

An Automated Framework for Accelerating Numerical Algorithms on Reconfigurable Platforms using Algorithmic/Architectural Optimization

Jung Sub Kim, Lanping Deng, Prasanth Mangalagiri, Kevin Irick, Kanwaldeep Sobti,
Mahmut Kandemir, Vijaykrishnan Narayanan, Chaitali Chakrabarti, Nikos Pitsianis and Xiaobai Sun

Abstract—This paper describes TANOR, an automated framework for designing hardware accelerators for numerical computation on reconfigurable platforms. Applications utilizing numerical algorithms on large-size data sets require high-throughput computation platforms. The focus is on N-body interaction problems which have a wide range of applications spanning from astrophysics to molecular dynamics. The TANOR design flow starts with a MATLAB description of a particular interaction function, its parameters, and certain architectural constraints specified through a graphical user interface. Subsequently, TANOR automatically generates a configuration bitstream for a target FPGA along with associated drivers and control software necessary to direct the application from a host PC. Architectural exploration is facilitated through support for fully custom fixed-point and floating point representations in addition to standard number representations such as single precision floating point. Moreover, TANOR enables joint exploration of algorithmic and architectural variations in realizing efficient hardware accelerators. TANOR's capabilities have been demonstrated for three different N-body interaction applications: the calculation of gravitational potential in astrophysics, the diffusion or convolution with Gaussian kernel common in image processing applications, and the force calculation with vector-valued kernel function in molecular dynamics simulation. Experimental results show that TANOR-generated hardware accelerators achieve lower resource utilization without compromising numerical accuracy, in comparison to other existing custom accelerators.

Index Terms—Algorithms implemented in hardware, Reconfigurable hardware, Signal processing systems, Numerical algorithms.



1 INTRODUCTION

TRADITIONALLY, ASICs have been used to achieve significant speedup in communication, signal processing and scientific computing. In many applications, algorithms are subject to change making reconfigurability desirable [1], [2]. Recent advances in reconfigurable hardware, especially field programmable gate arrays (FPGAs), provide a new hardware alternative for designers that combines the versatility of FPGAs with the performance advantage of ASICs [3].

Designers for digital signal processing (DSP) systems or numerical algorithms prefer to describe and prototype their designs in a high-level language such as MATLAB [4]. To realize such an application in reconfigurable hardware, the high-level or behavioral description has to be translated into a representation at the register transfer level (RTL). The first and rudimentary step in the process is the conversion of double

precision floating point (FP) MATLAB code into a fixed point (FX) or customized floating point version. Tradeoffs such as data precision, rounding modes and signed/unsigned representations are performed at the MATLAB description level. Next, hardware designers take the specifications and requirements from the developer to create a physical implementation in a hardware description language (HDL) such as VHDL or Verilog. Finally, the RTL HDL code is synthesized, placed and routed (P&R) onto an FPGA platform. The manual realization of the traditional RTL design flow tends to be tedious and error prone.

We describe in this paper, TANOR, a tool for generating accelerators for numerical computation on reconfigurable platforms [5]. It is intended for computation applications with very large sized data input, and with high accuracy and high throughput requirements. We illustrate the tool with algorithms for simulating N-body interaction problems, which have applications in molecular dynamics, celestial mechanics, plasma physics, fluid mechanics and semiconductor device characterization. TANOR is equally applicable to a wide variety of traditional DSP applications such as digital filters and transform computations including Discrete Cosine Transform (DCT) and Fast Fourier Transform (FFT). The design of TANOR is aimed at the following objectives,

- J. S. Kim, P. Mangalagiri, K. Irick, M. Kandemir and V. Narayanan are with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802. Email: jskim, mangalag, irick, kandemir, vijay@cse.psu.edu.
- L. Deng, K. Sobti and C. Chakrabarti are with the Department of Electrical Engineering, Arizona State University, Tempe, AZ 85287-5706. Email: ldeng2, ksobti, chaitali@asu.edu.
- N. Pitsianis and X. Sun are with the Department of Computer Science, Duke University, Durham, NC 27708. Email: nikos, xiaobai@cs.duke.edu.

Manuscript received July 29, 2008.

- Support for design specification of certain numerical algo-

rithms in a high-level language (MATLAB), making joint exploration of algorithmic and architectural variations feasible and accessible to application designers.

- Support for fully pipelined implementations in both floating point and fixed point formats, standard or customized.
- Optimizations with respect to area, power, accuracy and throughput in a fully automated design flow from MATLAB specification to HDL synthesis.
- Short design cycle due to development of accurate estimation models and evaluation mechanisms.

We demonstrate the effectiveness and adaptability of TANOR using three different target applications, namely, the calculation of gravitational potential in astrophysics, the diffusion or convolution with Gaussian kernel common in image processing, and the force calculation with vector-valued kernel function in molecular dynamics simulation.

The rest of the paper is organized as follows. Section 2 provides the relevant background and a brief summary of related work. Section 3 presents an overview of TANOR. Sections 4 and 5 elaborate on key hardware and software modules. Section 6 illustrates the capabilities of TANOR through design examples. Section 7 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 High Level Synthesis Flow

There are a number of existing design flows that bridge the gap between algorithm design and architecture development. An important component of these works is a seamless translation from a high level domain where algorithm designers work, to hardware (HDL based) and software (C based) specifications for a target architecture. This is essential for avoiding time consuming and error-prone manual intervention steps. Furthermore, it aids in algorithm exploration by allowing one to rapidly analyze the effects of different algorithmic parameters on system performance.

Existing automated design tools are either based on IP block or language translation. IP block based tools provide parameterizable IP libraries for common DSP functions and corresponding hardware library to which these behavioral functions can be directly mapped. For example, in [6], [7], the authors use an IP block based design environment to generate SystemC based architecture specifications, starting from a netlist of functional modules described in MATLAB. A similar methodology is presented in [1] for multiprocessor system-on-chips (MPSoC) target architectures.

There exist several commercial IP block solutions as well. FPGA vendors provide MATLAB/SIMULINK parameterizable IP libraries (known as blockset) and corresponding hardware IP libraries targeting specific FPGA devices, such as Xilinx System Generator [8]. In these design flows, a graphical design description in SIMULINK is automatically synthesized to a HDL description. However these flows have two primary limitations. First, the hardware libraries are usually vendor specific, limiting the portability to other target architectures. Secondly, they are library specific, requiring substantial manual intervention if one wishes to explore different algorithmic alternatives.

Language translation based tools, on the other hand, specify the input in high level language. Examples of such tools include "ImpulseC" [9], "CatapultC" [10] and "Cameron" [11], which map C-like script applications to FPGAs, "Calypto" [12] and "Sequential Equivalence Checking Tool" [13], which map SystemC scripts into RTL implementations by checking sequential equivalence of a system level model and its RTL version. The most prominent is perhaps AccelDSP by Xilinx, which takes input description in MATLAB. AccelDSP [14] (previously MATCH project [15] in Northwestern University, then AccelChip [16]) provides a GUI facilitated design environment to translate MATLAB input specifications into HDL. It features an automatic floating point to fixed point conversion module and integrates seamlessly with MATLAB to facilitate quick fixed point simulation. Several implementation choices are provided by the integrated IP library for non-trivial DSP functions such as FFT. Despite these features, AccelDSP has some limitations. The bit-width analysis method used is based on dynamic simulation and requires the user to provide accurate and comprehensive input test vectors in order to determine accurate bit-widths of each variable. Furthermore, it may not be suitable for high performance scientific applications due to limited I/O precision (32 bits only) and lack of floating point based datapath support. In addition, AccelDSP only generates fully flattened designs which in many cases renders the HDL code difficult to be read and changed. TANOR overcomes these limitations by automatically determining the bit-width of each variable, supporting custom fixed and floating point architectures, and finally, maintaining hierarchy in the generated HDL that is consistent with the initial MATLAB code.

