Design, Optimization, and Implementation of a Universal FFT Processor

Pinit Kumhom
Jeremy Johnson
and
Prawat Nagvajara

Technical Report DU-MCS-00-01
Department of Computer Science
Drexel University
Philadelphia, PA 19104
September 2000

Design, Optimization, and Implementation of a Universal FFT Processor[*]

Pinit Kumhom, *Jeremy R. Johnson* and *Prawat Nagvajara*

Drexel University, Philadelphia, PA

Abstract

There exist Fast Fourier transform (FFT) algorithms, called dimensionless[1] FFTs, that work independent of dimension. These algorithms can be configured to compute different dimensional DFTs simply by relabeling the input data and by changing the values of the twiddle factors occurring in the butterfly operations. This observation allows us to design an FFT processor, which with minor reconfiguring, can compute one, two, and three dimensional DFTs. In this paper we design a family of FFT processors, parameterized by the number of points, the dimension, the number of processors, and the internal dataflow, and show how to map different dimensionless FFTs onto this hardware design. Different dimensionless FFTs have different dataflows and consequently lead to different performance characteristics. Using a performance model we search for the optimal algorithm for the family of processors we considered. The resulting algorithm and corresponding hardware design was implemented using FPGA.

1 Introduction

In many applications, the Fast Fourier Transform (FFT) presents an intensive computational task due to the amount of data to be processed. The amount of data (i.e., problem size) depends on the number of points and the dimension of the transform. To this end, engineers and scientists rely on approaches such as highly-tuned code for uniprocessors, DSP processors, ASIC, IP cores, and reconfigurable architecture, to meet the performance requirements with respect to other design constraints such as physical space. A list of references to these approaches is provided in [1]. Our study, which is part of the SPIRAL project [?], focuses on using mathematical properties of the FFT to help us design a high-performance hardware.

The novelty of our work is threefold. First we base our processor on the dimensionless FFT [2] which allows a single hardware design to compute one, two, and three dimensional DFTS. Second, we provide a framework for systematically mapping alternative FFT algorithms onto parameterized hardware designs. This is obtained by mapping a mathematical description of the FFT, based on matrix factorizations [3], to hardware that implements flow control and generation of the necessary roots of unity (twiddle factors). Finally, there are many different FFT algorithms, each with different dataflow, and consequently different performance characteristics. Thus our hardware design becomes on optimization problem over the space of possible FFT dataflows [4].

We thus propose a universal FFT engine that is parameterized in terms of the number of points and dimension of the transform, and the choice of the algorithm. We consider a distributed architecture comprised of processing units (with local memory) connected via an interconnection network. We derived a class of optimal FFT dataflow diagrams based on memory-access cost function. The derivation followed a design flow that uses a performance model and its simulation to evaluate the choice of algorithm prior to design of the hardware [5]. Implementation of the engine for proof of concept on the Wildforce^{TM} [6] reconfigurable (FPGA) board is performed in two steps, hardware description language model simulation of the board and the actual execution of the configured board. We have validated the actual configured board and are in the process of benchmarking its performance. Future implementation may use the ASIC technology for the floating-point (complex numbers) arithmetical cores and the FPGA technology for the parameterized flow control units.

In Section 2 we review the dimensionless FFT and the space of FFT algorithms that we will consider. In Section 3 we describe a family of FFT processors and the mapping of algorithms in Section 2 to this architecture. In Section 4 we introduce a performance model for the architecture in Section 3 and find the optimal algorithm with respect to this model. In Section 5 we describe the implementation of the design selected in Section 4. Details not provided in this paper can be found in [1].

[*]This work was partially supported by DARPA through research grant DABT63-98-1-0004 administered by the Army Directorate of Contracting. Additional support was provided by MathStar, Inc.

