A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems

P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck,
C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare
A. Nayak, S. Periyacheri, M. Walkden, D. Zaretsky
Electrical and Computer Engineering
Northwestern University
2145 Sheridan Road
Evanston, IL-60208
banerjee@ece.northwestern.edu

Abstract

Recently, high-level languages such as MATLAB have become popular in prototyping algorithms in domains such as signal and image processing. Many of these applications whose subtasks have diverse execution requirements, often employ distributed, heterogeneous, reconfigurable systems. These systems consist of an interconnected set of heterogeneous processing resources that provide a variety of architectural capabilities. The objective of the MATCH (MATlab Compiler for Heterogeneous computing systems) compiler project at Northwestern University is to make it easier for the users to develop efficient code for distributed, heterogeneous, reconfirgurable computing systems. Towards this end we are implementing and evaluating an experimental prototype of a software system that will take MATLAB descriptions of various applications, and automatically map them on to a distributed computing environment consisting of embedded processors, digital signal processors and field-programmable gate arrays built from commercial off-the-shelf components. In this paper, we provide an overview of the MATCH compiler and discuss the testbed which is being used to demonstrate our ideas of the MATCH compiler. We present preliminary experimental results on some benchmark MATLAB programs with the use of the MATCH compiler.

1 Introduction

A distributed, heterogeneous, reconfigurable computing system consists of a distributed set of diverse processing resources which are connected by a high-speed interconnection network; the resources provide a variety of architectural capabilities and are coordinated to perform an application whose subtasks have diverse execution requirements. One can visualize such systems to consist of embedded processors, digital signal processors, specialized chips, and field-programmable gate arrays (FPGA) interconnected through a high-speed interconnection network; several such systems have been described in [9].

A key question that needs to be addressed is how to map a given computation on such a heterogeneous architecture without expecting the application programmer to get into the low level details of the architecture or forcing him/her to understand the finer issues of mapping the applications on such a distributed heterogeneous platform. Recently, high-level languages such as MATLAB have become popular in prototyping algorithms in domains such as signal and image processing, the same domains which are the primary users of embedded systems. MATLAB provides a very high level language abstraction to express computations in a functional style which is not only intuitive but also concise. However, currently no tools exist that can take such high-level specifications and generate low level code for such a heterogeneous testbed automatically.

The objective of the MATCH (MATlab Compiler for distributed Heterogeneous computing systems) compiler project [3] at Northwestern University is to make it easier for the users to develop efficient code for distributed heterogeneous computing systems. Towards this end we are implementing and evaluating an experimental prototype of a software system that will take MATLAB descriptions of various embedded systems applications, and automatically map them on to a heterogeneous computing environment consisting of field-programmable gate arrays, embedded processors and digital signal processors built from commercial off-the-shelf (COTS) components. An overview of

0-7695-0871-5/00 $10.00 © 2000 IEEE

Ex.1148 / IPR2025-00914 / Page 1 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

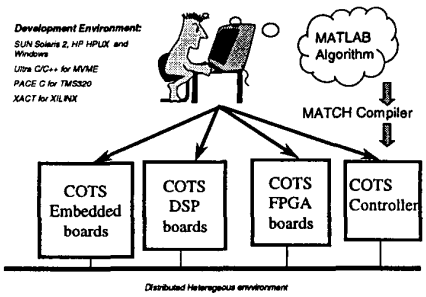

Figure 1. A graphical representation of the objectives of our MATCH compiler.

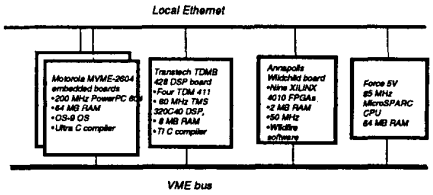

Figure 2. Overview of the Testbed to Demonstrate the MATCH Compiler

the easy-to-use programming environment that we are trying to accomplish through our MATCH compiler is shown in Figure 1. The goal of our compiler is to generate efficient code automatically for such a heterogeneous target while optimizing two objectives: (1) Minimizing resources (such as type and number of processors, FPGAs, etc) under performance constraints (such as delays, and throughput) (2) Maximizing performance under resource constraints.

The paper is organized as follows. Section 2 provides an overview of the testbed which is being used to demonstrate our ideas of the MATCH compiler. We describe the various components of the MATCH compiler in Section 3. We present preliminary experimental results of our compiler in Section 4. We compare our work with other related research in Section 5, and conclude the paper in Section 6.

2 Overview of MATCH Testbed

The testbed that we have designed to work with the MATCH project consists of four types of compute resources. These resources are realized by using off-the-shelf boards plugged into a VME cage. The VME bus provides the communication backbone for some control applications. In addition to the reconfigurable resources, our testbed also incorporates conventional embedded and DSP processors to handle the special needs of some of the applications. Real life applications often have parts of the computations which may not be ideally suited for the FPGAs. They could be either control intensive parts or could be even complex floating point applications. Such computations are performed by these embedded and DSP processors. An overview of the testbed is shown in Figure 2.