2.2 Scientific Application: N-Body Problems

While TANOR supports scalarized MATLAB code for a variety of DSP algorithms, its true capabilities are manifested in large input designs with high accuracy and throughput requirements. We use N-body simulation problems to illustrate the potential.

At any time step in an N-body simulation, one deal with the interactions between a set of M source particles and a set of N target particles. We may assume $M = O(N)$. The aggregation of the interactions from all source particles to a target particle is governed by a physical law, namely, an interaction function [17]. A naive calculation of the interactions at all target particles takes $O(N^2)$ arithmetic operations. Existing accelerators include GRAPE (GRAVity PipE) project [18], MD-GRAPE [19], [20], MD Engine [21], [22]. All these hardware systems are specialized ASIC based solutions for specific N-body applications. Reconfigurable systems utilizing FPGAs have been developed in [23], [24], [25], [26], [27]. Most of them are also customized for a particular interaction function.

There is an increasing need for an automated system design and generation to support various N-body problems, as recently noted in [18]. The GRAPE PGR system [28] is one of the recent efforts in developing a design framework, but currently it does not support optimizations in the algorithm-architecture design space. Also, some early efforts made

towards modeling molecular dynamics using SRC Computers in [29], [30], involve specialized language and manual intervention in many key steps such as extracting data dependency information (used in timing analysis). In contrast, TANOR provides a fully automated framework for different N-body interaction problems. It takes algorithmic description in MATLAB as input and employs both algorithmic and architectural means to explore and optimize system design without manual intervention.

3 TANOR DESIGN FLOW

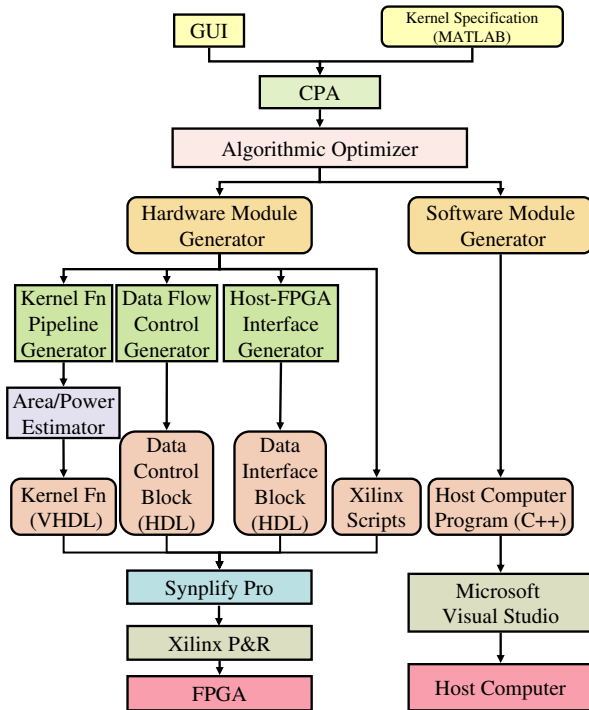


Fig. 1. Block Diagram of TANOR Design Flow

Figure 1 shows a block diagram of the design flow that is fully automated by TANOR. The input consists of an algorithmic description in MATLAB (for an N-body problem, for instance, it is the kernel function describing the interactions), a description of the data sets, optimization objectives and constraints on accuracy, latency, area, and power. It is provided through a graphical user interface (GUI). Since MATLAB is used as the specification language, we do not require a separate software emulator for functional verification of the design in the early stages of the design flow. The output is an efficient system in which the host computer interacts with the FPGA accelerator for enhancing the performance of the numerical algorithm in consideration.

The first two stages of the automated design flow are the code parser and analyzer (CPA), and the algorithmic optimizer. After these two stages, two different flows are adopted to generate hardware modules for FPGA boards and software modules for the host computer. Hardware modules are purely logic blocks composed of three different modules including kernel function, data flow control, and data interface. Hardware

modules are accompanied with Xilinx scripts to run the Xilinx synthesis and P&R tool flow without user intervention. The software module generates the host program to run the FPGA platform.

The input interface of GUI is composed of two tabs: algorithmic selection and architectural selection as shown in Figure 2. In the algorithm selection tab, a user can write the kernel function in an editor or select from a set of pre-defined kernel files. Information such as dimension, input data files, output directory and matrix tiling method (optional) can be chosen. In the architecture tab, the precision representation can be selected. The user can also specify the optimization techniques used in the algorithm optimizer stage using the GUI. Based on the settings, TANOR generates the required design automatically.

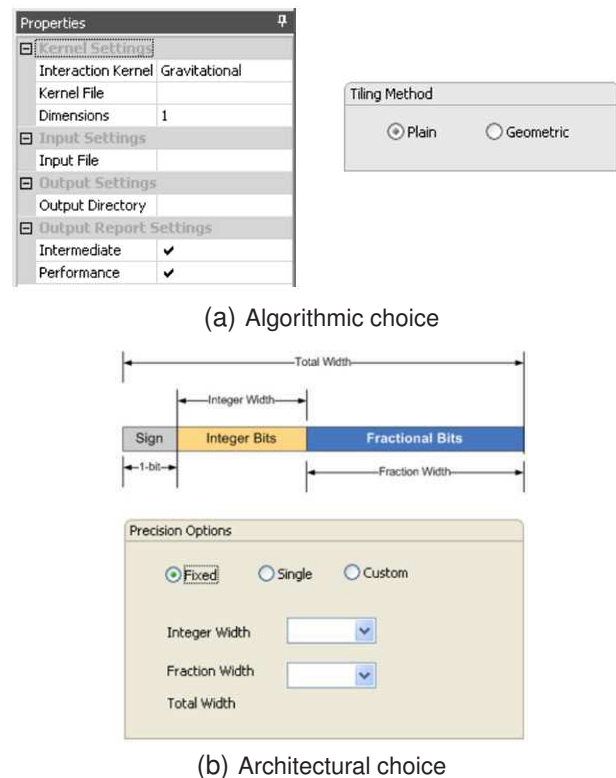


Fig. 2. Highlight of partial input interface of TANOR GUI

3.1 Code Parser and Analyzer

The first step in the TANOR design flow is the code parser and analyzer (CPA) module which processes the kernel specification along with the constraint information provided by the user. The MATLAB operations that are currently supported by TANOR are listed in Table 1. This set is compact and sufficient for supporting a variety of interaction kernels.

