ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume 27, Number 2, 2024, 123–136

FFT on a Heterogeneous System with a General-Purpose Map-Scan Accelerator

Mihai ANTONESCU[1, *] and Mihaela MALIŢA[2]

[1]University Politehnica of Bucharest, Romania, Faculty of Electronics, Telecommunications and Information Technology
[2]Rivier University, Nashua, NH, USA
Email: mihai.antonescu@upb.ro*,mmalita@rivier.edu
* Corresponding author

Abstract. This paper presents a use-case and evaluation for a custom general-purpose Map-Scan Accelerator using multiple Fast Fourier Transform (FFT) implementations. FFT computing is part of the "Spectral Methods" class of problems and qualifies to be accelerated because it is computationally intensive, of average complexity and widely used as one of the backbone algorithms in modern computing. In order to achieve the desired throughput or latency, different parameters are used for testing, regarding both the accelerator hardware and the FFT. Evaluation is done in simulation, targeting FPGA implementation. FFT is shown to work efficiently on our architecture, either latency or throughput (and in some circumstances, both by a smaller amount) can be improved linearly or even supralinearly compared to a single-core solution. Accelerations obtained vary between $\sim 0.5 \times p$ and $\sim 1.5 \times p$ where p is the number of accelerator cores.

Key-words: FFT; general-purpose accelerator; heterogenous computing; map-scan accelerator; parallelism.

1. Introduction

For improving computational performance, many studies have been made in the direction of developing accelerator architectures. Most of these accelerators are task-oriented and as such, do not translate well to other domains. The novelty of our research consists of developing a general-purpose accelerator designed to perform well in most classes of parallel applications in regard to either latency or throughput. In order to partially prove this statement, this paper presents the implementation and evaluation of the Fast Fourier Transform (FFT). This is done as part of a larger series of tests that aims to prove the general-purpose nature of our architecture

and evaluate its performance on a wide array of computational problems. Although the FFT's complexity class is only $n \times \log n$, it was chosen to evaluate the efficiency of our architecture as it is a frequently used function, it has a wide spectrum of applications and is representative of Spectral Methods class of problems. Even if its complexity class is relatively low, given the importance of this algorithm, we consider its acceleration to be an important area of research. Furthermore, our claim is that our general-purpose accelerator can be used to obtain at least linear acceleration for a wide array of applications, the FFT being one of them. Evaluation is done in simulation, targeting FPGA implementation, with code written in the accelerator's custom assembly language.

The general-purpose nature of the proposed accelerator architecture comes from the theoretical foundation found in Stephen Kleene's mathematical model [1] and has, as of yet, shown good results on different classes of computational problems, such as: dense linear algebra [2], sparse linear algebra [3], molecular dynamics [4] and others. This architecture is described in Section 2. and further detailed in [5–8]. This accelerator is designed to be part of a heterogeneous system in which it performs the intense parts of the computation.

Modern accelerator landscape is dominated by the following off-the-shelf solutions: multi-cores, GPUs, VPUs, DSPs, or circuits deployed on FPGAs. All these acceleration methods represent particular solutions selected according to the application in question. General-purpose accelerated computation is usually done on Graphics Processing Units that, with the exception of a few applications, fail to fully utilize their large number of processing cores. When discussing FFT implementations, while currently available hardware solutions each have their advantages, each implementation also suffers from some limitations: CPUs are fast, but not scalable and not inherently parallel, GPUs use many computational cores, but are not particularly well suited for the memory intensive parts of the FFT algorithm. DSPs and hardware implemented FPGA accelerators for FFT are application specific and thus offer high performance for the task at hand, but are not general-purpose computational cores to be efficiently used for other types of computation. Regarding FPGA computation, it should be noted that while the FPGA itself is reconfigurable and thus by definition general-purpose, the accelerators implemented on it are application specific and these accelerator architectures are considered when comparing against our general-purpose approach.

With the exception of MIC (Intel's Many Integrated Cores), which has been taken out of production, all other solutions are strictly dedicated or diverted solutions intended for other fields of applications. Most currently used FFT acceleration solutions are GPUs. This approach has established itself as an off-the-shelf solution, but with hundreds or even thousands of execution units it fails to produce justifiable accelerations compared to the immense computational resources it possesses. We consider that this comes from certain architectural inadequacies of a system optimized for graphic applications but which is used for a completely different domain.

In literature, the performance of GPGPU parallel accelerators is compared to CPU solutions. $3-5\times$ accelerations are reported in the field of video applications [9]. Experimental results presented in [10] show that GPU implementation of FFT may achieve 30 times the acceleration with respect to the CPU FFT implementation but only for large samples sizes. [11] emphasizes that the solution implemented in FPGA exceeds the performance of both GPU and CPU. Another application specific circuit implemented in FPGA is found at [15] and in [16] spintronic computational RAM is used to great effect especially in reducing energy consumption. Other types of heterogeneous systems have also been studied, such as [17], targeting large FFTs on a heterogeneous multi-core distributed system.