We use an off-the-shelf multi-FPGA board from Annapolis Microsystems [2] as the reconfigurable part of our testbed. This $WildChild^{TM}$ board has 8 Xilinx 4010 FPGAs (each with 400 CLBs, 512KB local memory) and a Xilinx 4028 FPGAs (with 1024 CLBs, 1MB local memory). A Transtech TDM-428 board is used as a DSP resource. This board has four Texas Instruments TMS320C40 processors (each running at 60MHz, with 8MB RAM) interconnected by an on board 8 bit wide 20MB/sec communication network. The other general purpose compute resource employed in the MATCH testbed is a pair of Motorola MVME2604 boards. Each of these boards hosts a PowerPC-604 processor (each running at 200 MHz, with 64 MB local memory) running Microware's OS-9 operating system. These processors can communicate among themselves via a 100BaseT ethernet interface.

A Force 5V board with MicroSPARC-II processor running Solaris 2.6 operating system forms one of the compute resources that also plays the role of a main controller of the testbed. This board can communicate with other boards either via the VME bus or via the Ethernet interface.

3 The MATCH Compiler

We will now discuss various aspects of the MATCH compiler that automatically translates the MATLAB programs and maps them on to different parts of the target system shown in Figure 2. The overview of the compiler is shown in Figure 3.

MATLAB is basically a *function oriented* language and most of the MATLAB programs can be written using predefined functions. These functions can be primitive functions or application specific. In a sequential MATLAB program, these functions are normally implemented as se-

Ex.1148 / IPR2025-00914 / Page 2 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

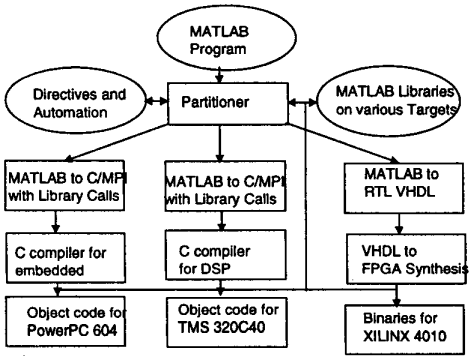

Figure 3. The MATCH Compiler Components

quential code running on a conventional processor. In the MATCH compiler however, these functions need to be implemented on different types of resources (both conventional and otherwise) and also some of these need to be parallel implementations to take best advantage of the parallelism supported by the underlying target machine.

MATLAB also supports language constructs using which a programmer can write conventional procedural style programs (or parts of it) using loops, vector notation and the like. Our MATCH compiler needs to automatically translate all such parts of the program into appropriate sequential or parallel code.

MATLAB being a *dynamically typed* language, poses several problems to a compiler. One of them being the well known *type inferencing problem*. The compiler has to figure out not only whether a variable was meant to be a floating point variable, but also the number of dimensions and extent in each dimension if the variable happens to be an array. For example, when the compiler sees a MATLAB statement `a = b * c`, it might mean one of several things: `a, b, c` are scalar variables (either integer, or short, or float, or double-precision); or `a` can be a one-dimensional vector, `b` can be a two-dimensional matrix, and `c` can be a one-dimensional vector; or `a, b, c` can be two-dimensional matrices; or `a, b` can be matrices, and `c` can be a scalar; or `a, c` can be matrices, and `b` can be a scalar. Clearly, when a compiler has to generate the code, the correct type needs to be declared or inferred by the compiler. Our compiler provides mechanisms to automatically perform such inferencing which is a crucial component of any compiler for a dynamically typed language. We have developed a shape algebra which can be used to infer types and shapes of variables in MATLAB programs [6].

Often, it may not be possible to infer these attributes, in which case our compiler takes the help of user *directives* which allow the programmer to explicitly declare the type/shape information. The directives for the MATCH compiler start with *%!match* and hence appear as comments to other MATLAB interpreters/compilers. An extensive set of directives have been designed for the MATCH compiler [3].

The maximum performance from a parallel heterogeneous target machine such as the one shown in Figure 2 can only be extracted by efficient mapping of various parts of the MATLAB program onto the appropriate parts of the target. The MATCH compiler incorporates automatic mechanisms to perform such mapping. It also provides ways using which an experienced programmer well versed with the target characteristics can guide the compiler to fine tune the mapping in the form of directives.

3.1 Compilation Overview

The first step in producing parallel code from a MATLAB program involves parsing the input MATLAB program based on a formal grammar and building an abstract syntax tree. After the abstract syntax tree is constructed the compiler invokes a series of *phases*. Each phase processes the abstract syntax tree by either modifying it or annotating it with more information.

Using rigorous data/control flow analysis and taking cues from the programmer directives (explained in Section 3.5.2), this AST is partitioned into one or more sub trees. The nodes corresponding to the predefined library functions directly map on to the respective targets and any procedural style code is encapsulated as a user defined procedure. The main thread of control is automatically generated for the Force V processor which keeps making remote procedure calls to these functions running on the processor (or processors) onto which they are mapped.

In the following sections we go into the details of these aspects of the MATCH compiler.