The code parser and analyzer phase transforms the input specifications into an intermediate representation known as ASG (Abstract Syntax Graph). Each node of this graph represents either an input or a computation, and edges capture the dependencies between them. More importantly, this graph reveals the opportunities for optimizations such as CSE (common sub-expression elimination). For functions with multiple

TABLE 1
Mathematical operations supported by TANOR

Control	For-Loop, FX and FP conversions
Arithmetic	+, -, *, reciprocal, sqrt
Memory	array indexing calls
LUT-based	Bessel, exponential, sin/cos

outputs, the common computations among the different outputs can be extracted and reused resulting in optimal resource usage. Figure 3 shows detailed operations of the code parser and analyzer.

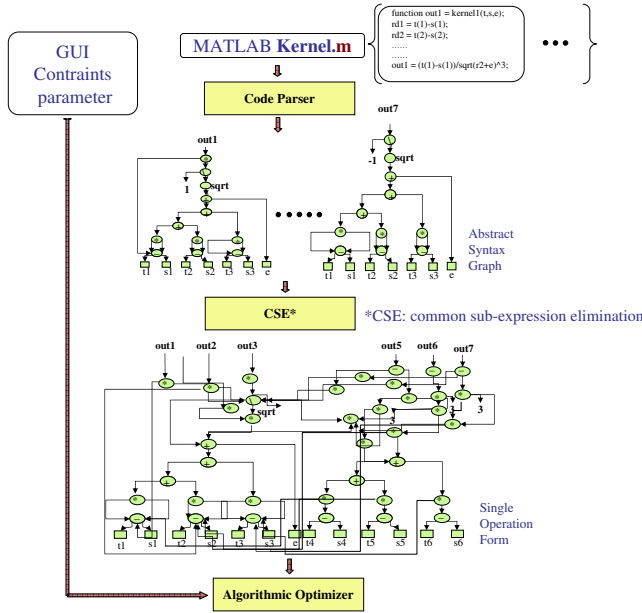


Fig. 3. The code parser and analyzer (CPA) module

3.2 Algorithm Optimizer

The algorithm optimizer operates on and transforms the ASG using one or more of the following schemes, namely, function evaluation, interaction matrix tiling or clustering, and data traversal within a cluster and across the clusters.

1. Interpolation schemes: TANOR employs a lookup table (LUT) approach for function evaluation. It considers the trade-off between the memory space and logic resources while respecting numerical accuracy requirement. In its present version, the LUT approach is based on the truncated expansion of the Taylor series of a smooth function f ,

$$T_d(x) = \sum_{k=0}^d \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k. \quad (1)$$

where x_0 is the reference or sample point closest to the evaluation point x . The Taylor polynomial coefficients at x_0 up to the d -th degree are pre-computed and stored in the LUT.

The same numerical accuracy can be achieved by using more sample points and hence a larger LUT with lower degree approximation, or a small table with higher degree approximation. Thus there is a tradeoff between the memory usage and the logic resource consumption and the scope of the tradeoff is subject to the design constraints and conditions.

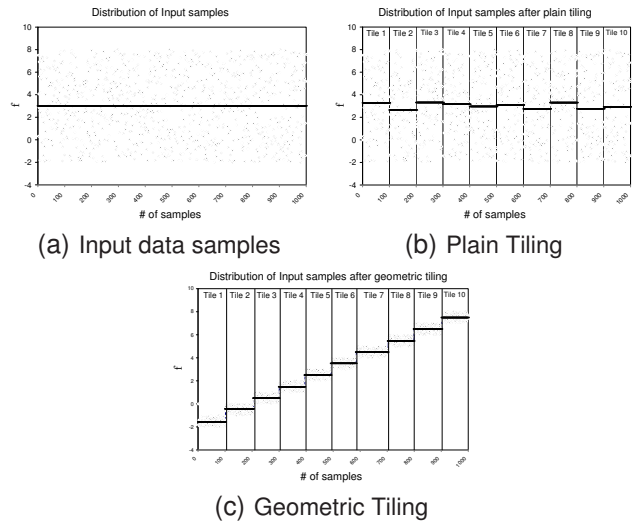


Fig. 4. Plain and geometric tiling on 1-D input. The input samples are uniformly distributed in the range [-2 8] as shown in (a). The solid line shows the mean of the samples in each tile. After applying plain tiling, the samples still have roughly the same dynamic range and mean in each tile, as shown in(b). However after applying geometric tiling, the input data has restricted dynamic range in each tile, as shown in (c).

This LUT approach has been used for evaluating trigonometric functions, square-root extraction, logarithmic and exponential function, and more complex functions such as the Bessel functions. Details can be found in [31].

2. Interaction Matrix Tiling: A tile in an interaction matrix represents the interaction between a source cluster and a target cluster. The tiling and the traversal ordering affect the accuracy, latency, as well as area and power of a hardware implementation. Currently TANOR supports two types of tiling schemes: *Plain tiling* (PT) and *Geometric tiling* (GT).

PT is a conventional matrix partitioning technique for enhanced cache performance, where the source and target data sets are partitioned into small blocks of pre-defined lengths. GT is a new technique which partitions the matrices based on their geometric and numeric structures. The partition process involves "binning" of the input samples in to different tiles based on their positions or coordinates. One can think of binning as a process in which a few significant bits of the coordinates are read and the samples are placed into the corresponding bins. Thus partitioning is an $O(N)$ operation, where N is the number of samples. For general N -body simulations, the change in particle coordinates with time is incremental, and re-partitioning is not done until the particles travel beyond the tile boundaries. Details for GT have been provided in our earlier paper [31].

Figure 4 shows the numerical ranges of the input samples in each tile after applying PT and GT. It shows that GT has the same effect in tuning cache performance as PT. In addition, it also aims at reducing the dynamic range of numerical values per tile and across tiles. This reduces the required bit-width

without compromising numerical accuracy and saving power and hardware resource consumption. A consequence of this is that higher accuracy can be achieved with reduced resources, as described later in Section 6.

3. Data Traversal: This is important in large sized N-body problems, where a summation of the force exerted on a certain target by all possible sources is performed. This requires a tile sequence to be generated in which every target tile is followed by all possible source tiles. The ordering of these tiles has a significant impact on numerical accuracy in accumulation of the computed results. When tiled intelligently, this can result in good performance with respect to accuracy while reducing the hardware resource requirement. Additional details are included in [31], [32].

3.2.1 Examples of Algorithm-Architecture Co-exploration

As a first example, we consider the calculation of the gravitational potential. We let TANOR use GT, along with a data traversal scheme in which the source tiles are first arranged in decreasing order of their distance from the target bins. This potentially allows the accumulation of data in non-decreasing order of magnitude, resulting in higher accuracy. In contrast, if PT is used, this data traversal scheme is not beneficial because the dynamic range within a tile and across tiles is roughly the same. Thus for the same accuracy, the GT-based scheme requires lower precision hardware than PT-based scheme, thereby saving both in terms of area and power, as will be demonstrated in Section 6.6.1.

For the second example, consider a case where TANOR uses a LUT-based implementation for realizing computationally intensive interaction functions. Typically LUT schemes allow limited trade-offs between logic resources and table size, which are achieved by varying the degree of the Taylor polynomial or by using a different interpolation or approximation scheme. If both GT and PT are considered on top of this, the design space becomes much larger. With GT, the dynamic range of the numerical values in each tile is restricted and thus only one segment of the table needs to be loaded for processing a tile. In contrast, if PT is used, each tile exhibits roughly the entire dynamic range and thus the complete table would have to be available [31].