From an energy consumption point of view, a comparison is made between solutions that use CPU, GPU, FPGA, or heterogeneous computing (CPU + accelerator based on FPGA) [12], considering FPGA implementations more efficient. An energy efficient many-core architecture was also developed in [13] in the form of a mesh connected matrix of processing units. As this and many other papers point out, power and energy are important consideration for accelerator design. Estimations based on a previous version of our architecture (that was developed in ASIC) point to a power usage of around 20W for 2048 computing elements [14] and roughly 5W when computing 64 FFTs of 1024 samples on an 1024 cell machine [22].

Results from GPU and FPGA implementations found in literature are summarized in [27], Table 1. Latency is expressed in both time and clock cycles. Additional surveys regarding FFT implementations can be found at [18, 19] and [20] in Table 29.

Our contributions consist of adapting the FFT algorithm to a Map-Scan architecture (of our own design), and evaluating its performance. We investigate obtainable acceleration in multiple implementations of the algorithm, depending on FFT and accelerator size and in relation to the desired throughput/latency trade-off. Acceleration is discussed in relation to a mono-core machine, specifically we are interested in the evaluation of *architectural acceleration*, by which we understand mainly the comparison of speed performance of two different systems operating with the same clock frequency. This study helps partially validate our novel architecture as a general-purpose accelerator, the lack of such architectures being the research gap we have identified and are actively studying in this and in other papers.

2. Map-Scan Structure & Architecture

The proposed accelerator is part of a heterogeneous system having a host & the accelerator itself. It receives programs and commands from the host and exchanges data with the host's memory.

2.1. Accelerator's structure

The structure of the general-purpose Map-Scan accelerator considered in our approach is based on the mathematical model of computation proposed by Stephen Kleene [1]. A previous version of this accelerator was also used in [6, 7, 21–23] having a Map-Reduce architecture. The Map-Scan accelerator is presented in Fig. 1, where:

- MAP: is a linear array of p cells, each having an accumulator-based execution unit and a m-location one-output port register file.
- CONTROL: containing: (1) COMPUTE (a mono-core controller used to send, through the log-depth pipelined net DISTRIBUTE, in each cycle an instruction to be executed in each cell of the MAP. Its program memory is loaded by the HOST's processor using the Program input. On the same input it receives commands and associated parameters), and (2) TRANSFER (which provides an automaton-based control for data transferred between the HOST's data memory and the register file found in each cell from MAP.
- SCAN: is a log-depth programmable generic scan network whose main function used in performing FFT is permutation.

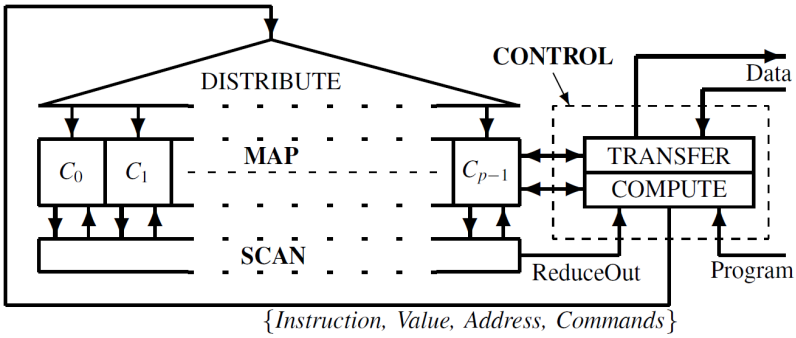

Fig. 1. The cellular system with two global loops: through the vector received back from SCAN and through the scalar received by the COMPUTE's processor from ReduceOut.

The compute and transfer processes can be used in parallel to one another, thereby reducing the memory wall effect.

The SCAN net receives from MAP a p-component vector and sends back a vector of the same size, thus closing a global loop over MAP. The ReduceOut output sends a scalar to COMPUTE. This scalar can be used by CONTROL to close through DISTRIBUTE, another loop over the MAP array. For the FFT function, the SCAN network is used for the permutation function.

2.2. Architecture and programming

COMPUTE is a standard mono-core controller whose program memory stores in each location a pair of instructions: one to be executed by COMPUTE's processor and another to be sent through DISTRIBUTE to be executed with a latency of $log_2 p$ in each active cell of MAP. Thus, the assembly program of the system has two columns of instructions, one for CONTROL and another for MAP and SCAN.

Data memory is distributed in the MAP array and can be seen as the two-dimension array, $\mathbf{M_{mp}}$, of m lines seen as ***horizontal vectors*** – H_0, H_1, …, H_{m-1} – whose components are distributed along the MAP array, and p columns seen as ***vertical vectors*** – V_0, V_1, …, V_{p-1} – whose components represent the local memory in each cell. The structure and the content of the matrix $\mathbf{M_{mp}}$ is represented in Fig. 2.