3.2 MATLAB Functions on FPGAs

In this section we describe our effort in the development of various MATLAB libraries on the Wildchild FPGA board described earlier. These functions are developed in Register Transfer Level (RTL) VHDL using the Synplify logic synthesis tool from Synplicity to generate gate level netlists, and the Alliance place-and-route tools from Xilinx. Some of the functions we have developed on the FPGA board include matrix addition, matrix multiplication, one dimensional FFT and FIR/IIR filters [4]. In each case we have de-

Ex.1148 / IPR2025-00914 / Page 3 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

Table 1. Performance characterization of the MATLAB matrix multiplication function on the WILDCHILD FPGA board. Times are in milli-seconds.

Matrix Size	Config Time	Download + Readback	Compute FPGA	Compute host
64 x 64	2620	31+7=38	1.95	300
128 x 128	2620	58+24=82	15	2380
248 x 248	2620	155+91=246	103	17702
496 x 496	2620	603+358=961	795	142034

veloped C program interfaces to our MATCH compiler so that these functions can be called from the host controller. In the next subsections we discuss the implementations of three of these functions, namely, matrix multiplication, the filter function, and the FFT.

3.2.1 Matrix Multiplication

The MATLAB function C = A * B performs the multiplication of two matrices A and B and stores the result in the matrix C. Our specific implementation of this function can be used to multiply two matrices of size up to 500X500 elements in 16-bit fixed-point fractional 2's complement format. The configuration and compute times for Matrix Multiplication are displayed in Table 1. It is evident that the data transfer and configuration times dominate the evaluation time. However, host compute time is about two orders of magnitude longer than FPGA compute time.

3.2.2 Filter Function

Filtering is one of the most common operations performed in signal processing. Most filters belong to one of two classes - FIR for Finite Impulse Response and IIR for Infinite Impulse Response filter. We have implemented the general MATLAB library function *filter(B,A,x)* [22] which applies the digital filter H(z) = B(z)/A(z) to the input signal x. Our specific implementation of the filter function allows a maximum order of 64 on vectors of maximum size 250,000 elements. The data precision used is 8 bits fixed-point in fractional 2's complement format. The cascaded form of the digital filter lends well to implementation on the multi-FPGA architecture of the WILDCHILD system due to the presence of near-neighbor communication capability via the systolic bus. Several FPGAs can be strung together in series to implement the required filter operation. The performance characteristics of this filter implementation for various number of taps and various data sizes is shown in Table 2. These characterizations are used by the automated mapping algorithm of the MATCH compiler described in Section 3.6.1.

Table 2. Performance characterization of the MATLAB filter function on the Wildchild FPGA board of the MATCH testbed. Runtimes in milli-seconds are shown for various number of taps and various data sizes.

Filter Taps	Vector Size	Config Time	Download + Readback	Compute
16	16K	2600	132+15=147	3
16	64K	2600	188+58=246	13
16	256K	2600	440+230=670	52
64	16K	2600	132+15=147	13
64	64K	2600	188+58=246	52
64	256K	2600	440+230=670	210
256	16K	2600	132+15=147	52
256	64K	2600	188+58=246	210
256	256K	2600	440+230=670	840

Table 3. Performance characterization of the MATLAB FFT function on the Wildchild FPGA Board. Times in milli-seconds

FFT Size	Config Time	Download + Readback	Compute FPGA	Compute host
128	3656	105+46=151	0.050	510
256	3656	106+47=153	0.130	1111
512	3656	107+48=155	0.290	2528
1024	3656	109+48=157	0.640	5624

3.2.3 Fast Fourier Transform

The discrete Fourier transform (DFT) algorithm is used to convert a digital signal in the time domain into a set of points in the frequency domain. The MATLAB function *fft(x)* computes the frequency content of a signal x and returns the values in a vector the same size as x.

Our specific implementation of the FFT function can compute the Fast Fourier Transform of up to 1024 points where each point is a complex number with real and imaginary parts in 8-bit fixed-point fractional 2's complement format. The FFT operation exhibits a high level of parallelism during the initial stages of computation but communication between processors becomes high during the final stages. The Fast Fourier Transform was run at 8 MHz and results are displayed in Table 3.

3.3 MATLAB Functions on DSPs

In this section we will describe our effort in the development of various MATLAB library functions on the DSPs. These functions are developed on the Transtech DSP boards

Ex.1148 / IPR2025-00914 / Page 4 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

Table 4. Performance characterizations of the MATLAB matrix multiplication function on the Transtech DSP board for various matrix sizes and processor configurations. Runtimes are in seconds.

	Execution Time(in secs)					
Size	1x1	1x2	2x1	2x2	1x4	4x1
16 x 16	0.004	0.005	0.005	0.005	0.005	0.005
64 x 64	0.096	0.067	0.071	0.051	0.068	0.076
128 x 128	0.639	0.378	0.391	0.242	0.279	0.304
256 x 256	3.66	2.52	2.57	1.44	1.527	1.611

utilizing multiple DSPs using message-passing among multiple processors in C using our own custom implementation of MPI. We subsequently used the PACE C compiler from Texas Instruments to generate the object code for the TMS320C40 processors. Our current set of functions includes real and complex matrix addition, real and complex matrix multiplication ,one and two dimensional FFT. Each of these libraries has been developed with a variety of data distributions such as blocked, cyclic and block-cyclic distributions. In the next section, we go through the implementation details of one of these functions, namely, matrix multiplication.