After the algorithmic optimization phase, the system is partitioned into software and hardware modules, as shown in Figure 1. The task of the hardware module (see Section 4) is to generate the configuration file for programming the FPGAs. The software module (see Section 5) generates the host computer program (in C++) that interfaces the FPGA with the host computer.

4 TANOR HARDWARE MODULES

The primary objective of the hardware module is to generate an efficient accelerator that can be mapped onto an FPGA. The generated architecture is composed of three main blocks (Figure 5): Kernel function pipeline block, Data flow control block, and Host-FPGA interface block. We describe them next.

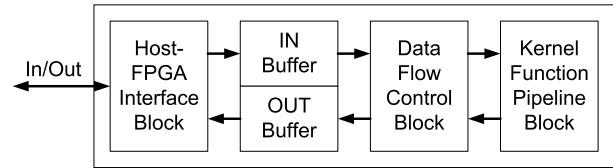


Fig. 5. Block Diagram of TANOR Architecture

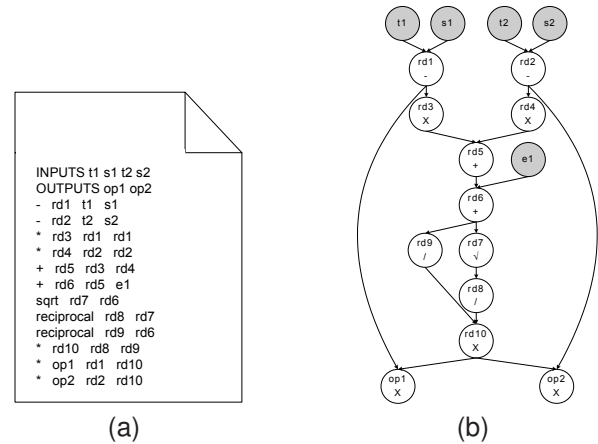


Fig. 6. Example design: (a) Input design file in SOF format. (b) Corresponding dataflow graph.

Our system is currently targeted to support Synplify Pro 8.6.2 and Xilinx ISE 8.2i as back-end tools to generate the bit file to be configured on the FPGA. Perl and Tcl scripts that are required by the back-end FPGA compiler are also automatically generated.

4.1 Kernel Function Pipeline Generator

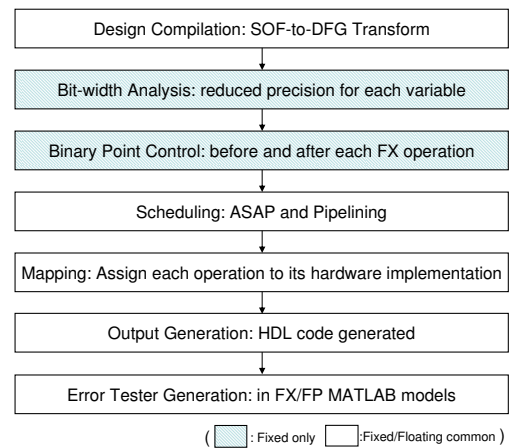


Fig. 7. Kernel Function Pipeline Generator

The Kernel Function Pipeline Generator (KFPG) block generates efficient, synthesizable HDL code for the interaction function under consideration. It supports *IEEE-754 Floating Point Standard* and custom floating point formats where the exponent and mantissa widths are user defined, as well as fixed point, where the precision of the internal variables are

TABLE 2
Interval Arithmetic calculation rules in TANOR

Operand	Example	Range Calculation
+/-	$op = in_1 \pm in_2$	$op^{lo} = in_1^{lo} \pm in_2^{lo}$, $op^{hi} = in_1^{hi} \pm in_2^{hi}$
*	$op = in_1 * in_2$	$op^{lo} = \min. \{in_1^{lo} \cdot in_2^{lo}, in_1^{hi} \cdot in_2^{lo}, in_1^{lo} \cdot in_2^{hi}, in_1^{hi} \cdot in_2^{hi}\}$, $op^{hi} = \max. \{in_1^{lo} \cdot in_2^{lo}, in_1^{hi} \cdot in_2^{lo}, in_1^{lo} \cdot in_2^{hi}, in_1^{hi} \cdot in_2^{hi}\}$
Sqrt	$op = \text{sqrt}(in_1)$	$op^{lo} = \text{sqrt}(in_1^{lo})$, $op^{hi} = \text{sqrt}(in_1^{hi})$
Recip.	$op = 1/in_1$	$op^{lo} = 1/in_1^{hi}$, $op^{hi} = 1/in_1^{lo}$
LUT	$op = \text{besselj}(in_1)$	op^{lo} and op^{hi} are explicitly specified

automatically generated. The datapath operators are listed in Table 2.

The input to the KFPG block comes from the CPA block described in Section 3.1. It is an optimized description of the interaction function in single operation form (SOF), as shown in part (a) of Figure 6. The SOF file is processed by the KFPG module to generate an HDL description that can be mapped to hardware. The key steps are depicted in Figure 7. Among these steps, bit-width analysis and binary point control is only applicable for fixed point implementation, while the other steps are common for both fixed and floating point implementations.

1) Design Compilation: The design compilation step transforms the behavioral specification of the design into an internal graph-based representation, known as data-flow graph (DFG) [33]. This process is essentially a one-to-one transformation of every operation specified in each operand of the SOF into a node of a DFG. The DFG is represented by nodes, where each $v_i \in V$ represents either an operation or an input and a set of edges $E = \{(v_i, v_j); i, j = 0, 1, \dots, n\}$. A directed edge e_{ij} from $v_i \in V$ to $v_j \in V$ exists in E if v_j is generated as a result of operating upon v_i . An example of this SOF to DFG transform is shown in Figure 6.

2) Bit-width Analysis: The goal of Bit-width Analysis is to determine the data type information for each internal variable, including the bit-width, the number type, the truncation mode and the overflow mode. This is an important optimization step because the bit-width information is used by downstream modules to determine the latency and size of hardware components for implementing the corresponding operations on these variables.

In case of floating point (FP) implementation, the user has the option of selecting the widths of mantissa and exponent fields. This choice is typically based on area and accuracy constraints of the application.

In the case of fixed point (FX) implementations, the user only specifies the dynamic range of the inputs and the maximum allowed bit-width. The Bit-Width Analysis extracts fixed point precision parameters (integer and fraction widths) for each variable. Specifically, it traverses the DFG and calculates the dynamic range of each node using one of the two range estimation techniques, namely, Interval Arithmetic (IA) and Affine Arithmetic (AA) [34], [35].

In IA analysis, each quantity x is represented by an interval $\bar{x} = [x_{lo}, x_{hi}]$. These intervals are added, subtracted, multiplied, etc., in such a way that each computed interval \bar{x} is guaranteed to contain the unknown value of the corresponding real

quantity x . Some rules used in TANOR for dynamic range calculation are shown in Table 2.