$\mathbf{M_{mp}}$	V_0	V_1	…	V_{p-1}
H_0	s_{00}	s_{01}	…	$s_{0\ p-1}$
H_1	s_{10}	s_{11}	…	$s_{1\ p-1}$
⋮	⋮	⋮	⋮	⋮
H_{m-1}	$s_{m-1\ 0}$	$s_{m-1\ 1}$	…	$s_{m-1\ p-1}$

Fig. 2. Matrix of data distributed in MAP.

The CONTROL section has following data and control resources: accumulator(used as

left operand and destination), address(used for relative addressing), scalar memory(the local memory/register file), program counter, instruction register, program memory.

The SCAN network is modeled after the Benes permutation network pattern which is adapted to perform, in addition to the permutation function, pack, prefix and reduce functions [5]. The SCAN network receives at the input a vector of p components of form $\{data, destination, function, enable\}$ and generates at the output a data vector that corresponds to the function applied on the enabled data. FFT computing only requires usage of the permutation function. A basic permutation cell is made up of two multiplexers, a small control circuit and two output pipeline registers. If other functions (for accelerating applications other than FFT) are desired, additional circuitry can be synthesized in each cell. It should be noted that while the FFT requires only specific permutations and has fixed data movement patterns, this architecture uses a network capable of any permutation in order for it to be of use with other applications and maintain the general-purpose nature of the accelerator.

From the system level point of view the accelerator is used as a hardware library of functions and for efficiency the development of function libraries is usually done in assembly language. The corresponding instruction set architecture is dual in the sense that the system executes two instructions in each clock cycle, one in CONTROL and the other in the MAP-SCAN array. For CONTROL, the instruction set is a conventional one, while for the MAP-SCAN array, in addition to the standard logic-arithmetic instructions, shared with CONTROL, there is a subset of global instructions and a spatial control (cell-level enable/disable) subset.

3. FFT Implementations

FFT calculation in a parallel structure poses two categories of problems. Those related to data movement (transfer to/from host's memory and access to operands in the butterfly pattern) and those related to the computation itself. Computation (additions and multiplications) is done in the processing elements found in the MAP section which can be considered a SIMD structure. Computation (without the data movement aspects of the algorithm) can be accelerated by a factor of p, the number of processing cores used.

Depending on application requirements in terms of throughput and latency, multiple ways of organizing data can be done. The way in which data is organized will directly affect the way the FFT algorithm is written and position on the throughput vs latency spectrum. Depending on the approach used (decimation in time or decimation in frequency) an additional permutation must occur as the first or as the last step in the algorithm. Regarding twiddle factors, the presented implementations assume these constants are precomputed and stored appropriately in cell memory. If cell memory is limited, they can be computed in place, at a cost in performance.

3.1. Data movement

3.1.1. Data transfer

When an algorithm is run on a parallel structure the memory wall problem needs to be addressed. The computational capacity of a parallel structure implies a "hunger" for data that must be sated. Each specific algorithm will have an operational intensity (number of operations per bytes transferred) and for n-sample FFT computation only for large values of n, the operational intensity becomes large enough so that data transfer can become negligible.

For relatively small sizes, data transfer is achieved in a comparatively large time that cannot be neglected.

The general accelerator that we are analyzing solves the transfer problem through a transfer mechanism that is transparent to the calculation process. If local memory size in each cell is large enough computation can be carried out in one section of the memory while a second section outputs already computed results or prepares input data for future work.

Consequently, in the following sections it will be sufficient to evaluate calculation time because transfer times to and from the host memory do not affect total speed performance.

3.1.2. Butterfly permutation

The SCAN network is used with its secondary function, that of a permutation network. Its depth is in $O(\log n)$, which will become a performance limiting factor for our FFT implementations. Depending on application needs, data can be organized in several ways (as described in subsection 3.2.), each treating the butterfly data movement in a particular way.

3.1.3. Transpose

An important function that we will use to optimize large FFTs will be the transpose of a square matrix. This will require the permutation function from the SCAN network.

Considering the matrix **V** of components v(i,j), whose scalar components must be transposed into the matrix **W** of scalar components w(i,j), for $i, j = 0, 1, \ldots, n-1$, we consider the following transpose algorithm:

```
/*****************************************
Algorithm for matrix transpose
Initial: v(i,j), for i,j = 0, 1, ... n-1
Final:   w(i,j), for i,j = 0, 1, ... n-1
vector: A = [a[0] a[1] ... a[n-1]]
Note: add and sub operations are modulo n
*****************************************/
for(k=0; k<n; k=k+1) begin
 (1) A <= [v(0-k,0) v[1-k,1]...v[n-1-k,n-1]
 (2) A <= [a(0-k) a(1-k) ... a(n-1-k)]
 (3) for(j=0; j<n; j=j+1) w(j+k,j) <= a(j)
end
```

The steps (1) and (3) in the main loop are performed in the MAP array of the system (see Fig. 1). For step (2), the permutation function of the SCAN circuit is used. While for the special case when $n = p$ the rotate function can pe performed in the MAP array, the general case, when $n < p$, in MAP array can be distributed as $\lfloor p/n \rfloor$ $n \times n$ matrices stored in n vectors. In this case, the rotate operation on sub-vectors of n components can only be done using the permute function appropriately defined. A number of n specific permutations must be defined by specifying n p-component destination control vectors used by the SCAN network.

