Extended Java-FFT Environment

Alberto Quinchanegua, Iván García Alsina
Advisor: Domingo Rodríguez

Electrical and Computer Engineering Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
albertoq@ece.uprm.edu, ivangarcia44@hotmail.com

Abstract

This document presents a signal processing tool called Java–FFT for the codification of Kronecker products factors, called functional primitives, in language C. We present mathematical preliminaries for the addition of new functional primitives to a tool environment, which assist a developer in the generation of new algorithms based on Kronecker products algebra formulation. This tool is made to aid the analysis, design, modification, and implementation of Fast Fourier Transform (FFT) algorithms. The tool was written as Java applets, so it can be used with Internet browsers. The tool will be modified to include specifically two new functional primitives in the user interface. In the new version, the tool will generate C code that run efficiently on personal computers (PCs) by exploiting their architecture features.

1. Introduction

The Java-FFT tool is an environment for automatic C-language code generation for Fast-Fourier Transform (FFT) algorithms. The tool is written in Java applets so it can be accessed trough the Internet or stand-alone. The purpose of developing this tool was to aid the analysis, design, modification and implementation of FFT algorithms. The Java-FFT environment uses Kronecker based formulations to generate C-language code. These formulations are composed of functional primitive operators. The **Figure 1** diagram shows the steps of developing a program using the Java-FFT environment. The tool was made because generating manually Kronecker based formulations was too difficult and expensive. The Java-FFT tool implements the Kronecker User Interface Language Tool (KUILT) to facilitate this process. **Figure 2** shows the role of the KUILT in the Application development environment of the Computational Signal Processing Group. Actually the KUILT generates C code, but in the future it will generate code for Java. To generate source code in the Java-FFT environment tool the following steps are made: The algorithm is expressed in the matrix-vector multiplication form. The matrix is factored into Kronecker products of functional primitives. Finally, the Java-FFT tool helps to identify the most useful functional primitives and automatically generates the C code.

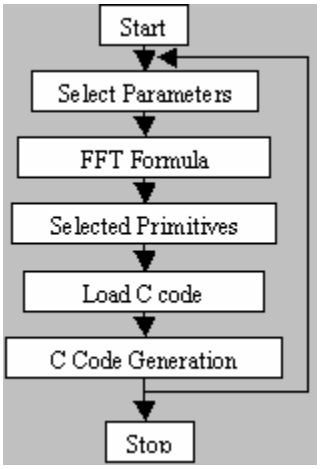

Figure 1. C code generation process.

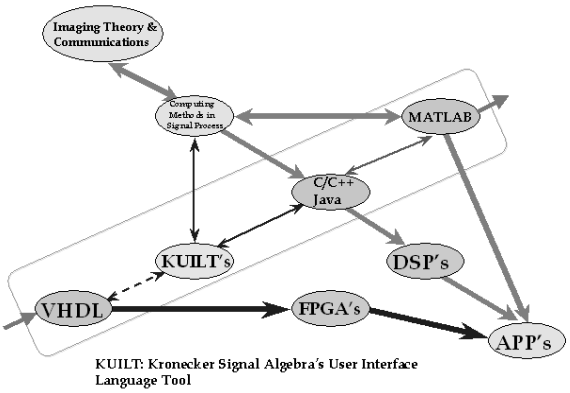

Figure 2. An example of a Simulink® model composed of interconnected functional blocks.

In order to introduce the Java FFT tool we present the mathematical formulations for the FFT, using a Kronecker language, and then we describe the Java FFT environment.

2. Mathematical Formulations

In this section a mathematical formulation of the FFT is presented using a Kronecker products algebra technique. First some definitions are presented before description of FFT Kronecker formulation.