For large DFGs, IA is vulnerable to the range explosion problem. In that case, TANOR applies Affine Arithmetic (AA) [34], [35], which solves the range explosion problem by keeping track of correlations among intervals. In AA, the uncertainty of a signal x is represented by an affine form \bar{x} , which is a first-degree polynomial:

$$\bar{x} = x_0 + x_1 \xi_1 + x_2 \xi_2 + \dots + x_n \xi_n, \quad \text{where } \xi_i \in [-1, 1] \quad (2)$$

Each ξ_i is an independent uncertainty source that contributes to the total uncertainty of the signal x . Some basic arithmetic operations in AA form provided in TANOR are as follows [34]:

$$\begin{aligned} \bar{x} \pm \bar{y} &= (x_0 \pm y_0) + \sum_{i=1}^n (x_i \pm y_i) \xi_i \\ c\bar{x} &= (cx_0) + \sum_{i=1}^n (cx_i) \xi_i \\ \bar{x} \pm c &= (x_0 \pm c) + \sum_{i=1}^n (x_i) \xi_i \end{aligned}$$

Once the dynamic range for each variable is decided, TANOR applies the bit-width analysis algorithm [36] shown in Figure 8 to compute the fixed point bit-width parameters of each node in the DFG.

3) Binary point tracking: This step is used only for fixed point design. FX representation creates a virtual binary point which partitions the total bit-width into integer and fractional parts. TANOR supports varying bit-widths of signals when traversing the DFG, and so it is essential to automatically keep track of the binary point for each variable. This is done by considering bit-level effects such as bit-alignment and sign extension. For example, consider the operation $x = y + z$. If $y = a2^{-m}$, $z = b2^{-n}$ and $m < n$, then $x = (a2^{-m} + b)2^{-n}$. In other words, the fractional parts of y and z have to be matched before the integer parts can be added. This may require zeros to be appended after the LSB of the operand. Some binary point tracking rules used in TANOR are shown in Table 3 when the operand has bit-width $[I, F]$ where I and F are the integer and fractional bit-widths, respectively.

TABLE 3
Binary point tracking rules in TANOR

Operator	Inputs Bit-width	Binary point position
+/-	$[J_{in1} \ F_{in1}]$ $[J_{in2} \ F_{in2}]$	$F_{op} = \max(F_{in1}, F_{in2})$ F_{op} bits to the left of LSB
*	$[J_{in1} \ F_{in1}]$ $[J_{in2} \ F_{in2}]$	$F_{op} = F_{in1} + F_{in2}$ F_{op} bits to the left of LSB
Sqrt	$[J_{in} \ F_{in}]$	$F_{op} = J_{in}/2 + 1$, if J_{in} is even $F_{op} = (J_{in} - 1)/2 + 1$, if J_{in} is odd F_{op} bits to the right of MSB
Recipr.	$[J_{in} \ F_{in}]$	F_{in} bits to the right of MSB
LUT	The binary point position is decided by the format of lookup table, or range of data stored in the table.	

4) Scheduling: Scheduling puts a time stamp to each task in the behavioral specification, where time is measured in number of clock cycles in the case of synchronous systems. The scheduling is done in three phases. In the first phase, the latency of each operation is identified. The TANOR hardware library contains primitive hardware modules for which latency

Input: Data Flow graph ($G(V,E)$), Dynamic range for each input in interval analysis notation $inp_i = (inp_i^{lo} \ inp_i^{hi})$, user constraints on maximum bit-width (B_{max}) and maximum fractional bit-width F_{max} .

Output: Data Flow graph ($G(V,E)$) with each node annotated with its dynamic range in interval analysis notation, i.e, $x_i = (x_i^{lo} \ x_i^{hi})$ and bit-representation $QI.F$.

- 1: for all i such that $v_i \in V$ is an input do
- 2: Determine integer bit-width (X) of v_i using its dynamic range ($v_i^{lo} \ v_i^{hi}$).
- 3: Assign $I = \min(B_{max}, X)$.
- 4: Assign $F = \min(B_{max} - I, F_{max})$.
- 5: $V = V - \{v_i\}$
- 6: end for
- 7: while $V \neq \phi$ do
- 8: Select a vertex v_i whose all the predecessors have been assigned the bit-width
- 9: Compute the dynamic range ($v_i^{lo} \ v_i^{hi}$) using Table 1.
- 10: Determine integer bit-width (X) of v_i using its dynamic range ($v_i^{lo} \ v_i^{hi}$).
- 11: Assign $I = \min(B_{max}, X)$.
- 12: Assign $F = \min(B_{max} - I, F_{max})$.
- 13: $V = V - \{v_i\}$
- 14: end while

Fig. 8. Bit-width analysis algorithm in TANOR.

can be estimated as a function of the bit-widths of input and output operands, as shown in Table 4.

TABLE 4
Latency estimation for each operation in TANOR.

Operator	Data Type	Bit-width(s) of Input(s)	Bit-width of output	Latency
+/-	FX	$[I_1 \ F_1]$ $[I_2 \ F_2]$	$[I_{op} \ F_{op}]$	$L = 1$
	FP	$[e \ m]$	$[e \ m]$	$L = \begin{cases} 9 & m \leq 4 \\ 10 & 4 < m \leq 13 \\ 11 & 13 < m \leq 28 \\ 12 & 28 < m \leq 61 \\ 13 & 61 < m \end{cases}$
*	FX	$[I_1 \ F_1]$ $[I_2 \ F_2]$	$[I_{op} \ F_{op}]$	$L = 3 + c_1 + c_2$ $c_i = \lfloor \frac{I_i + F_i}{18} \rfloor, i=1,2$
	FP	$[e \ m]$	$[e \ m]$	$L = \begin{cases} 4 & m \leq 16 \\ 6 & 16 < m \leq 33 \\ 7 & 33 < m \leq 50 \\ 8 & 50 < m \leq 63 \end{cases}$
Sqrt	FX	$[I_1 \ F_1]$	$[I_{op} \ F_{op}]$	$L = 3 + c_1$ where $c_1 = F_{op} + \frac{I_1 - 1}{2}$
	FP	$[e \ m]$	$[e \ m]$	$L = m + 4$
Recipr.	FX	$[I_1 \ F_1]$	$[I_{op} \ F_{op}]$	$L = \min(36, 4 + F_1 + F_{op})$
	FP	$[e \ m]$	$[e \ m]$	$L = m + 4$
LUT	FX	$[I_1 \ F_1]$	$[I_{op} \ F_{op}]$	$L = 2$
	FP	$[e \ m]$	$[e \ m]$	

In the second phase, TANOR utilizes an "As Soon As Possible" (ASAP) scheduling scheme to generate fully pipelined implementation of the behavioral description. ASAP is a minimum latency schedule obtained by topologically sorting the vertices of the sequencing graph in depth-first order. In the third and final phase, the number of delay elements that must be introduced to synchronize the operations, are identified. This is done using lifetime analysis, where the lifetime of each variable is obtained from the scheduled DFG. This helps in the generation of a fully pipelined implementation of the behavioral description.

Figure 9 presents an example design during the scheduling step. The input DFG is obtained after bit-width analysis of the DFG described in Figure 6. Delay stages are inserted to fully pipeline the design as shown in Figure 9(b). TANOR uses custom-length shift registers to facilitate variable delay

insertion.