3.3.1 Matrix Multiplication

We have implemented the MATLAB matrix multiplication function on the Transtech DSP board containing 4 processors. We designed our matrix multiplication function to be generic enough to handle data distributed differently. We assume that for the matrix multiplication C = A * B, the matrices A and B can be arbitrarily distributed on the processors in a general block cyclic distribution. Table 4 shows the results of matrix multiplication on the Transtech DSP board. The speedup for the matrix multiplication is also around 2.54 on 4 processors for data distributed in a cyclic(4),cyclic(4) manner.

3.4 Automatic Generation of User functions

Since MATLAB allows procedural style of programming using constructs such as loops and control flow, the parts of the program written in such a style may not map to any of the predefined library functions. All such fragments of the program need to be translated appropriately depending on the target resource onto which they are mapped. As shown in Figure 3, we wish to generate C code for the DSP and embedded processors and VHDL code for the FPGAs. In most cases we need to translate them into parallel versions to take advantage of multiple resources that can exploit data parallelism.

3.4.1 Code Generation for Embedded and DSP

Our MATCH compiler generates sequential and message-passing parallel C code for the embedded and DSP processors. If the target is a single processor, then we generate scalarized C code. For example, corresponding to a MATLAB array assignment statement, the types, shapes and sizes of the variables accessed in the statement are either inferred or declared through directives; a scalarized C code is subsequently generated which has a set of nested `for` loops whose loop bounds are determined by the sizes of the arrays being referenced in the assignment statement. Detailed issues of scalarization have been reported in [6].

The particular paradigm of parallel execution that we have presently implemented in our compiler is the *single-program-multiple-data* (SPMD) model, where all processors execute the same program but on different portions of array data. The way the computations are distributed among processors by the compiler is the *owner computes rule* in which operations on a particular data element are executed by only those processors that actually "own" the data element. Ownership is determined by the alignments and distributions that the data is subjected to.

3.4.2 Code Generation for FPGAs

Each user function is converted into a process in VHDL. Each scalar variable in MATLAB is converted into a variable in VHDL. Each array variable in MATLAB is assumed to be stored in a RAM adjacent to the FPGA, hence a corresponding read or write function of a memory process is called from the FPGA computation process. Control statements such as IF-THEN-ELSE constructs in MATLAB are converted into corresponding IF-THEN-ELSE constructs in VHDL. Assignment statements in MATLAB are converted into variable assignment statements in VHDL. Loop control statements are converted into a finite state machine as shown in Figure 4.

For each loop statement, we create a finite state machine with four states. The first state performs the initialization of loop control variables and any variables used inside the loop. The second state checks if the loop exit condition is satisfied. If condition is valid, it transfers control to state 4, which is the end of the loop. If condition is not valid, it transfers control to state 3, which performs the execution of statements in the loop body. If there is an array access statement (either read or write), one needs to generate extra states to perform the memory read/write from external memory and wait the correct number of cycles.

The above steps described how VHDL code is generated on a single FPGA. When we have multiple FPGAs on

Ex.1148 / IPR2025-00914 / Page 5 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

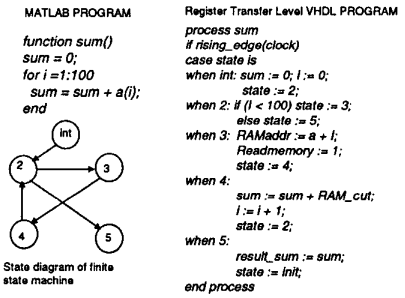

Figure 4. Example compilation of a MATLAB program with loops into RTL VHDL. On the top left we show an example MATLAB code which performs a summation of elements of an array. We show the state diagram of a finite state machine to perform the same computation in the bottom left. On the right we show the corresponding RTL VHDL code.

a board such as the WILDCHILD board, we generate the VHDL code using the owner-computes rule assuming the SPMD style of programming described in the previous section. Since the FPGAs have limited computation power, we do not have a general communication library such as MPI running on the FPGAs, but a very basic set of communication functions to perform data transfers among the FPGAs using the cross-bar network on the board.

3.5 Mapping the Program Fragments onto the Target

The MATCH compiler supports both *automated* as well as *user directed* mapping to cater to a wide range of applications. While the automated mapping is meant to produce reasonably good results with no user intervention, the user directed approach is intended to give a greater control to an experienced programmer to fine tune the program. Both approaches are complimentary and can be selectively combined in a given application.

3.5.1 Automated Mapping

When possible, the MATCH compiler tries to automatically map the user program on to the target machine taking into account the specified timing constraints, device capabilities and costs. The automatic mapping is formulated as a mixed integer linear programming problem with two optimization objectives: (1) Optimizing resources (such as type and number of processors, FPGAs, etc) under performance constraints (such as delays, and throughput) (2) Optimizing performance under resource constraints. The performance characterization of the predefined library functions and the user defined procedures guide this automatic mapping. Examples of performance characterizations are illustrated in Table 2 for the FPGA board and Table 4 for the DSP board in our MATCH testbed.