Let us consider the small example of a 4×4 matrix. In Fig. 3, the first column represents the initial content of matrices V and W. For each of the 4 runs of the main loop there is a column that represents the evolution of the content of matrix W.

```
  Initial        k = 0     k = 1     k = 2     k = 3
V:0 1 2 3  (1) 0 5 A F   C 1 6 B   8 D 2 7   4 9 E 3
  4 5 6 7
  8 9 A B  (2) 0 5 A F   1 6 B C   2 7 8 D   3 4 9 E
  C D E F
W:x x x x  (3) 0 x x x   0 x x C   0 x 8 C   0 4 8 C
  x x x x        x 5 x x   1 5 x x   1 5 x D   1 5 9 D
  x x x x        x x A x   x 6 A x   2 6 A x   2 6 A E
  x x x x        x x x F   x x B F   x 7 B F   3 7 B F
```

Fig. 3. Transpose algorithm exemplified for a 4×4 matrix.

The execution time on a system with p cells in the MAP section (see Fig. 1) belongs to $O(nlog\ p)$, while acceleration belongs to $O((n/log\ p) \times \lfloor p/n \rfloor) = O(p/log\ p)$. In the previous evaluation, the log-depth SCAN net limits the performance of the algorithm. For each rotate performed in step (2) of the algorithm we must use the permutation function in order to reduce the linear time to a logarithmic one, but even this logarithmic time affects performance too much.

The implementation on the MAP-SCAN accelerator allows the "absorption" of the latency effect of $-1+2 \times log_2 p$ clock cycles due to the fact that the content of the vector applied to the SCAN network does not depend on the content of the previously applied vectors. The vectors read in step (1) can be applied in turn to the SCAN network, so that the propagation in logarithmic time, in the permuted form, of the first vector is expected only once. The following permuted vectors will be available for step (3) as soon as they are needed to be reloaded into the W matrix. Therefore, the SCAN pipeline can act strictly in parallel with the computation performed in the MAP array, so that the $(O(log\ p))$-delay does add up only once to the duration of the loop. In conclusion, the execution time of the transposition operation for $n \times n$ matrices for $n \leq p$ goes from $O(n \times log\ p)$ to $O(n)$.

3.2. Data organization

Several ways to organize input data are possible, depending on the number of samples of the FFT and desired performance or memory resources we want involved in the calculation. Thus, the samples of the FFTs can be loaded on horizontal vectors or vertical vectors, and for a FFT with a large number of samples organization, in matrices of vertical and horizontal vectors/sub-vectors. As described in Fig. 2, vertical vectors represent data stored inside the memory of one computational cell and horizontal vectors represent data stored at a specific memory address along all computational cells. The term sub-vectors is used to refer to horizontal data that only spans across a fraction of the processing elements.

3.2.1. Horizontal vector/sub-vectors organization of samples

This case loads samples in horizontal vectors. Communication for the $log_2 n$ "butterflies" is done using the permutation function, where n is the number of samples. The number of FFTs performed in parallel will be $\lfloor p/n \rfloor$. For each of the $log_2 n$ stages of the computation, execution time expressed in number of clock cycles will be $const + 2 \times log_2 p$. Obtained acceleration will be theoretically limited to $O(p/log\ p)$ with very low memory consumption.

This solution is used when memory size constrains are in place or when low latency is

preferred to high throughput.

3.2.2. Vertical vector organization of samples

When the number of FFTs/second is high, we adopt the solution to load all samples for each FFT in one cell as a vertical vector. In this case p FFTs are computed in parallel and the program does not need to activate the permute function of the SCAN network. The program executed by cells of the MAP array is almost identical with one of a mono-core version. Acceleration is in $O(p)$. The price we pay is a large local memory in each cell and a high latency. On a positive note this organization offers the highest throughput.

3.2.3. Square matrix organization of samples

The square matrix organization of the samples is useful for large values of n. In this case, samples will occupy multiple memory locations across all processing cores. This can be seen as multiple horizontal vectors stacked vertically. The immediate solution is to first apply the horizontal vector method for the $\sqrt{n}$ lines, followed by applying the vertical vector method on the columns of the matrix or matrices. The first stage, applied to horizontal vectors has low efficiency because it ensures an acceleration in $O(p/log\ p)$ (see Section 3.2.1.). The second stage is very efficient because it offers an acceleration in $O(p)$ (see 3.2.2.).