[1]Patent #US6003056

2 Dimensionless FFT Algorithms

Let $X(a_1, \ldots, a_t)$ be a function of t variables, where $0 \le a_i < n_i$. The t-dimensional $n_1 \times \cdots \times n_t$ DFT of X is

$$\hat{X}(b_1, \ldots, b_t) = \sum_{0 \le a_i < n_i} e^{\frac{2\pi i}{n_1} a_1 b_1} \cdots e^{\frac{2\pi i}{n_t} a_t b_t} X(a_1, \ldots, a_t)$$

The multidimensional DFT can be interpreted as a matrix-vector product. Let x and $\hat{x}$ be the vectors of size $N = n_1 \cdots n_t$ obtained by ordering the elements of X and $\hat{X}$ lexicographically. Then, $\hat{x} = (F_{n_1} \otimes \cdots \otimes F_{n_t})x$, where $\otimes$, denotes the tensor (Kronecker) product and F_{n_i} is the n_i-point discrete Fourier matrix [3]. FFT algorithms can be represented by factorizations of this matrix [9, 3].

A dimensionless FFT [2] can compute any multidimensional DFT where $n_1 \cdots n_t = N$, for a fixed N. For example, a Fourier transform on 16 points can have dimension equal to 1 (F_16), 2 ($F_2 \otimes F_8$, $F_4 \otimes F_4$, and $F_8 \otimes F_2$), 3 ($F_2 \otimes F_2 \otimes F_4$, $F_2 \otimes F_4 \otimes F_2$, and $F_4 \otimes F_2 \otimes F_4$), or 4 ($F_2 \otimes F_2 \otimes F_2 \otimes F_2$). Independent of dimension, these matrices can be factored into

$$P_4(I_8 \otimes F_2)T_4P_3(I_8 \otimes F_2)T_3P_2(I_8 \otimes F_2)T_2P_1(I_8 \otimes F_2)T_1P_0P_d,$$

where I_8 is an identity matrix, the P_i are permutation matrices and the T_i are diagonal matrices. Only the twiddle factors, T_i and the initial permutation P_d change as the dimension changes. The internal dataflow, given by the permutations P_i are fixed and hence the factorization provides a dimensionless FFT.

In particular, if we set $P_0 = I_{16}$ and $P_1 = P_2 = P_3 = P_4 = L_2^{16}$, the stride permutation [9] which gathers elements of a vector of size 16 at stride 2

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 2 & 4 & 6 & 8 & 10 & 12 & 14 & 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \end{pmatrix},$$

the algorithm is called the dimensionless Pease algorithm because it corresponds in the one-dimensional case to an algorithm described by M. Pease in [8]. Table 1 shows P_d and the twiddle factors that configure the dimensionless Pease algorithm to compute a 1-D and a 2-D FFT. The permutation R_{2^t} is the t-bit bit reversal permuation [9].

The set of internal permutations define the dataflow of the FFT. Figure 1 shows the dataflow of the Pease algorithm (the boxes indicate an F_2 with twiddle computation). Alternative algorithms with different dataflow exist (see [4] for a classification of possible dataflows). For example, the permutations $P_0 = (L_2^8 \otimes I_2)$, $P_1 = L_2^{16}(L_4^8 \otimes I_2)$, $P_2 = (L_4^8 \otimes I_2)L_2^{16}$, $P_3 = (L_4^8 \otimes I_2)L_2^{16}(L_4^8 \otimes I_2)$, $P_4 = L_2^{16}(L_2^8 \otimes I_2)$, define an algorithm whose dataflow is shown in Figure 2.

In the following sections we will see that different dataflows lead to different performance characteristics. The set of possible dataflows are those sequences of permutations that can be configured to compute the FFT. That is, those dataflows for which there exist an initial permutation and a sequence of twiddle factors, such that the resulting matrix factorization is equal to any compatible multidimensional DFT. To optimize our design, we search over the space of allowable dataflows for the one with the best performance.

Table 1: Configuration of the dimensionless Pease algorithm.