2.1 Kronecker Product

The Kronecker product between the matrices A and B is defined as $A \otimes B = \lfloor a_{(i,j)} \cdot B \rfloor$; thus,

$$A \otimes B = \begin{bmatrix} a_{(0,0)}B & \cdots & a_{(0,s-1)}B \\ a_{(1,0)}B & \cdots & a_{(1,s-1)}B \\ \vdots & \ddots & \vdots \\ a_{(r-1,0)}B & \cdots & a_{(r-1,s-1)}B \end{bmatrix} \quad (1)$$

where $a_{(i,j)}$ is the entry in the *i*-th row and *j*-th colum of the matrix

$$A = \lfloor a_{(i,j)} \rfloor_{\substack{0 \le i < r \\ 0 \le j,s}}$$

$$A \otimes B = C = \lfloor a_{(p,q)} \rfloor \otimes \lfloor b_{(r,s)} \rfloor = \lfloor c_{(t,u)} \rfloor, \quad (2)$$

where $c_{(t,u)} = c_{(p,r;q,s)} = a_{(p,q)} \cdot b_{(r,s)}$. Define the $m \times 1$ vector U_m as $U_M = [u_1, u_2, u_3, \cdots, u_M]^T$, where $u_0 = u_2 = u_3 = \cdots = u_{m-1} = 1$. Define I_M as the identity matrix of order M. Thus:

$$(I_M \otimes F_N) = diag\,[F_{N,0}, F_{N,1}, \cdots, F_{N,M-1}] \quad (3)$$

$$(U_M^T \otimes I_N) = [I_{N,0}, I_{N,1}, \cdots, I_{N,M-1}] \quad (4)$$

2.2 Kronecker Formulation of the FFT

The traditional Cooley-Tukey algorithm for the fast computation of the discrete Fourier transform (DFT) can be represented in terms of Kronecker product factors as follows:

$$F_N = (F_R \otimes I_S) T_{N,S} (I_R \otimes F_S) P_{N,R} \quad (5)$$

where *N* is the size of the FFT, *N=R.S*, and $P_{N,R}$ is an associated permutation matrix called stride permutation. The expressions (2) allow us to identify functional primitives or basic structures that compose an algorithm. A functional primitive can be modeled as a flow graph using the mathematical expression to define the dataflow for the algorithm computation. For example, consider the Kronecker factor $(I_2 \otimes F_2)$ that represents a first stage in the 4-point FFT computation. Graphically the data flow generated by the action of the Kronecker factor over an input vector of length *N=4* can be determined as follow:

$$(I_2 \otimes F_2) \cdot x = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes F_2 \right) \cdot x$$

$$(I_2 \otimes F_2) \cdot x = \begin{bmatrix} F_2 & 0 \\ 0 & F_2 \end{bmatrix} \cdot x \quad (6)$$

where $F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

The expression (6) below represents the parallel action of matrices F_2 over segments of the input vector x. A diagonal matrix $T_{N,S}$ contains the twiddle factors.

2.3 Flow diagram of a Functional Primitive.

Each functional primitive is associated with a flow diagram, simulating the algorithm data flow in every stage of the FFT computation. The initial stage of the FFT represented by the functional primitive $(I_2 \otimes F_2)$ can be implemented as a butterfly stage as shows the **Figure 3**.

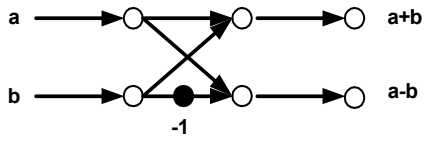

Figure 3. Data flow diagram of a single Butterfly computation.

Other example of the data flow representation for the computation of the functional primitive $(U_M^T \otimes I_N)$ is shown in the **Figure 4.**

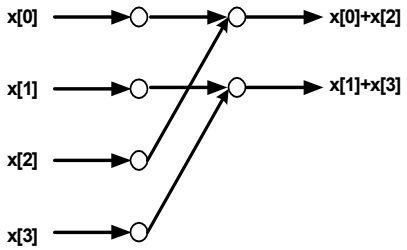

Figure 4. Data flow diagram for the computation of the Kronecker factor $(U_M^T \otimes I_N)$.