We have developed an automatic mapping tool called SYMPHANY [5] which takes as input (a) a control and data flow graph of a MATLAB program which represents various MATLAB functions as nodes (b) Characterizations of the MATLAB functions on various resources such as single or multiple FPGAs and DSP processors in terms of delays and costs (c) Performance constraints in the form of throughput and delays. The SYMPHANY tool uses a mixed integer linear programming formulation and solution for the time constrained resource optimization problem to solve the resource selection, pipelining, and scheduling problems while optimizing the resources.

3.5.2 User Guided Mapping

In cases where such an automatic mapping is not satisfactory or if the programmer is in a better position to guide the compiler, special user *directives* are provided for this purpose. These directives describe the target architectures to the compiler, the availability of predefined libraries on them and other relevant characteristics.

3.6 Final Code Generation

After generating the ASTs for each of the individual parts of the original MATLAB program, these ASTs are suitably translated for appropriate target processors. Depending on the mapping (performed as discussed in Section 3.5), the targets for each of these ASTs could be different. The ASTs corresponding to FPGAs are translated to RTL VHDL and those corresponding to Host/DSP/Embedded processors are translated into equivalent C language programs. At nodes corresponding to operators and function calls, the annotated information about the operands and the operator/function are checked. Depending upon the annotated information a call to a suitable C function is inserted that accomplishes the task of the operator/function call (pre defined or user written) in the MATLAB program. For example, to invoke the *fft* function on the cluster of DSP processors, the compiler generates a call to a *wrapper* function *fft(DSP,....)*, instead of the set of actual calls needed to invoke the fft on the DSP processors cluster. This wrapper contains the mechanism to invoke the function on respective resource. Finally, these generated programs are compiled using the respective target compilers to generate the executable/configuration bit streams.

Ex.1148 / IPR2025-00914 / Page 6 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

4 Experimental Results

We have implemented a preliminary version of the MATCH compiler. In this section we will report results of the MATCH compiler on some benchmark MATLAB programs.

4.1 Matrix Multiplication

We first report on results of the simple matrix multiplication function on various parts of the testbed. We perform the same matrix multiplication function on three targets on the testbed. The following MATLAB code represents the matrix multiplication test benchmark. We use directives to map the same matrix multiplication computation to three targets in our heterogeneous testbed. The compiler calls the appropriate library functions and the related host code.

The results of executing the code on the testbed are shown in Table 5. The column shown as "Force" refers to a matrix multiplication library running on the Force board using the RTEXPRESS library [7] from Integrated Sensors Inc., using 32 bit real numbers. The column shown as "DSP" refers to a matrix multiplication library written in C using one processor on the Transtech DSP board, using 32 bit real numbers. The column shown as "FPGA" refers to a matrix multiplication library function written in VHDL in the Wildchild FPGA board, using 8-bit fractional numbers. The numbers in parenthesis under the column for FPGA refers to the FPGA configuration and data read and write times off the board. It can be seen that even including the FPGA configuration and data download times it is faster to perform matrix multiplication on the Wildchild FPGA board. It should be noted however that the FPGA board is operating at a smaller clock cycle (20 MHz) instead of the Force board running at 85 MHz and the DSP board running at 60 MHz. However the FPGA board has more parallelism since it has 8 FPGAs working in parallel; also the data precision on the FPGA computation is only 8 bits while the Force board and DSP boards are operating on 32 bit integer numbers.

Table 5. Comparisons of runtimes in seconds of the matrix multiplication benchmark on various targets of the testbed

	Execution Time(in secs)		
Size	Force RTE	DSP	FPGA
	(85 MHz, 32 bit)	(60 MHz, 32 bit)	(20 Mhz, 8 bit)
64X64	0.36	0.08	0.002 (0.038)
128X128	2.35	0.64	0.015 (0.082)
248X248	16.48	4.6	0.103 (0.246)
496X496	131.18	36.7	0.795 (0.961)

4.2 Fast Fourier Transform

We next report on results of a one-dimensional Fast Fourier Transform function on various parts of the testbed. We perform the same FFT function on four targets on the testbed using a program similar to the previous matrix multiplication example.

The results are shown in Table 6. The column shown as "Force" refers to the FFT running on the Force board with the RTEXPRESS Library [7] from ISI, using 32 bit real numbers. The column shown as "DSP" refers to the FFT written in C using one processor on the Transtech DSP board, using 32 bit real numbers. The column shown as "FPGA" refers to the FFT written in VHDL in the Wildchild FPGA board, using 8 bit fractional numbers. It can be seen that even including the FPGA configuration and data download times it is faster to perform the FFT on the Wildchild FPGA board. It should be noted however that the FPGA board is operating at a smaller clock cycle (9 MHz) instead of the Force board running at 85 MHz and the DSP board running at 60 MHz.

Table 6. Comparisons of runtimes in seconds of the FFT benchmark on various targets of the testbed.