To configure for (1-D) F_{16}, set $P_d = R_{16}$, $\omega = e^{\frac{2\pi i}{16}}$,
$T_1 = diag(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)$ $T_2 = diag(1,1,1,1,1,1,1,1,1,\omega^4,1,\omega^4,1,\omega^4,1,\omega^4)$ $T_3 = diag(1,1,1,1,1,\omega^2,1,1,\omega^2,1,\omega^4,1,\omega^4,1,\omega^6,1,\omega^6)$ $T_4 = diag(1,1,1,\omega,1,\omega^2,1,1,\omega^3,1,\omega^4,1,\omega^5,1,\omega^6,1,\omega^7)$
To configure for (2-D) $F_4 \otimes F_4$, set $P_d = R_4 \otimes R_4$, $\omega = e^{\frac{2\pi i}{16}}$,
$T_1 = diag(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)$ $T_2 = diag(1,1,1,1,1,1,1,1,1,\omega^4,1,\omega^4,1,\omega^4,1,\omega^4)$ $T_3 = diag(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)$ $T_4 = diag(1,1,1,1,1,1,1,1,1,\omega^4,1,\omega^4,1,\omega^4,1,\omega^4)$

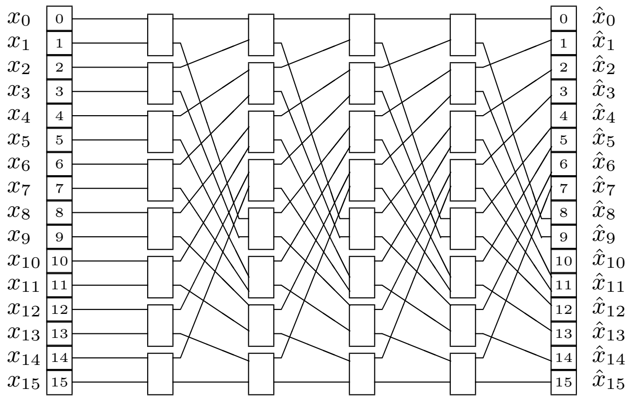

Figure 1: Dataflow diagram for dimensionless Pease algorithm

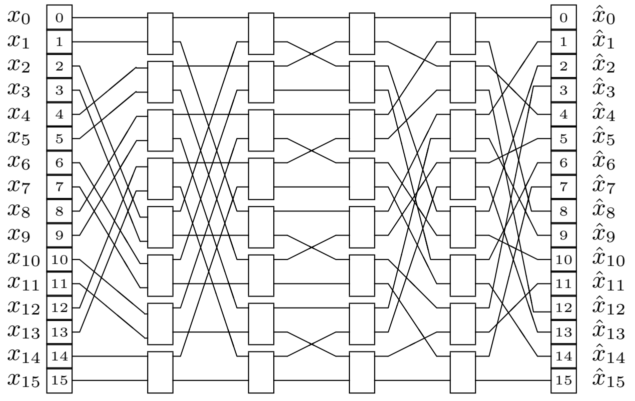

Figure 2: Dataflow diagram for alternative FFT algorithm

3 Architectural Framework

In this section, we describe the architecture of our FFT processor, and illustrate how the algorithms from the previous section can be mapped onto this architecture. The proposed architecture shown in Figure 3 contains 3 main units: the interface, the interconnection network, and the processor elements (PEs.) The interface unit is used to transfer parameters and data to/from the system. The reconfigurable interconnection network provides the communication between the PEs. Each PE has 3 main units: a memory (M), a computation unit (CU), and an address generator (AG). This design is similar to the Xputer proposed in [?].

When used as an FFT processor, the data is distributed amongst the processor memories, the computation unit is used to generate twiddle factors and perform butterfly operations, and the address generators calculate the addresses of the two inputs to each butterfly operation. In order to map an FFT algorithm, represented as a matrix factorization, onto this architecture, address generators must be configured from the permutations occuring in the factorization, and the computation unit must be configured to compute the appropriate twiddle factors. The initial permutation is incorporated into the interface so that the data arrives in the appropriate order. To simplify the address generators, we assume that

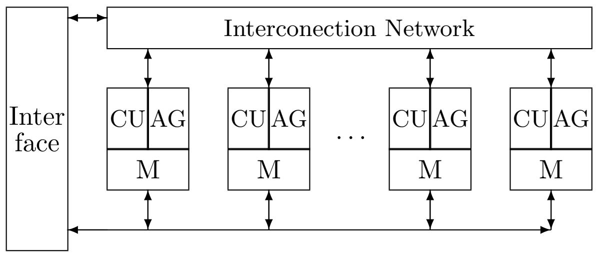

Figure 3: The architecture

the FFT operates on a vector of size $N = 2^n$, the number of PEs is equal to $M = 2^m$, and that the permutations occuring in the FFT are tensor permutations. A tensor permutation is a permutation obtained by permuting the bits in the binary representation of the addresses [9]. Bit reversal and stride permutations are examples of tensor permutations.