In order to avoid the decrease of the acceleration due to the first stage, we propose the following algorithm:

```
/*****************************************
FFT algorithm for square matrix organization
*****************************************/
(1) Perform first half "butterflies" on the vertical vectors
(2) Transpose the resulting matrix
(3) Perform the second half "butterflies" on the resulting
vertical vectors
```

Thus, we benefit from two advantages: usage of the vertical solution in computing FFT (acceleration in $O(p)$), and the use of the transpose operation instead of multiple ”butterfly” data movements(acceleration in $O(p)$). This helps avoid latencies introduced by the permute function used to solve the “non-locality” issue due to the “butterfly” interconnection.

3.2.4. Rectangular matrix organization of samples

An intermediate solution allows the organization of input data into rectangular matrices. In this case, a combination of the first two versions is used, with the disadvantage of operating with horizontal vectors that do not avoid the latency introduced by permutations.

This solution offers the possibility to compromise between latency and throughput.

4. Results

The accelerator was simulated in a version with $p = 64$ computational cells. Programs running on the accelerator were written in assembly language. They were designed as part of

a kernel library so that the transfer operations were not included, representing distinct functions in the library. The weight of these operations is reduced anyway and decreases with the increase of the number of samples. Furthermore it is almost completely negated by the fact that the data transfer in the MAP-SCAN accelerator can be done transparently with the calculation in stream processing mode, similar to what is seen in [8]. Each cell's memory is split into two halves that can be processed and IO-accessed independently, thus allowing complete data prefetching and result offloading without affecting computation in any way. Transfer comes into effect only for the first and last FFTs to be computed.

Real-to-complex FFTs and fixed-point arithmetic were used in the evaluation experiments. The scenario of having missing input samples is not covered in this study.

4.1. Horizontal vector/sub-vectors organization of samples

The acceleration provided for this implementation is limited by the *log*-depth of the scan network. It is almost constant with the number of sample.

In this version, samples can be loaded as vectors or as sub-vectors. The system used allowed the evaluation of a FFT with 64 samples, two FFTs of 32 samples, or four FFTs of 16 samples.

In Table 1 results are presented on six columns used to show in turn: the number of samples per FFT, the number of FFTs performed in parallel, the number of clock cycles per sample, throughput acceleration, latency acceleration, and latency in number of clock cycles.

Table 1. Acceleration for samples organized as horizontal vectors or sub-vectors on 64-cell machine

Samples/FFT	FFTs in parallel	Cycles/sample	Th. Acc.	Lat. Acc.	Lat. in cycles
8	8	5.5	28.5	3.6	353
16	4	7.3	28.4	7.1	464
32	2	9	28.4	14.2	575
64	1	10.7	28.4	28.4	686

Throughput acceleration is almost constant and smaller than the structural degree of parallelism which is 64, limited by the latency of the permutation function performed by the SCAN network. Latency acceleration improves with the number of samples because the scan network is inappropriately sized for smaller FFTs.

4.2. Vertical vector organization of samples

If the samples for each FFT are organized as a vertical vector, then the "non-locality of butterflies" does not affect the execution efficiency, because each cell is programmed with a program designed for a mono-core structure. The throughput acceleration obtained is $\simeq 96$, regardless of the number of samples, provided that the local memory in each cell is sufficient. For the same reason latency acceleration is also almost constant.

Table 2 shows the evaluation results of this format. Unlike the first solution, it uses a lot of memory, paying in memory to increase throughput acceleration.

The supralinear value of the acceleration (greater than the number $p = 64$ of cells) is due to the parallelism between control and execution. In a single-core implementation, the computation control and the computation itself are performed by the same structure, while in the accelerator we are analyzing, computation is executed in MAP and control in COMPUTE.

Table 2. Cycles per sample for samples organized as vertical vectors

Samples/FFT	FFTs in parallel	Cycles/sample	Th. Acc.	Lat. Acc.	Lat. in cycles
16	64	2.21	94.7	1.49	2268
32	64	2.71	95.3	1.49	5563
64	64	3.22	95.6	1.50	13.2k
128	64	3.73	95.8	1.50	30.6k
256	64	4.25	95.9	1.50	69.7k

4.3. Square matrix organization of samples

When the input data is organized as square matrices, in our development system we used 4 cases: 8 FFTs with $8 \times 8 = 64$ samples, 4 FFTs with $16 \times 16 = 256$ samples, 2 FFTs with $32 \times 32 = 1024$ samples, and 1 FFT with $64 \times 64 = 4096$ samples. The results are presented in Table 3.

Table 3. Acceleration for samples organized as square matrices on 64-cell machine

Samples/FFT	FFTs in parallel	Cycles/sample	Th. Acc.	Lat. Acc.	Lat. in cycles
64	8	3.5	86.68	10.83	1798
256	4	4.5	89.88	22.47	4608
1024	2	5.5	91.79	45.89	11.2k
4096	1	6.52	92.94	92.92	26.7k

Acceleration is also supralinear (for the same reason as the previous case) and tends to $\simeq 96$ with the increase of the number of samples due to the fact that the weight of the transpose operation decreases with the value of n.