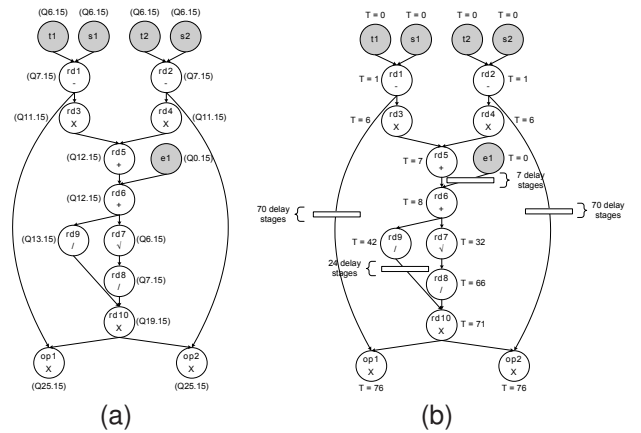


Fig. 9. Example design for scheduling: (a) Fixed point representation for each variable after bit-width analysis. Here it is shown using Q I.F notation, where I and F represents the integer and fractional bits respectively. (b) DFG after scheduling. Here the calculated delay for each operation is shown in "T", and the delay stages inserted for fully pipelining is depicted with a rectangle.

5) Mapping: In this step, each operation specified by a node (except for inputs) of a DFG is mapped to a functional unit from the list of available library components. The current list of available library components is shown in Table 5. There exists a one-to-one correspondence between each operation type and library component. For example, an add(+) operation is mapped to LogiCORE Adder/Subtractor v7.0 [37] provided by Xilinx. Lookup table based operations are mapped to BRAM's by inferring behavioral code. The delay elements that are required for synchronization, are mapped to custom-length shift-registers. The depth and width of the shift-register stages are determined by the lifetime and bit-width of the variable, respectively.

TABLE 5
List of Library Components.

Operation	Functional Unit
+/-	LogiCORE Adder/Subtractor v7.0
*	LogiCORE Multiplier v8.0
Reciprocal	LogiCORE Divider v1.0
Sqrt	LogiCORE Cordic v3.0
Floating Point Operator	LogiCORE Floating-Point Operators v3.0
LUT	Block RAMS
Round	Customized
Shift Register	Customized

6) Output Generation: The output of the design is created in a format that is easily processed by downstream synthesis, and P&R tools. We use VHDL as the output format.

7) Error Tester Generation: The final step in KFPG module is generation of MATLAB functional models both for fixed point and floating point designs. These models accurately model the behavior of the hardware system. The user can directly substitute these models in the verification environment to quickly analyze the mathematical errors that are introduced due to reduced precision of the implemented hardware kernels.

4.2 Area and Power Estimation Block

TANOR provides an area/power estimation block with the KFPG block to facilitate early design space exploration. It is used to determine whether a particular implementation will fit in the target FPGA without waiting for the time-consuming synthesis and P&R steps. Accurate area and power estimation models for implementations using Xilinx Virtex-2Pro FPGA family have been described in [38]. Area models have been derived for the library components listed in Table 5 for both fixed and floating point data. The inputs are the bit-widths of the operands and the outputs are given in terms of number of slices, block RAMs and 18x18-bit multipliers. Power models have been derived which take the area estimation results as inputs and provide estimates for logic power, signal power, clock power and I/O power. In all cases, the model coefficients have been derived by using curve fitting and non-linear regression analysis [39]. Validation of these models is shown in Section 6.6.3.

4.3 Multiply and accumulate block

In many matrix computations that involve large data sets, accumulation with high accuracy needs to be supported. The accumulation can be done either off-chip in the host computer or on-chip using a customized MAC block. Figure 10 shows the block diagram of an on-chip MAC block. The bit-widths of the mantissa and exponent can be varied according to the application requirement. The default precision of the MAC block is set to be single precision floating point. The intermediate result is stored in a local buffer utilizing a FIFO due to the latency associated with floating point multipliers and adders. The state machine for scheduling timing for the MAC block takes into account the latency of the variable precision multipliers and adders. This MAC block is duplicated for each output from the kernel function unit to achieve full throughput.

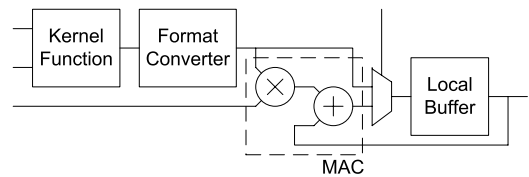


Fig. 10. Block Diagram of MAC Block

4.4 Data Flow Control Generator

The Data Flow Control Generator shown in Figure 11 contains instantiations of memory blocks and state machines that control the data flow for the N-body problem. There are several types of memory components in an FPGA including registers, FIFOs, RAMs, and ROMs. The specific components are chosen based on an analysis of access patterns and size of each input type. In general, constants are stored in registers, and input data is stored in one FIFO and two single-port RAMs. In addition, FIFOs are used to store results; the number of these FIFOs depends on the number of kernel function pipelines used.

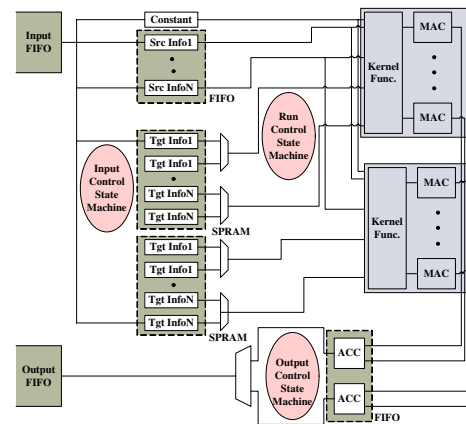


Fig. 11. Block Diagram of Data Flow Control Block for the N-body problem.

There are three state machines responsible for controlling data flow to provide high throughput performance. The first state machine handles storing of input data. The second state machine controls hardware execution and generates signals for reading data from input FIFO and RAM, and writing the data to the output FIFOs. The third state machine decides the access order of output FIFO to avoid collisions. The dataflow sequence is summarized below.

- 1) The target information for one tile is stored.
- 2) The source information is fed in the sequence specified by the tiling method. The hardware execution starts as soon as the first source tile is available.
- 3) Once the entire source information for the current tile is read and the hardware execution has started, the next target tile is fetched.
- 4) Steps (2) and (3) are repeated until all target information is processed and the result is calculated.

In this architecture, spatial and temporal parallelism is exploited for improving performance. Specifically, spatial par-

allelism is exploited by using multiple kernel function blocks and sharing the source information that is common to all the target tiles. Temporal parallelism is exploited by overlapping the kernel function evaluation and accumulation phases.

The automated HDL generation for this block uses HDL templates that contain various *parameter* and *generate* sentences supported by Verilog-2001 standard. The parameter value setting and instantiation mapping is automatically done by our tool from information generated by CPA and kernel function code generator.

4.5 Host-FPGA interface block

The Host-FPGA interface block is composed of PCI interface controller that generates the control signals for the PCI Express interface core¹ and input/output buffer as shown in Figure 12. The PCI interface controller block supports DMA operations to maximize the communication speed using burst mode transactions. Burst mode transactions enable minimum latency communication and therefore helps to achieve the theoretical maximum throughput of the PCI Express interface.