The N elements of the input vector are distributed consecutively in segments of size N/M to the processor memories. A memory location containing data is assigned a global n-bit address $(b_{n-1}b_{n-2}\dots b_0)$, where the leading m bits are the PE number and the trailing $n-m$ bits are a local offset. To conserve memory, the computation is performed inplace. This requires a modification to the factorization presented in the previous section, so that each stage is of the form $P(I \otimes F_2)TP^{-1}$. The conjugating permutation P determines the addresses for each butterfly operation. Since all permutations are assumed to be tensor permutations, the permuted addresses can be generated by permuting the bits of a binary counter.

For example, the 16-point Pease algorithm is transformed into the four stage factorization

$$L_2^{16}(I_8 \otimes F_2)T_4L_8^{16} \cdot L_4^{16}(I_8 \otimes F_2)T_3L_4^{16} \cdot$$
$$L_8^{16}(I_8 \otimes F_2)T_2L_2^{16} \cdot (I_8 \otimes F)2)T_1P_d.$$

The addresses obtained for the 8 butterfly operations (two consecutive data elements are used in a butterfly operation) in each of the four stages are shown in Table 2. Each stage shows the permuted address bits, and all addresses are obtained by counting from 0 to 15.

Table 2: Butterfly addresses for the 16-bit Pease algorithm

Stage 0	Stage 1	Stage 2	Stage 3
$b_3b_2b_1b_0$	$b_2b_1b_0b_3$	$b_1b_0b_3b_2$	$b_0b_3b_2b_1$
0000->P_0	0000->P_0	0000->P_0	0000->P_0
0001->P_0	0010->P_0	0100->P_0	1000->P_0
0010->P_1	0100->P_1	1000->P_1	0001->P_1
0011->P_1	0110->P_1	1100->P_1	1001->P_1
0100->P_2	1000->P_2	0001->P_2	0010->P_2
0101->P_2	1010->P_2	0101->P_2	1010->P_2
0110->P_3	1100->P_3	1001->P_3	0011->P_3
0111->P_3	1110->P_3	1101->P_3	1011->P_3
1000->P_0	0001->P_0	0010->P_0	0100->P_0
1001->P_0	0011->P_0	0110->P_0	1100->P_0
1010->P_1	0101->P_1	1010->P_1	0101->P_1
1011->P_1	0111->P_1	1110->P_1	1101->P_1
1100->P_2	1001->P_2	0011->P_2	0110->P_2
1011->P_2	1011->P_2	0111->P_2	1110->P_2
1110->P_3	1101->P_3	1011->P_3	0111->P_3
1111->P_3	1111->P_3	1111->P_3	1111->P_3

Butterfly operations are assigned to PEs using a round robin schedule. Assuming 4 PEs, Table 3 shows the address sequences for each PE for the 16-bit Pease algorithm. The twiddle factors needed for a butterfly are determined in a manner similar to address calculation.

Table 3: Address sequences for 16-point Pease algorithm

PE	Stage 0	Stage 1	Stage 2	Stage 3
0	0,1,8,9	0,2,1,3	0,4,2,6	0,8,4,12
1	2,3,10,11	4,6,5,7	8,12,10,14	1,9,5,13
2	4,5,12,13	8,10,9,11	1,5,3,7	2,10,6,14
3	6,7,14,15	12,14,11,15	9,13,7,15	3,11,7,15

4 Performance Model

In Section 2, it was shown that there are different FFT algorithms with different dataflow patterns. Each algorithm can be mapped onto the architectural framework outlined in Section 3. It is not clear a priori which algorithm should be used when selecting the ultimate design for the FFT processor. Using the performance model, a search, over a subset of possible FFT algorithms, was performed in order to select the most efficient design. The algorithm found using this search process substantially reduces the traffic over the interconnection network when compared to the Pease algorithm.The use of a performance model, instead of complete simulation, allows us to explore many different possible designs at an early stage of the design process. The use of performance models early in the design process has been promoted in [10].