	Execution Time(in secs)		
Size	Force RTE	DSP	FPGA
	(85 MHz, 32 bit)	(60 MHz, 32 bit)	(9 MHz, 8 bit)
128	0.62	0.668	0.00005 (0.51)
256	2.61	3.262	0.00013 (0.51)

4.3 Image Correlation

After investigating simple MATLAB functions, we now look at slightly more complex MATLAB programs. One benchmark that we investigated is the image correlation benchmark whose code is shown below. The MATLAB program takes two 2-dimensional image data, performs a 2-dimensional FFT on each, multiplies the result and performs an inverse 2-dimensional FFT on the result, to get the correlation of two images. The MATLAB program annotated with various directives appears as follows. The type and shape directives specify the size and dimensions of the arrays. The USE directives specify where each of the library functions should be executed. It specifies that the two FFTs and the inverse FFT should be executed on the DSP board, and the matrix multiplication should be executed on

Ex.1148 / IPR2025-00914 / Page 7 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

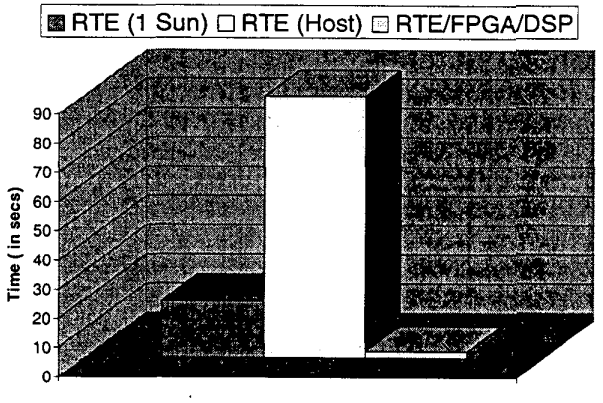

Figure 5. Performance (runtimes in seconds) of the MATCH compiler for the Image Correlation Application for am image size of 256 X 256 on various platforms.

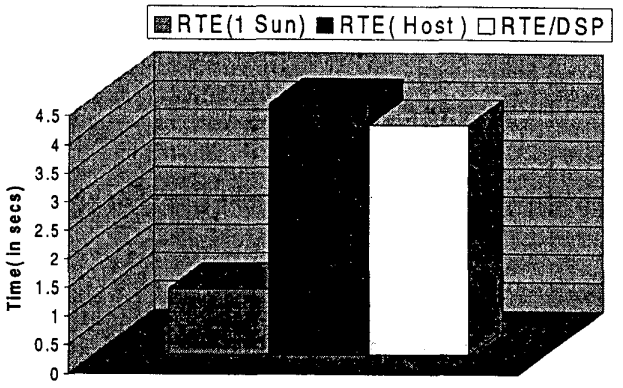

Figure 6. Performance (runtimes in seconds) of the MATCH compiler for the Pulse Compression Function from the STAP Application for am image size of 256 X 256 on various platforms.

the FPGA board.

The performance of this correlation benchmark using the MATCH compiler is shown for various platforms in Figure 5. It can be seen that the results of the RTEexpress library is faster on the SUN Ultra 5 workstation (200 MHz Ultra Sparc) over the Force board (85 MHz microsparc CPU). However, the implementation of the custom matrix multiplication library on the FPGA board and the custom FFT library function on the DSP board is much faster than the RTExpress libraries. This is because the RTExpress libraries perform a lot of memory allocations and copies of data structures before they perform the actual computations [7].

4.4 Space Time Adaptive Processing

The next benchmark we studied is the coded for Space Time Adaptive Processing. We took a MATLAB version of the STAP code from Air Force Research Labs and implemented various parts of the code on the testbed. The performance of this pulse compression function from the STAP benchmark using the MATCH compiler is shown for various platforms in Figure 6. Again, it can be seen that the results using the RTEexpress library is faster on the SUN Ultra 5 workstation (200 MHz Ultra Sparc) over the Force board (85 MHz microsparc CPU). The results using the RTExpress library on the host and FFT library on the DSP is faster than the complete implementation on the host, but not fast enough to outperform the results on the SUN Ultra 5 workstation.

5 Related Work

In this section we review related work in the area of software environments for heterogeneous computing.

5.1 High-Performance MATLAB Projects

Several projects have involved running parallel MATLAB interpreters in parallel, such as the Multi-MATLAB [23] which targeted both parallel machines such as the IBM SP2 and network of workstations, and the Parallel Toolbox [24] which runs on a network of workstations. These systems are different from our MATCH compiler in that the processors execute MATLAB interpreters, rather than compiled code.