4.4. 256 and 1024 component FFT in various forms

To compare the three previous solutions, we will consider the case $n = 256$ on 64-cell machine implemented (1) as a FFT represented as 4 horizontal vectors of 64 components each, (2) as 4 FFTs represented in 16 vertical vectors containing 4 matrices 16×16, and (3) as 64 FFTs represented as vertical vectors. In Table 4 it is obvious that the third solution is the best for throughput, but this is paid for with a very large amount of memory and a large latency. The first solution requires 64 times less memory and throughput decreases only three times. The second version is a good compromise, when it is acceptable.

Table 4. Acceleration for samples organized in various forms on 64-cell machine

FFTs in parallel	H. vec. occupied	Cycles/sample	Th. Acc.	Lat. Acc.	Lat. in cycles
1	4	10.47	38.62	38.62	3121
4	16	4.5	89.88	22.47	4608
64	256	4.25	95.9	1.50	69.7k

We will have the possibility to choose the solution that meets the requirements of the project: using minimal memory or either a high throughput paid in high latency or a low latency paid in low throughput variant. Intermediate variants between the two extremes are also possible.

For comparison, the FFT 4096 was tested on three other systems: local testing (on an Intel i7770HQ, 8CPU, 2.8 GHz) of a CPU implementation of the FFTW algorithm [24] [25], local

testing (on an Nvidia GeForce GTX1050, 640 cores, 1.354 GHz) of a GPU implementation [25], FPGA Xilinx IP implementation (targetting ZYNQ 7020 SoC, default settings) [26].

Frequency scaled to 1.354 GHz, the CPU implementation achieved roughly 126 MSa/s (Mega samples per second), the GPU 175 MSa/s and our proposed general-purpose accelerator (64 cell machine, square memory configuration) 207 MSa/s. Something to note is that our implementation is in fixed point, while the CPU and GPU operate in floating point, and as such an approximately $\times 3$ performance degradation is expected when our architecture switches to floating point. In the case of computing FFTs with a large number of samples, fixed point will lose precision and floating point will remain the only possible approach even if it will mean obtaining higher latencies. Another thing to note is that compared to the GPU, our machine uses 64 cores instead of 640 and would be much more energy efficient. This also shows our architectural benefits as the computing cores are much more efficiently used. The Xilinx IP configurator (on default settings for fixed point FFT) offers a latency of 14465 cycles, roughly half of what our machine offers. However this IP targets only FFT computation, and is not a general-purpose architecture. Other implementations do offer superior results.

In order to compare with the values presented in [27] estimated latencies for 1024 samples FFTs are shown in Table 5. In the "Vertical" data organization mode, the number of vertical vectors occupied is related to throughput, to the number of FFTs processed and to the number of cell used. It does not affect latency in any way. Accelerator size is given in number of cells.

Table 5. FFT 1024 latency and throughput estimations

Implementation	Accelerator size	FFTs in parallel	H. vec.	V. vec.	Lat. in cycles
Horizontal	1024	1	1	1024	1257
Horizontal	64	1	16	64	14.2k
Vertical	X	X	1024	X	347k
Square	32	1	32	32	11.2k

Using these estimations and comparing with [27], Table 1, we can place our solution (from a latency point of view) between FPGA implementations and GPU implementations. FPGA implemented accelerators outperform our design in terms of latency and resources used, but are bound to the application for which they are designed.

Given the two sets of comparisons, we can conclude that our architecture offers advantages compared to currently available hardware solutions. While other solutions can offer similar or even better results in terms of latency or throughput, CPUs are not scalable, GPUs are not particularly well suited for the memory intensive parts of the FFT algorithm and FPGA accelerators are application specific and not general-purpose. Based on obtained simulation results and ASM code metrics, additional estimations have been made for different implementations. These are presented in [27], Table 2.

5. Discussion

While standard solutions offer throughput accelerations well below the computing potential of the devices used, the Map-Scan solution we propose can achieve supralinear throughput accelerations in many cases. For an accelerator equipped with p execution units we managed to obtain throughput accelerations of $\sim 1.5 \times p$. This fact is due to: (1) transparency in relation to computation with which data transfer between the accelerator and the host memory is executed

(allowed by the dual control of the system: COMPUTE and TRANSFER) , (2) the decoupling between execution carried out in the MAP and control that is carried out in COMPUTE.

Latency improvement varies greatly from a minimum of ~ 1.5 (effects of control separated from computation) to a maximum of just under 96 (1.5×64, where 64 is the number of execution units used). Depending on application requirements a choice regarding the algorithmic variant used must be made in order to compromise between latency and throughput. This performance was achieved due to the pairing between the MAP (used for computation) and SCAN (used for permutation to efficiently perform the "butterfly" data movement) parallel patterns. By adding the SCAN-Permute circuit we greatly improved a previous use of a MAP-only architecture described in [22] and [23].