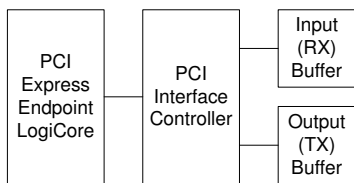


Fig. 12. Block Diagram of Host-FPGA interface block

5 TANOR SOFTWARE MODULES

The key task of the software module is to generate the host computer program (in C++) that facilitates the data communication between the FPGA accelerator and the host computer. The structure of the software system is composed of four hierarchical stages: Algorithm Optimizer, Packetizer/Depacketizer, Direct Memory Access Controller and Device Drive for PCI. In the Algorithm Optimizer stage, matrix tiling and data traversal is implemented in MATLAB. The Packetizer generates the packets that combine input data and control commands necessary to utilize the hardware system. The Direct Memory Access (DMA) controller is the primary method of sending/receiving data from/to the FPGA accelerator. Since the initiation of a burst mode transaction from the host is not supported by the operating system, the transfer information is written to the FPGA accelerator. The accelerator then initiates send/receive operations utilizing burst mode transactions. Jungo WinDriverTM is used as a device driver to initialize and set up PCI device registers.

The simplified procedure to operate the FPGA accelerator from the host computer is shown in Figure 13. The communication of data to and from the FPGA board is synchronized using interrupts. The input data is structured by the host

interface by inserting commands that aid the hardware in identifying the tile boundaries. The structured input sequence is then written into an input buffer, which is subsequently read by the PCI Interface in bursts. The Host-FPGA Interface block generates an interrupt after processing a target tile, and the partial results are written back to the output buffer. The results are in turn processed and re-ordered by the host to fit the output format.

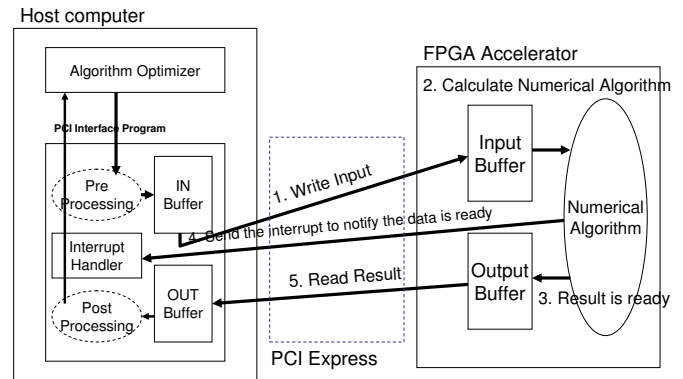


Fig. 13. Interaction between the host computer and the FPGA accelerator

6 CASE STUDY

6.1 System Configuration and Experimental Setup

We first describe the experimental setup. The synthesis environment consists of Synplify Pro 8.6.2 and Xilinx ISE 8.2i. The target platform is the Xilinx Virtex2Pro-100 device. More specifically, we have used DN6000k10PCIe-4 [41] logic emulation system as the target hardware platform. It consists of six Xilinx Virtex2Pro-100 devices out of which we have used two. The PCI Express block and data interface FIFO block are configured to provide PCI Express 4 Lane DMA mode of operation on one of the FPGAs, and the other FPGA is used to implement the blocks that are specific to the numerical algorithm. The testbed system is shown in Figure 14.

The user provides a MATLAB description of the kernel function, data sets, optimization objectives and constraints through a GUI interface on the host computer as shown in Figure 2. TANOR takes about 3 seconds to translate the MATLAB specifications to HDL code on a Pentium-4 2.4GHz processor. The hardware synthesis part takes time proportional to the size of the design that needs to be mapped. For instance,

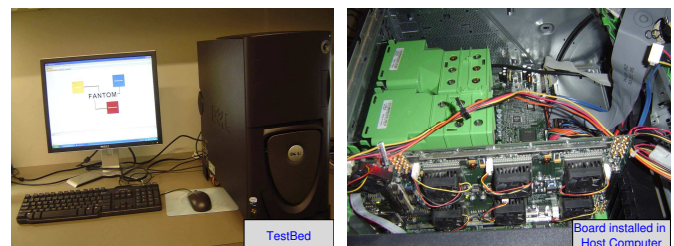


Fig. 14. TANOR System

1. The PCI Express Endpoint LogiCORE from Xilinx is used for PCI Express Core. The 4 lane configuration of the PCI Express Core can send/receive 64bits of data at a frequency of 125MHz [40].

the synthesis time ranged from 1 to 7 hours for the following experiments. In the end, the result window shows the result summary including area, timing and accuracy.

In the rest of the section, we demonstrate the capability of our automated tool for three different kernel functions: the Gaussian kernel, the Gravitational kernel, and a force calculation kernel applied in molecular dynamics.

6.2 Interaction Kernel Function Examples

6.2.1 The Gaussian Kernel

The Gaussian kernel is frequently used in image processing and reconstruction [42]. A two-dimensional image is used for the test data set. The Gaussian kernel is represented by

$$\mathbf{h}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}, \quad (3)$$

where x_i, x_j are spatial positions of the i -th and j -th pixels, respectively. A truncated Taylor series expansion is used for hardware implementation of the function evaluation. Specifically, the Gaussian kernel is approximated by Taylor polynomial of degree 2.

6.2.2 The Gravitational Kernel

The Gravitational kernel is used in astrophysical N-body simulations [18]. In every time step of the simulation, the gravitational force, its time derivative, and the gravitational field potential exerted on a target set of particles, \mathbf{T} , due to mass at a source set of particles, \mathbf{S} , is computed. There are seven kernel functions for calculating the magnitude of these parameters at every target particle. The input test set consists of the source and target particle locations in a three-dimensional space R^3 .

$$\mathbf{a}_i = c \cdot \sum_{s_j \in \mathbf{S}} \frac{m(s_j) \cdot (\mathbf{t}_i - \mathbf{s}_j)}{\|\mathbf{t}_i - \mathbf{s}_j\|^3}, \quad (4)$$

$$\dot{\mathbf{a}}_i = c \cdot \sum_{s_j \in \mathbf{S}} m(s_j) \cdot \left[\frac{\mathbf{v}_{ij}}{\|\mathbf{t}_i - \mathbf{s}_j\|^3} - \frac{3(\mathbf{v}_{ij} \cdot (\mathbf{t}_i - \mathbf{s}_j))(\mathbf{t}_i - \mathbf{s}_j)}{\|\mathbf{t}_i - \mathbf{s}_j\|^5} \right], \quad (5)$$

$$\theta_i = c \cdot \sum_{s_j \in \mathbf{S}} \frac{m(s_j)}{\|\mathbf{t}_i - \mathbf{s}_j\|}, \quad \mathbf{t}_i \in \mathbf{T}. \quad (6)$$

Here t_i and s_j denote the spatial position of the target and source particles in \mathbf{T} and \mathbf{S} , respectively $\|t - s\|$ denotes the Euclidean distance between t and s , $m(s_j)$ is the mass of the particle at location s_j , and c is a constant. Equations 4 and 6 compute the acceleration and potential of a target particle t_i , and equation 5 computes the time derivative of the particle acceleration.

6.2.3 Force Calculation in Molecular Dynamics

Force calculation in molecular dynamics simulation is computationally very intensive and is implemented as follows [30].

$$\vec{f}_i = \sum_{j \neq i} \left(\frac{12A}{r_{ij}^{14}} - \frac{6B}{r_{ij}^8} + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}^3} \right) \vec{r}_{ij}, \quad (7)$$

where \vec{r}_{ij} is the distance vector between atoms i and j in 3-dimensional space, and r_{ij} is the magnitude of \vec{r}_{ij} . A and B are constants and q_i and q_j is the charge on atoms i and j , respectively.

6.3 Evaluation metrics

The following metrics have been used to evaluate the different configurations in our experiments.

