \|y\|_{\infty} \left(1 - \frac{1}{\|l\|_{\infty}}\right)$

Proof: First the term for $z_N$ is expanded and using theorem 3 it is possible to replace the individual $l_i$ terms with the upper bound $\|l\|_{\infty}$.

$$\|r\|_{\infty} \leq \|y\|_N + \sum_{k=1}^{N-1} \|l\|_{\infty}^k |y_k|$$ (23)

It is then possible to compact the telescopic sum into a geometric sequence, which has a maximum value when $i = N$ i.e using all of the terms. Given that the index of the the largest $y$ value may not be known a larger bound can be formed by including this in the geometric sum as the final term.

In fact we can further loosen the bound by assuming that all values are the max, so

$$\|z\|_{\infty} \leq \|y\|_N + \sum_{k=1}^{N-1} \|l\|_{\infty}^k |y_k| \leq \|y\|_{\infty} \left(1 + \sum_{k=1}^{N-1} \|l\|_{\infty}^k\right)$$ (24)

using the formulas for the sum of a geometric sequence the max bound becomes

$$\|z\|_{\infty} \leq \|y\|_{\infty} \frac{1 + \|l\|_{\infty}^N(1 - \|l\|_{\infty}^{N-1})}{1 - \|l\|_{\infty}}$$ (25)

and finally in the case that $\|l\|_{\infty} < 1$ a simpler form using the infinite geometric sum can be used:

$$\|y\|_{\infty} \frac{1 + \|l\|_{\infty}(1 - \|l\|_{\infty}^{N-1})}{1 - \|l\|_{\infty}} < \|y\|_{\infty} \left(1 - \|l\|_{\infty}\right)$$ (26)

The hardware optimised algorithm presented here requires the calculation of two additional intermediates $\frac{z}{a}$ and $\frac{z}{a}$.

**Theorem 6.** Given a tridiagonal matrix $T$ is diagonally dominant by row and let $T$ have LU factorisation $T = LU$ with $\|l\|_{\infty} < 1$, and previously stated conditions the intermediate value $\frac{z}{a}$ $\leq \frac{\|z\|_{\infty}}{|b_0|}$.

**Theorem 7.** Given a tridiagonal matrix $T$ is diagonally dominant by row and let $T$ have LU factorisation $T = LU$ with $\|l\|_{\infty} < 1$ the intermediate value $\frac{z}{a}$ $\leq \frac{\|z\|_{\infty}}{|b_0|}$.

Finally it is possible to then derive the bounds for the final values.

**Theorem 8.** Given a tridiagonal matrix $T$ is diagonally dominant by row and let $T$ have LU factorisation $T = LU$ with $\|l\|_{\infty} < 1$ and previously stated conditions with $b_n > 1$ and $c_n < 1 \forall n \leq N$ then $\|v\|_{\infty} < \frac{\|z\|_{\infty}}{|b_0|}$.

Proof:

$$v_N = \frac{z}{a}$$ (27)

$$|v_N| < \frac{\|z\|_{\infty}}{|b_0|}$$ (28)

the recursion begins at $v_{N-1}$, it is possible to see that

$$|v_{N-1}| < \frac{\|z\|_{\infty}}{|b_0|} + V_N$$ (29)

$$\frac{|v_{N-1}|}{|b_0|} < \frac{\|z\|_{\infty}}{|b_0|} + \frac{\|v\|_{\infty}}{|b_0|}$$ (30)

expanding the recursion in this manner the result for a sequence given by

$$|v_n| < \sum_{i=0}^{N-n} \frac{\|v\|_{\infty} \|z\|_{\infty}}{|b_0|^{i+1}}$$ (31)
then assuming $|b_0| > 1$ and $\|c\|_{\infty} < 1$ the limit of this sequence is given by:

$$\|v_n\| < \sum_{i=0}^{N-n} \left\| \frac{1}{|b_0|^{i+1}} \right\| \|z\|_{\infty} < \|z\|_{\infty}$$

(32)

Combining the previous theorems it is now possible to define a set of conditions that can ensure the absolute value of any variable in the algorithm does not exceed a certain bound. This will prove extremely useful when applying the fixed-point designs to given problems.