The performance model was implemented using ADEPT [5], a performance modeling tool based on Petri-Nets and implemented in VHDL. In the performance model, data are represented by tokens containing information that affects performance. The flow of the tokens emulates the dataflow in the system. A token in our system contains a pair of addresses corresponding to a butterfly operation. The sequence of addresses are generated and mapped to PEs by a scheduler. The PE, the memory (M) and the interconnection models have a mechanism of passing the tokens that imitates the computation steps including memory read, twiddle factor and butterfly operations, and memory write. The operations and the memory accesses are emulated as delays while the address sequences dictate the flow of data. We use the total simulation time (not the CPU time) reported by the VHDL simulator as the performance cost.

The optimization problem is to find the FFT dataflow with minimal running time. The set of dataflows considered are obtained from the Pease algorithm by multiplying the permutations by tensor permutations of the form $P \otimes I_2$. Any such permutations lead to a valid dataflow and any valid dataflow can be obtained in such a way. Since an exhaustive search is prohibative, the search was limited to the case where P is a stride permutation. The search was performed with a model using four processors with the number of data points ranging from 16 to 1024. Figure 4 compares the performance of the optimal dataflow found with the Pease algorithm.

The addressing for the optimal algorithm for 16 points is given in Table 4. The addresses were generated by the sequence of bit permutations $(b_2b_1b_3b_0)$, $(b_2b_1b_0b_3)$, $(b_2b_0b_1b_3)$, $(b_0b_1b_3b_2)$. Comparing this sequence of addresses to the Pease algorithm shows that the Pease algorithm has 36 non-local memory accesses while the optimal algorithm only has 16. There is a pattern in the optimal dataflow found by the search, which can be parameterized by the number of processors and data points. This dataflow has many interesting properties which simplify its implementation: (1) all data access in the first m stages is local and (2) in the remaining stages, half of the data accessed by a processor is local and the other half is exchanged with one other processor.

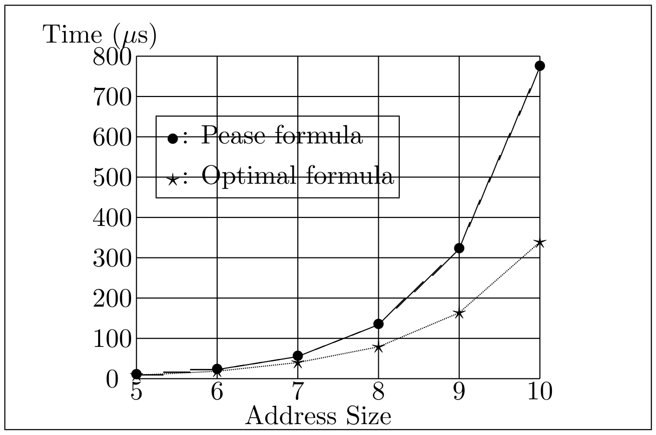

Figure 4: Comparison of the optimal and Pease algorithms

Table 4: Address sequences for optimal 16-point algorithm

PE	Stage 0	Stage 1	Stage 2	Stage 3
0	0,1,2,3	0,2,1,3	0,4,1,5	0,8,2,10
1	4,5,6,7	4,6,5,7	2,6,3,7	4,12,6,14
2	8,9,10,11	8,10,9,11	8,12,9,13	1,9,3,11
3	12,13,14,15	12,14,11,15	10,14,11,15	5,13,7,15

5 Implementation

In this section, we describe the implementation of a universal FFT processor on the Wildforce^{TM} [6] FPGA board. The processor design is based on the optimal dataflow found in the previous section. The board consists of 5 Xilinx FPGA chips (XC4085XLA), with local memories, connected via a configurable crossbar interconnect. The board communicates with a host using a PCI bus. The architecture in Section 4, is mapped to the board as follows. One processor, CPE0, along with its FIFO is used for interface unit. The remaining processors implement four PEs each with a computation unit and address generator. The crossbar is used for the interconnection network.

Before the computation is performed the processor is configured using parameters containing size and dimension information. The interface uses this information to perform the initial permutation, P_d, and the remaining

PEs use the parameters to configure their address generators and the computation units.