While the FFT is in the complexity class of $O(n \times \log n)$, it can still be accelerated with good results. The following speed-ups are achieved: Horizontal setup: n, Vertical setup: 1.5 (with increased throughput), Square setup with $n = p \times p$ samples: $\sqrt{n}$.

Regarding scalability, this subject can be approached from multiple perspectives:

Hardware scalability: the architecture is completely scalable, no form of architectural issues appeared on currently tested sizes, on FFT or on other algorithms. FFT computation is only limited by cell (processing core from MAP section) memory size and total available LUTS or area. Hardware is easily scalable, the number of processing elements being an easily changeable architecture parameter.

Software scalability: when trying to compute a very large FFT on a limited hardware machine (smaller number of computational cells), a mechanism for function calls and memory transfers to/from the host machine is needed. With it, the FFT is computed part by part. While this mechanism is under development and we could not test this setup, the only forseable problem is transfer bottleneck which is resolved by a separate mechanism within our architecture [8].

FFT computation of smaller sample sizes on a larger hardware structure: this setup requires multiple groups of cells, each group being preoccupied with its own FFT. Each group can then use a horizontal, rectangular or square data setup and algorithm, depending on application needs. The only limitation is that all cell groups need to have the same parameters and will run the same algorithm.

Considering that enough data is available, increasing the number of cells (and cell memory) will maintain the obtained acceleration.

The concept of general-purpose hardware accelerators is not widely approached, GPGPUs being the standard acceleration solution for most problem classes. Because of this and as this project is in early development, eloquent comparisons cannot be made. Comparing our FPGA targeted proof of concept accelerator with ASIC implemented GPUs or comparing our general-purpose accelerator with single-function FPGA accelerator designs is inappropriate. However, some quantitative comparisons were attempted in Section 4.. With this and other published or in development papers (tackling different architectural aspects or problem categories), we aim to prove the merits of this architecture.

While this project shows great results in simulation and in previous versions of the architecture, this study is limited by the lack of a proper SDK and toolchain with which complete hardware evaluations can be done.

Future work regarding this accelerator has two broad directions: (1) proving its general-purpose nature by tackling problems from different domains and (2) creating proper software layers above it so that it can be better tested and used.

6. Conclusions

This study of FFT computations on our proposed architecture highlights the numerous benefits it entails, such as: configurability of used circuit resources, scalability with problem size and choice between low latency or high throughput algorithmic versions (and intermediary results between these two extremes). High throughput variants tend to reach throughput accelerations of $\sim 1.5 \times p$ (with "p" being the number of processing cores used) while low latency variants (on a large enough machine) take $\sim 110 \times log(n)$ clock cycles to complete obtaining a latency acceleration of a little under $1/2 \times p$.

Obtained results show that the spectral methods class of problems (to which the FFT belongs) can efficiently fit on our proposed general-purpose accelerator architecture.

References

[1] S. KLEENE, *General recursive functions of natural numbers*, Mathematische Annalen **112**, 1936, pp. 727–742.

[2] M. MALIŢA, G. V. POPESCU and G. M. ŞTEFAN, *Heterogeneous computing for Markov models in big data*, Proceedings of 2019 International Conference on Computational Science and Computational Intelligence, Las Vegas, NV, USA, 2019, pp. 1500–1505.

[3] V. DRAGOMIR and G. M. ŞTEFAN, *Sparse matrix-vector multiplication on a map-reduce many-core accelerator*, Romanian Journal of Information Science and Technology **23**(3), 2020, pp. 262–273.

[4] M. MALIŢA, D. MIHĂIŢĂ and G. ŞTEFAN, *Molecular dynamics on FPGA based accelerated processing units*, MATEC Web of Conferences **125**, 2017, paper 04012.

[5] M. ANTONESCU and G. M. ŞTEFAN, *Multi-function scan circuit*, Proceedings of 2020 International Semiconductor Conference, Sinaia, Romania, 2020, pp. 123–126.

[6] G. M. ŞTEFAN and M. MALIŢA, *Can one-chip parallel computing be liberated from ad hoc solutions? A computation model based approach and its implementation*, Proceedings of 2014 International Conference on Circuits, Systems, Communications and Computers, Santorini Island, Greece, 2014, pp. 582–597.

[7] M. MALIŢA, G. M. ŞTEFAN and D. THIÉBAUT, *Not multi-, but many-core: Designing integral parallel architectures for embedded computation*, ACM SIGARCH Computer Architecture News **35**(5), 2007, pp. 32–8.

[8] G. V. POPESCU, *Improvements in data transfer for a Map-Reduce accelerator*, Romanian Journal of Information Science and Technology (ROMJIST) **25**(3–4), 2022, pp. 368–380.