**Proposition 1.** For a given integer $Z ; 0 < Z$ there exists a set of conditions such that all intermediate variables in the Thomas algorithm can be bounded by $Z$, given that $b(n)$ is a positive monotonically increasing function of the row index, and $|a_n| < 1$, $|c_n| < 1$ and $|b_n| > 1 \forall n \leq N$. The following conditions are sufficient but not necessary:

1. $\|t\|_{\infty} < 1$ this implies $\|a\|_{\infty} < |b_0|
2. $\|y\|_{\infty} < Z |b_0| - \|t\|_{\infty}$$
3. $\|c\|_{\infty} < |b_0|
4. $\Delta b \leq \|c\|_{\infty}$

### V. HARDWARE IMPLEMENTATION

#### A. FPGA Resource Usage

The Thomas Solver hardware will be tested using the ZedBoard Xilinx Zynq7020 Evaluation Kit. The Zynq7020 is a system-on-chip which consists of two ARM-A9 processors connected to Xilinx Artix-7 FPGA fabric, allowing a high-speed interface between CPU and FPGA. Using the Zynq7020 the system of equations will be formulated in floating-point on the ARM-A9 CPU, these will then be transferred to the FPGA via AXI interfaces and solved using the FPGA Thomas solver. The fixed-point results are then converted back to floating-point and compared to the results for the same problem solved using floating-point arithmetic.

The results in table V-A were obtained post-implementation from the Vivado Design Suite, the base design used has $N_{\text{max}} = 512$ with 10 threads ($M_{\text{max}} = 10$), and variable arithmetic. A floating point design and three fixed-point solvers with the following data representation, [integer bits, fractional bits], are tested, 32bit[2,30], 24bit[2,22], 16bit[2,14]. For the arithmetic cores the provided Xilinx base IP cores were used, and set to make maximum usage of DSPs, and the Radix-2 divider algorithm is used for as part of the fixed-point divider.

Each of the solver designs have the same magnitude of latency, with the floating-point design providing the lowest total latency, although the fixed-point designs maybe sped up by using higher radix divider algorithms. The disadvantage of the higher radix divider algorithms is that the maximum throughput is reduced due to the iterative nature of the algorithms, but this is useful if it is not possible to achieve maximum throughput of processing on tridiagonal system per clock. The main advantage of the fixed point solvers is the reduced resource usage, which provides the opportunity to maximise coarse grain parallelism by allowing more solver cores to fit onto a device and also increasing the maximum number of pipelined tridiagonal systems each core can solve. As can be expected the amount of memory resources is proportional to the total data width used for the fixed-point designs, whilst the floating-point solver, although 32bits wide, uses significantly more memory resources (BRAM and memory LUTs).

#### B. Performance

The latency performance of the solver can be evaluated using equation 10 once the implemented FPGA clock speed is known. The floating-point was only able to achieve a maximum clock frequency of 100MHz whilst the fixed-point designs were able to achieve double this at 200MHz. Therefore although the fixed-point designs may have slightly higher latency in terms of clock cycles, the speed of computation is considerably faster due to this higher clock rate.

The average time in milliseconds per tridiagonal system is shown in table V-B for minimum throughput, a single tridiagonal system, and maximum throughput, where the pipeline is completely full. These results are compared to the average time taken for a 2.6GHz on a top of the range desktop machine. If the solver was to be used for single tridiagonal systems the speed is fractionally less than a top of the range 2.6GHz CPU, but this is not taking advantage of the pipelined design. At maximum throughput it is possible to achieve up to a 36x

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Div</td>
<td>0.0008ms</td>
<td>0.0011ms</td>
<td>0.0015ms</td>
<td>0.0018ms</td>
</tr>
<tr>
<td>Multiplier</td>
<td>0.0022ms</td>
<td>0.0029ms</td>
<td>0.0033ms</td>
<td>0.0039ms</td>
</tr>
<tr>
<td>Subtractor</td>
<td>0.0025ms</td>
<td>0.0036ms</td>
<td>0.0042ms</td>
<td>0.0047ms</td>
</tr>
<tr>
<td>Thomas Forward</td>
<td>0.0030ms</td>
<td>0.0036ms</td>
<td>0.0041ms</td>
<td>0.0046ms</td>
</tr>
<tr>
<td>Thomas Backward</td>
<td>0.0031ms</td>
<td>0.0037ms</td>
<td>0.0041ms</td>
<td>0.0046ms</td>
</tr>
<tr>
<td>Administration</td>
<td>0.0033ms</td>
<td>0.0039ms</td>
<td>0.0044ms</td>
<td>0.0049ms</td>
</tr>
</tbody>
</table>