This Kronecker factor represents a linear combination between elements of an input vector, taking the vector indices as reference to obtain the sum of the odd and the even elements separately. The elements of an input

vector can be readdressed using a permutation matrix in order to generate a new parallel expression for this linear combination as follow:

$$\left(U_M^T \otimes I_N\right) = \left(I_N \otimes U_M^T\right) P_{MN,N} \quad (7)$$

3. Java-FFT Environment

The tool was implemented in java because of the portability of the language. The Java-FFT tool could be used in a Java enabled browser, such as Netscape, Microsoft Internet Explorer, and Hot Java. Java applets where used to make the tool accessible through the Internet. Java applets are applications made to run in Internet browsers. The Java-FFT tool needs Java 2 SDK to run.

3.1 Java-FFT User Interface

The Java-FFT user interface is designed to enter the Kronecker based formulation and all related parameters. The user interface has two modes: analysis and synthesis. In the analysis (default) mode the Kronecker formulation is predefined by Cooley-Tukey as follows:

$$F_N = (F_S \otimes I_R) T_{N,S} (I_S \otimes F_R) P_{N,S} \quad (8)$$

The radix parameter is used to select the size of the basic Fourier transform factor. In the synthesis mode, shown in **Figure 5**, the user uses primitive operators to form the Kronecker formulation. The primitive operator buttons can only be seen in this mode. After the formulation is entered the user presses the See Code button to display the generated code in a new window.

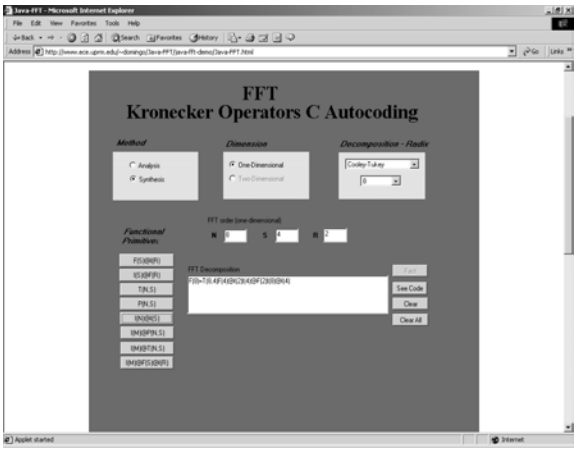

Figure 5. Java-FFT user interface (Synthesis mode).

3.2 User Interface Control Components:

- **Method Selection**: This control has two options: analysis and synthesis.
- **Primitive Operator Buttons**: They are used to add Kronecker primitive operators in the formulation text area.
- **Kronecker Based Formulation Text Area**: A text area shows the current combination of Kronecker primitive operators.
- **FFT Order Parameters**: The FFT order parameters are entered in text boxes. These are N, S, and R for the one-dimension case and N1, S1, R1, N2, S2, and R2 for the two-dimension case.
- **Dimension Selection**: Selection of one or two dimensional FFT algorithms.
- **Decomposition-Radix Selection**: Currently Cooley-Tukey is the only decomposition method, since it is highly used and it is very efficient. The user can also select from a drop-down menu the radix.
- **Fact Button**: Generates the Kronecker based formulation using Cooley-Tukey and the selected radix. This button can only be seen in Analysis mode.
- **See Code Button**: This button is used to generate the code in C using the Kronecker formulation specified in the text area. A new window is opened to show the code. See **Figure 4**.
- **Clear Button**: Used to clear the Kronecker formulation.
- **R and S Text Areas**: Display the factorization used. These text areas can only be seen in Analysis mode.