There are several projects that involve MATLAB compilers or translators. The MCC compiler from MATHWORKS Corporation [22]. translates MATLAB into C code suitable for compilation and execution in single processors. The MATCOM compiler from MathTools [21] translates MATLAB into C++ code suitable for compilation and execution in single processors. DeRose and Padua developed the FALCON compiler [20] at the University of Illinois which translates MATLAB scripts into Fortran 90 code. Quinn et al [19] developed the OTTER compiler which translates

Ex.1148 / IPR2025-00914 / Page 8 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

MATLAB programs directly into SPMD C programs with calls to MPI. Banerjee et al extended the PARADIGM compiler which translated MATLAB programs to calls to the SCALAPACK library [26]. The RTExpress Parallel Libraries [7] from Integrated Sensors Inc. consist of efficient, parallel performance tuned implementations (using C plus MPI) of over 200 MATLAB functions.

All the above compiler projects target sequential or homogeneous parallel processors. In contrast our MATCH compiler is generating code for heterogeneous processors. Finally, they have no notion of generation of compiled code to satisfy performance or resource constraints. Our MATCH compiler tries to perform automated mapping and scheduling of resources.

5.2 Compilers for Configurable Computing

Numerous projects in configurable computing have been described in [14, 13, 12, 10, 9]. Several researchers have performed research on the development of software which can help reduce the amount of time to take a high level application and map it to a configurable computing system consisting of FPGAs.

The Cameron project [17] at Colorado State University is an attempt to develop an automatic tool for image processing applications (VSIP libraries) in Khoros. The CHAMPION project [18] at the University of Tennessee is building a library of pre-compiled primitives that can be used as building blocks of image processing applications in Khoros. The CORDS [8] project has developed a hardware/software co-synthesis system for reconfigurable real-time distributed embedded system.

There have been several commercial efforts to generate hardware from high-level languages. The Signal Processing Workbench (SPW) from the Alta Group of Cadence, translates from a block diagram graphical language into VHDL, and synthesizes the hardware. The COSSAP tool from Synopsys also takes a Block Diagram view of an algorithm and translates it to VHDL or Verilog. However, the levels that one has to enter the design in SPW or COSSAP is at the block diagram level with interconnection of blocks which resembles structural VHDL. The Renoir tool from Mentor Graphics Corporation lets users enter state diagrams, block diagrams, truth tables or flow charts to describe digital systems graphically and the tool generates behavioral VHDL/Verilog automatically. Tools such as Compilogic from Compilogic Corporation translate from C to RTL Verilog [27].

There have been several efforts at developing compilers for mixed processor-FPGA systems. The RAW project [14] at MIT exposes its low-level hardware details completely to the software system and lets the compiler orchestrate computations and data transfers at the lowest levels. The BRASS group at the University of California, Berkeley has developed the GARP [13] architecture which combined a MIPS-II processor with a fine-grained FPGA coprocessor on the same die; a C compiler for the architecture has also been developed. The Transmogrifier C compiler takes a subset of C extended with directives and compiles it to a set of Xilinx FPGAs on a board. The RAPID project at the University of Washington has developed a RAPID-C compiler [16] which compiles a language similar to C with directives onto the RAPID reconfigurable pipelined datapath architecture. The Chimaera [12] project is based upon creating a new hardware system consisting of a microprocessor with an internal reconfigurable functional unit. A C Compiler for the Chimaera architecture has been developed [28]. The NAPA-1000 Adaptive Processor from National Semiconductor [10] features a merging of FPGA and RISC processor technology. A C compiler for the NAPA 1000 architecture has been developed as well [11].

Our MATCH compiler project [3] differs from all of the above in that it is trying to develop an integrated compilation environment for generating code for DSP and embedded processors, as well as FPGAs, using both a library-based approach and automated generation of C code for the DSP and RTL VHDL code for the FPGAs.

6 Conclusions

In this paper we provided an overview of the MATCH project. As described in the paper, the objective of the MATCH (MATlab Compiler for Heterogeneous computing systems) compiler project is to make it easier for the users to develop efficient code for heterogeneous computing systems. Towards this end we are implementing and evaluating an experimental prototype of a software system that will take MATLAB descriptions of various embedded systems applications in signal and image processing, and automatically map them on to an adaptive computing environment consisting of field-programmable gate arrays, embedded processors and digital signal processors built from commercial off-the-shelf components.

7 Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Agency under contract F30602-98-2-0144 and in part by the National Science Foundation under grant NSF CDA-9703228.

References

[1] P. Joisha and P. Banerjee, "PARADIGM (version 2.0): A New HPF Compilation System," *Proc. 1999 International Paral-*

Ex.1148 / IPR2025-00914 / Page 9 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

lel Processing Symposium (IPPS'99), San Juan, Puerto Rico, April 1999.

[2] Annapolis Microsystems Inc., "Overview of the WILDFORCE, WILDCHILD and WILDSTAR Reconfigurable Computing Engines," http://www.annapmicro.com.

[3] P. Banerjee et al, *MATCH: A MATLAB Compiler for Configurable Computing* Systems." Technical Report, Center for Parallel and Distributed Computing, Northwestern University, Aug. 1999, CPDC-TR-9908-013.

[4] S. Periyayacheri, A. Nayak, A. Jones, N. Shenoy, A. Choudhary, and P. Banerjee, "Library Functions in Reconfigurable Hardware for Matrix and Signal Processing Operations in MATLAB," *Proc. 11th IASTED Parallel and Distributed Computing and Systems Conference (PDCS'99)*, Cambridge, MA, Nov. 1999.