### TABLE III

<table>
<thead>
<tr>
<th>Component</th>
<th>Max Throughput</th>
<th>Min Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU(2.6GHz)</td>
<td>0.0020ms(1x)</td>
<td>0.0020ms(1x)</td>
</tr>
<tr>
<td>Floating</td>
<td>0.0012ms(16x)</td>
<td>0.0063ms(31x)</td>
</tr>
<tr>
<td>Fixed[2,30]</td>
<td>0.00055ms(36x)</td>
<td>0.040ms(50x)</td>
</tr>
<tr>
<td>Fixed[2,22]</td>
<td>0.00055ms(36x)</td>
<td>0.036ms(55x)</td>
</tr>
<tr>
<td>Fixed[2,14]</td>
<td>0.00057ms(35x)</td>
<td>0.028ms(72x)</td>
</tr>
</tbody>
</table>
Given a Black-Scholes Implicit pricing problem, expressed in theorem 9. meet condition 2 of proposition 1. This is more formally y values and an appropriate transformation for the after applying and basic grid constraint to bound coefficient A, i.e. all values calculated in the algorithm, sup |A_r| < Z. This constraint in equation 39 in the case of maximum market parameter values, this resulted in dt = 0.001. Finally to meet the final for ||y|| a linear transform constant of \( \frac{6.45Z}{\|y\|_\infty} \) was used, calculated using equation 41, to meet this condition the problem was chosen so that \( S_N = 2 \) and \( K = 1 \).

Table VI-B gives the expected rounding error for the number of fractional bits used in the fixed-point design. The expected rounding error \( e_{\text{rnd}}(x, f) \), where \( e_{\text{rnd}} \) is the rounding error and \( f \) is the number of fractional bits, for rounding a floating-point number \( x \) to a fixed-point representation is given by:

\[
E(e_{\text{rnd}}(x, f)) = \frac{2^{f-1}}{2} ; x \in U[0, \infty]
\]  

It has been assumed that the rounding error is uniformly distributed white noise over \( x \in \mathbb{R} \) [18]. If an error obtained is smaller than this value indicates that the fixed-point value was rounded to 0, and the actual value is smaller than is possible to represent in the fixed-point representation. Errors within a similar magnitude as the magnitude of the expected error indicate that the fixed-point result is on average as accurate as is possible for the given fixed-point representation.

Figure VI-B shows the absolute error with respect to the floating point result for the fixed-point solver using 30 fractional bits. The most striking feature of this plot is how the error resembles the shape of the payoff function indicating that the magnitude of the option price plays a role in the error function. A worst case error function has been derived from the observation that an option price, \( V \), \( V < S_n \), i.e. the European option price must be at least less than the asset price due the the effect of the strike. The maximum error is then a function of the asset price, minimum expected rounding error and \( n \) to take into account error prorogation factors through the iterations.

\[
E(S_n) = nS_n \frac{2^{f-1}}{2}
\]  

Figures VI-B and VI-B show the absolute errors for the fixed-point solver with 22 and 14 fractional bits respectively. Unlike the 30 fractional bits both of sets of errors show a shape differing from the one observed for 30 fractional bits, with a peak near the strike price and then decending again. Although they both show and different shape to the 30 fractional bits.
their respective absolute errors with respect to their minimum fractional resolution is a lot better, with the largest magnitude of error being of the same order, this is up to 100 times smaller in magnitude than the error predicted by equation 43.

These results show that the the fixed-point arithmetic is accurate up to a given decimal place, which then after the accuracy begins to degrade. This explains why the 30 fractional bit errors were a lot larger than its respective minimum fractional resolution and not so for the 22 and 14 fractional bits.

VII. CONCLUSIONS AND FUTURE RESEARCH

This work has proposed and introduced a prototype design for a high performance FPGA based tridiagonal solver. Fixed-point designs can be used to minimise resource usage and obtain higher clock rates compared to floating point designs. When compared to a 2.6GHz CPU on a top of the range desktop it was possible to achieve up to a 36x speed-up and 16x speed-up for the fixed-point and floating-point designs respectively. For the fixed-point designs the errors introduced in the results due to the limited fractional resolution was investigated. Overall in the implicit option pricing example the errors were well behaved with the maximum for the 22 and 14 bit fractional representations only being 10x that of the expected rounding error, and 50x for 30 fractional bits. It is intended that this work is further integrated into a larger FPGA based implicit pricing system to achieve a high speed and low cost solution for accelerating options pricing. Future work will look at improving the accuracy whilst trying to retain high clock rates and low FPGA resource usage by using mixed-precision architectures; for example computing the forward iterations in 16bit fixed-point and the backwards in 32bit floating-point. Following up from the theoretical analysis further work with affine arithmetic techniques [19] it is possible to develop a deeper understanding of the fixed-point error behaviour and propagation in the algorithm. Advancing from this work it would be enlightening to also investigate the performance of more inherently parallel algorithms such as parallel cyclic reduction on an FPGA.

ACKNOWLEDGEMENTS

I would like to thank David Thomas (Imperial College London) for his help and support in this work.

REFERENCES