[9] M. N. HAQUE and M. S. UDDIN, *Accelerating fast Fourier transformation for image processing using graphics processing unit*, Journal of Emerging Trends in Computing and Information Sciences **2**(8), 2011, pp. 367–375.

[10] D. PUCHALA, K. STOKFISZEWSKI, B. SZCZEPANIAK and M. YATSYMIRSKYY, *Effectiveness of fast Fourier transform implementations on GPU and CPU*, Przeglad Elektrotechniczny **92**(7), 2016, pp. 69–71.

[11] C. CULLINAN, C. WYANT and T. FRATTESI, *Computing performance benchmarks among CPU, GPU, and FPGA*, [Online] `https://digital.wpi.edu/downloads/5d86p189q`, Accessed on 08.05.2023.

[12] B. BETKAOUI, D. B. THOMAS and W. LUK, *Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing*, Proceedings of 2010 IEEE International Conference on Field-Programmable Technology, Beijing, China, 2010, pp. 94–101.

[13] R. PRASAD, *Integrated Programmable-Array accelerator to design heterogeneous ultra-low power manycore architectures*, Université de Bretagne Sud, France, Università degli studi, Bologna, Italy, 2022.

[14] M. MALIŢA, O. NEDESCU, A. NEGOIŢĂ and G. ŞTEFAN, *Deep learning in low-power stereo vision accelerator for automotive*, Proceedings of 2018 IEEE International Conference on Consumer Electronics, Las Vegas, NV, USA, 2018, pp. 1-6.

[15] A. MUKHERJEE, A. SINHA and D. CHOUDHURY, *A novel architecture of area efficient FFT algorithm for FPGA implementation*, ACM SIGARCH Computer Architecture News **42**(4), 2015, pp. 1–6.

[16] H. CILASUN, S. RESCH, Z. I. CHOWDHURY, E. OLSON, M. ZABIHI, Z.-Y. ZHAO, T. PETERSON, J.-P. WANG, S. S. SAPATNEKAR and U. KARPUZCU, *CRAFFT: High Resolution FFT accelerator in spintronic computational RAM*, Proceedings of 2020 ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 2020, pp. 1–6.

[17] Y. LI, J.R. DIAMOND, X. WANG, H. LIN, Y YANG and Z HAN, *Large-scale fast Fourier transform on a heterogeneous multi-core system*, The International Journal of High Performance Computing Applications **26**(2), 2012, pp. 148–158.

[18] M. GARRIDO, *A survey on pipelined FFT hardware architectures*, Journal of Signal Processing Systems **94**, 2022, pp. 1345–1364.

[19] E. KONGUVEL and K. MU, *A survey on FFT/IFFT processors for next generation telecommunication systems*, Journal of Circuits, Systems and Computers **27**(3), 2018, paper 1830001.

[20] C. BÎRĂ, *[Digital] Electronics by Example When Hardware Greets Software*, MATRIX ROM, Bucharest, 2024.

[21] V. CODREANU, L. PETIRICĂ, and R. HOBINCU, *Increasing vector processor pipeline efficiency with a thread-interleaved controller*, Proceedings of 2011 International Conference on System Theory, Control and Computing, Sinaia, Romania, 2011, pp. 1–4.

[22] I. LÖRENTZ, M. MALIŢA and R. ANDONIE, *Fitting FFT onto an energy efficient massively parallel architecture*, Proceedings of the Second International Forum on NextGeneration Multicore/Manycore Technologies, Saint-Malo, France, 2010, pp. 1–11.

[23] C. BÎRĂ, L. GUGU, M. MALIŢA and G. ŞTEFAN, *Maximizing the SIMD behaviour in SPMD engines*, Proceedings of the 2013 World Congress on Engineering and Computer Science, San Francisco, CA, USA, 2013, pp. 1–6.

[24] M. FRIGO and S. G. JOHNSON, *The design and implementation of FFTW3*, Proceedings of the IEEE **93**(2), 2005, pp. 216–231.

[25] A. HURD, 2018, *fftw-cufftw-benchmark*, [Online] `https://github.com/hurdad/fftw-cufftw-benchmark`, Commit cfc8aa8, Accessed on 08.05.2023.

[26] AMD-Xilinx, *PG109 (04.05.2022): Fast Fourier Transform v9.1 LogiCORE IP Product Guide*, [Online] `https://docs.xilinx.com/r/en-US/pg109-xfft/Fast-Fourier-Transform-v9.1-LogiCORE-IP-Product-Guide`, Accessed on 08.05.2023.

[27] M. ANTONESCU and M. MALIŢA, *Supplementary material of the paper M. ANTONESCU and M. MALIŢA, FFT on a Heterogeneous System with a General-Purpose Map-Scan Accelerator*, Romanian Journal of Information Science and Technology, 2023. Accessed: Oct. 1, 2023. [Online] `https://drive.google.com/file/d/1ECA9tcI8YBevY-2wAvjW7lCTfOtOQQ8G/view?usp=sharing`