[5] U. Nagaraj Shenoy, A. Choudhary, P. Banerjee, *Symphany: A System for Automatic Synthesis of Adaptive Systems,* Technical Report, Center for Parallel and Distributed Computing, Northwestern University, CPDC-TR-9903-002, Mar. 1999.

[6] P. Joisha, A. Kanhare, M. Haldar, A;. Nayak, U. Nagaraj Shenoy, A. Choudhary, P. Banerjee, *Array Type and Shape Analysis in MATLAB and Generating Scalarized C code* Technical Report, Center for Parallel and Distributed Computing, Northwestern University, CPDC-TR-9909-015, Sep. 1999.

[7] M. Benincasa, R. Besler, D. Brassaw, R. L. Kohler, Jr. "Rapid Development of Real-Time Systems Using RTExpress". *Proceedings of the 12th International Parallel Processing Symposium*, pages 594-599, Mar 30 - Apr 3, 1998.

[8] R. P. Dick, N. K. Jha. "CORDS: Hardware-Software Co-Synthesis of Reconfigurable Real-Time Distributed Embedded Systems". *IEEE/ACM International Conference on Computer Aided Design*, pages 62-68, San Jose, California, November, 1998.

[9] J. L. Gaudiot and L. Lombardi, "Guest Editors Introduction to Special Section on Configurable Computing," *IEEE Transactions on Computers*, Volume 48, No. 6, June 1999, pp. 553-555.

[10] C. E. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, The NAPA Adaptive Processing Architecture *Proc. IEEE Symp. on FPGAs for Custom Computing Machines*, Napa Valley, CA, Apr. 1998. *http://www.national.com/appinfo/milaero/napa1000/.*

[11] M. B. Gokhale and J. M. Stone, NAPA C: Compiling for a Hybrid RISC/FPGA Architecture, *Proc. IEEE Symp. on FPGAs for Custom Computing Machines*, Napa Valley, CA, Apr. 1998.

[12] S. Hauck, T. W. Fry, M. M. Hosler, J. P. Kao. "The Chimaera Reconfigurable Functional Unit". *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 16-18, 1997.

[13] J. Hauser and J. Wawrzynek, "GARP: A MIPS Processor with a Reconfigurable Coprocessor," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 16-18, 1997. http://http.cs.berkeley.edu/projects/brass/

[14] A. Agarwal et al, "Baring it all to Software: Raw Machines" *IEEE Computer*, September 1997, Pages 86-93.

[15] D. Galloway, "The Transmogrifier C Hardware Description Language and Compiler for FPGAs," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1995.

[16] D. Cronquist, P. Franklin, S. Berg, C. Ebeling, "Specifying and Compiling Applications for RAPID," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1998.

[17] W. Najjar et al, "Cameron Project: High-Level Programming of Image Processing Applications on Reconfigurable Computing Machines," *PACT Workshop on Reconfigurable Computing,* October 1998, http://www.cs.colostate.edu/cameron/

[18] D. Bouldin et al, "CHAMPION: A Software Design Environment for Adaptive Computing Systems" http://microsys6.engr.utk.edu/ bouldin/darpa.

[19] M. Quinn, A. Malishevsky, N. Seelam, Y. Zhao, "Preliminary Results from a Parallel MATLAB Compiler," *Proc. Int. Parallel Processing Symp. (IPPS)*, Apr. 1998, pp. 81-87.

[20] L. DeRose and D. Padua, "A MATLAB to Fortran90 Translator and its Effectiveness," *Proc. 10th ACM Int. Conf. Supercomputing (ICS)*, May 1996.

[21] Mathtools Home Page, http://www.mathtools.com

[22] MathWorks Home Page, http://www.mathworks.com

[23] A. E. Trefethen, V. S. Menon, C. C. Changm G. J. Caajkowski, L. N. Trefethen, '*Multi-MATLAB: MATLAB on Multiple Processors,* Technical Report 96-239, Cornell Theory Center, Ithaca, NY, 1996.

[24] J. Hollingsworth, K. Liu, P. Pauca, *Parallel Toolbox for MATLAB*, Technical Report, Wake Forest University (1996).

[25] S. Pawletta, W. Drewelow, P. Duenow, T. Pawletta, M. Suesse, "A MATLAB Toolbox for Distributed and Parallel Processing," *Proc. MATLAB Conference*, Cambridge, MA, 1995.

[26] S. Ramaswamy, E. W. Hodges, and P. Banerjee, "Compiling MATLAB Programs to SCALAPACK: Exploiting Task and Data Parallelism," *Proc. Int. Parallel Processing Symp. (IPPS-96)*, Honolulu, Hawaii, Apr. 1996, pp. 613-620.

[27] Compilogic Corporation, "The C2Verilog Compiler." http://www.compilogic.com.

[28] Z. A. Ye, *A C Compiler for a Processor with Reconfigurable Logic,* M.S. Thesis, Technical Report, Center for Parallel and Distributed Computing, Northwestern University, CPDC-TR-9906-009, June 1999.