After the board is configured, data is downloaded and distributed to the memories of the PEs. Computation is performed on single-precision complex data. The computation is divided into 2 phases: "local" and "remote". During the local phase all data are in the local memory modules. During each stage of the the remote phase, half of data is available in local memory while the other half must be obtained over the interconnect. Before each remote stage the interconnect must be configured corresponding to the permutation at that stage.

The address generator in each PE generates a sequence of addresses. The controller uses the address to read the data which is sent to the computation unit via a FIFO while the address is put in a FIFO for the writing. The computation unit includes the twiddle factor generator and arithmetic units for performing butterfly operations. The computation unit is implemented using pipelining so that an output is ready every other clock cycle as long as inputs are fed continuously. The implementation requires one pipelined floating-point adder and one pipeline floating-point multiplier. The output from the computation unit is written back to memory at the same address stored in the FIFO.

Special properties of the permutations in the optimal dataflow allow address generation to use the adder shown in Figure 5. The base address (INT), and increment (INC) are computed from the stage number and the number of points (see [1]).

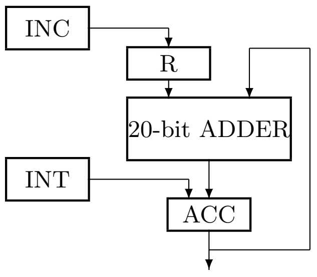

Figure 5: Adder used for generating address sequence

6 Summary and Future Work

In this paper we have presented a family of algorithms, called dimensionless FFTs, for computing multidimensional FFTs. We also introduced a parameterized family of FFT processors and showed how to map algorithms onto this hardware design. Finally, using a performance model we were able to find the optimal FFT algorithm for the architecture we considered. A prototype implementation of the optimal algorithm and resulting hardware design was carried out using FPGA and the resulting board configuration was validated. Currently we are in the process of benchmarking our implementation and comparing it to other FFT processors. In the future we hope to extend this methodology to a larger class of algorithms and processor designs.

References

[1] P. Kumhom, J. R. Johnson and P. Nagvajara, "Design, optimization, and implementation of a universal FFT engine," Tech. Rep. DU-MCS-00-01, Drexel University, 2000, `http://www.mcs.drexel.edu`.

[2] J. M. F. Moura, J. Johnson, R. Johnson, D. Padua, V. Prasanna, and M. M. Veloso, "SPIRAL: Portable Library of Optimized SP Algorithms," 1998, `http://www.ece.cmu.edu/~spiral/`.

[3] L. Auslander, J. R. Johnson and R. W. Johnson, "Dimensionless fast Fourier transforms," Tech. Rep. DU-MCS-97-01, Drexel University, 1997, `http://www.mcs.drexel.edu`.

[4] C. Van Loan, *Computational Framework for the Fast Fourier Transform*, SIAM, Philadelphia, PA, 1992.

[5] J. R. Johnson and R. W. Johnson, "Distributed memory FFT algorithms and dataflow," in *Proc. of 1999 High Performance Embed Computing Conference (HPEC99)*, MIT Lincoln Lab, Cambridge, MA, September 1999.

[6] "Unified modeling (UM) reference manual (ADEPT Version A.1)," Tech. Rep. 960620.0, CSIS, University of Virginia, 1996.

[7] Annapolis Micro Systems, Inc., *WILDFORCE Reference Manual Revision 3.4*, 1999.

[8] J.R. Johnson, R.W. Johnson, D. Rodriguez, and R. Tolimieri, "A methodology for designing, modifying, and implementing fourier transform algorithms on various architecture," *Circuit, Systems, and Signal Processing*, vol. 9, no. 4, pp. 249–500, 1990.

[9] M. C. Pease, "An adaptation of the fast fourier transform for parallel processing," *ACM*, vol. 15, no. 2, pp. 252–264, April 1968.

[10] R.W. Hartenstein, A.G. Hirschbiel, and M. Weber, "Xputer-an open family of non von neuman architecture," Tech. Rep. 195/89, University of Kaiserslautern, 1989.

[11] "RASSP Hardware/Software Codesign Application Note," `http://www.atl.external.lmco.com`.