3.3 Code Generation

To generate the code, the environment creates a vector that contains the functional primitives that form the Kronecker based formulation. The elements of this vector are obtained by parsing the string entered in the formulation text area. The vector is used then to generate the C code. The first step in generating the code is to place a fixed header. Then the headers for each function used are defined. The first code included inside the main function is the declaration of the variables used to store the matrices used in the formulation. The matrices are then initialized and the functional primitive's matrices are initialized using the respective function. The matrices then are multiplied according to the order specified in the formulation. The final matrix is then printed. The functions used to declare the functional primitives and multiply matrices are declared at the beginning of the code and their code is placed at the end of the file. The code of these functions is taken from external files. When using an Internet browser, these files are retrieved from a

database through a web server. An example of the C code generation is shown in the **Figure 6.**

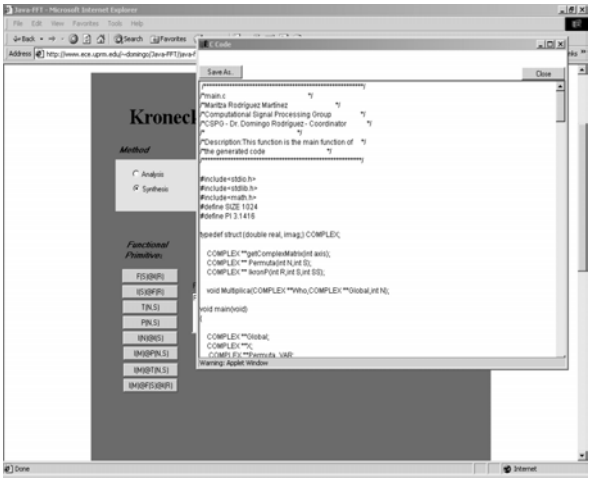

Figure 6. C code generated by the Java-FFT tool.

3.4 Additions to the Java-FFT Environment

Two new functional primitives are going to be added to the Java-FFT environment. They are the Kronecker factors $\left(U_M^T \otimes I_N\right)$ and $\left(I_N \otimes U_M^T\right)$ presented previously. These factors are useful for the development of new signal processing algorithms for the DFT Multi-Beamforming processing. To add these functional primitives the first step is to create two additional buttons for the synthesis mode. The code for the Java-FFT tool has appending and parsing sections for functional primitives, which are similar for all of them. With careful study of this repeated code we can add the new functional primitives. We also need to create two additional C-language code files that include the functions that create the functional primitives that are going to be added.

Another goal that we have is to make that the C code generated by the Java-FFT tool execute efficiently in personal computers (PCs). To achieve this we will modify the code to make it exploit all features of the architecture of PCs, such as a cache memory.

4. Conclusion

In this paper, we presented a Java-based programming tool for the automatic generation of C code for the FFT algorithms based on Kronecker products algebra formulations. Since this tool was written as a Java applet it can be accessed through the Internet. The tool is going to be modified to include two new functional primitives in the user interface and to generate code that efficiently run in PCs.

5. Future Work

The following are modifications that we are considering to make to the Java-FFT tool:

- Modify the user interface to improve its usability and robustness.
- Allow the user to add custom functional primitives.
- Make the Java-FFT tool generate Java and C# code in addition to the C code.

References

[Johnson90] J. R. Johnson, R. W. Johnson, D. Rodríguez, R. Tolimieri, "A Methodology for Designing, Modifying, and Implementing Fourier Transform Algorithms on Various Architectures," *IEEE Transactions on Circuits, Systems, and Signal Processing*, Vol. 9, No. 4, 1990.

[Rodriguez02] D. Rodríguez, A. Quinchanegua, H. Nava. "Signal Operator Cores for SAR Real Time Processing Hardware," *IEEE IGARSS 2002*, Toronto, Canada, July 2002.

[Rodriguez02] D. Rodríguez, D. Rueda, H. Nava, A. Quinchanegua. "High Performance SAR Raw Data Generation Algorithms for Remote-sensed Imaging Applications," *IEEE IGARSS 2002*, Toronto, Canada, July 2002.

[Rodriguez91] Domingo Rodriguez, " Tensor product algebra as a tool for VLSI implementation of the discrete Fourier transform," ICASSP '91, Vol. 2, pp.1025-1028. May 1991.

[Rodriguez00] M. Rodríguez, D. Rodríguez, "Java-Based Tool Environment for Automatic Multidimensional FFT Code Generation," *International Conference on Signal Processing, Applications and Tecnology*, Orlando, Florida, Nov. 1999.