- Area is reported by the number of occupied slices, and the maximum number of pipelines (or parallel versions) that can fit onto a single FPGA.
- Power consumption is measured using XPower [43] for a 125 MHz clock.
- Performance is obtained through actual time measurements on our target platform for a problem size of 5K particles averaged over 20 different executions.
- Accuracy is computed in MATLAB. It is represented by the Euclidean distance of the deviation vector between the FP (or FX) implementations and its double precision counterpart, divided by the Euclidean distance of the double precision counterpart:

$$\frac{\sqrt{\Delta x_1^2 + \dots + \Delta x_n^2}}{\sqrt{x_1^2 + \dots + x_n^2}} \quad (8)$$

6.4 Trade-off analysis

Table 6 shows the performance of the three kernels with respect to power, performance, latency and accuracy. We chose two different configurations of fixed point and floating point implementations to demonstrate the effect of precision on the performance metrics. The notation FX-[x y] stands for fixed point implementation with x and y representing the maximum bit-width and maximum fractional bit-width, respectively. The notation FL- e m , stands for floating point implementation with e and m representing the exponent and mantissa bit widths, respectively.

We first compare the kernels with respect to the number of operations, defined as the number of primitive operations used to implement each kernel function, and the number of pipelines per FPGA, defined as the maximum number of data-path pipelines that can be included in one FPGA. The gravitational kernel has the highest number of operations per data-path pipeline since there are 7 kernel functions that have to be implemented. Consequently, it has the smallest number of pipelines per FPGA. In the Gaussian kernel, FL-e8m23 has the largest resource requirement due to its large precision and consequently, the smallest number of pipelines. In contrast, FX-[64 15] has the smallest resource requirement and supports the largest number of pipelines. Note that the number of pipelines is proportional to the performance, so the configuration with the highest number of pipelines also shows the best performance. Thus configuration FX-[64 15] for the Gaussian kernel supports maximum number of pipelines and has the highest performance. It also has the lowest accuracy. In general, there is an inverse relationship between accuracy and performance. TANOR is flexible enough to use FL configurations for high accuracy and FX for high performance.

The latency refers to the number of clock cycles required to compute each kernel. The performance metric is latency

TABLE 6
Resource utilization, power and performance evaluation

(a) Gaussian Kernel

Configuration	# of Operations	# of Pipelines per FPGA	Latency (Clk cycle)	Power (W)	Performance (Gflops)	Accuracy
FX-[64 15]	19	11	59	5.5	26.1	2.70e-03
FX-[64 21]	19	8	69	5.3	19.2	3.13e-05
FL-e8m16	19	8	95	5.2	19.2	6.20e-07
FL-e8m23	19	6	109	5.6	13.9	2.42e-07

(b) Gravitational Kernel

Configuration	# of Operations	# of Pipelines per FPGA	Latency (Clk cycle)	Power (W)	Performance (Gflops)	Accuracy
FX-[64 12]	56	3	83	6.5	22.2	1.59e-02
FX-[64 15]	56	3	92	7.1	22.2	2.51e-04
FL-e8m16	56	3	137	6.5	22.2	1.34e-04
FL-e8m23	56	2	161	6.1	13.8	1.35e-06

(c) Molecular Dynamics Kernel

Configuration	# of Operations	# of Pipelines per FPGA	Latency (Clk cycle)	Power (W)	Performance (Gflops)	Accuracy
FX-[64 13]	25	6	78	4.9	18.4	3.65e-02
FX-[64 16]	25	5	88	5.6	15.2	3.03e-03
FL-e8m16	25	5	120	4.7	15.2	1.01e-03
FL-e8m23	25	4	131	5.2	12.1	8.80e-06

TABLE 7

Comparison with GRAPE-6: Gravitational kernel

	GRAPE-6 CHIP	TANOR CHIP
Device	ASIC	FPGA (XC2VP100-6)
Device tech.	0.25 μ m	0.13 μ m
Pipelines/chip	6	3
Frequency(MHz)	90	125
Real Peak Flops	17.2G	22.2G
Power Consumption	~12W	~6.5W

insensitive because the implementation uses a fully pipelined architecture for the highest performance. The power numbers correspond to maximal utilization of resources in the FPGA. Since maximum possible number of pipelines are mapped to each FPGA, there is very little difference in the power numbers between different configurations.

6.5 Comparison with GRAPE-6

Next we compare the hardware generated automatically by TANOR with the GRAPE-6 architecture that was custom designed for numerical simulation of gravitational interactions. The results are shown in Table 7. We see that the hardware generated by TANOR achieves a performance comparable to the GRAPE-6 chip in terms of FLOPS and power consumption. A direct comparison is difficult because of the differences in underlying technology (0.25 μ m vs 0.13 μ m) and implementation style (ASIC vs FPGA) of the two systems. However, these results serve as an indicator of the quality of the output generated by TANOR.

6.6 Algorithm-Architecture Codesign

In this section we illustrate the use of TANOR for joint exploration of algorithmic and architectural options. The results presented in the next two subsections are from actual synthesis, P&R steps.

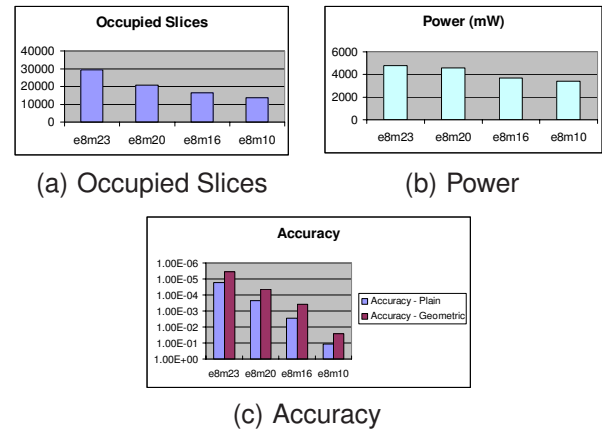


Fig. 15. Example of architecture-algorithm co-exploration for the Gravitational kernel

6.6.1 Example 1

The area, power and accuracy tradeoffs for different configurations for the Gravitational kernel are shown in Figure 15. The results are for a single pipelined structure. We see that the accuracy, area and power consumed increase with increase in the number of mantissa bits. We also show the effect on accuracy for two types of data traversal schemes, namely, PT and GT, in Figure 15(c). The use of GT with mantissa bit width of 16 can achieve accuracy comparable to that of PT with mantissa bit width of 20. Note that this decrease in the bit width translates into a 20% reduction in utilized area on the FPGA, and a 19% reduction in power consumption as shown in Figure 15(a) and 15(b).

6.6.2 Example 2

Next we show how TANOR facilitates design space exploration with the computation of the Bessel function $J_0(x)$ using

LUT based interpolation schemes. The inputs to $J_0(x)$ are the distances between particles in the target set \mathbf{T} and the source set \mathbf{S} . Each set contains 5K particles randomly generated in the range $(0, 100)$. The accuracy requirement is of the order of 10^{-5} .

TABLE 8

LUT implementations of $J_0(x)$ with different degrees and geometric tiling

Function $J_0(x)$		Plain				Geom.
Configuration		PT ₀	PT ₁	PT ₂	PT ₈	GT ₂
Taylor degree n		0	1	2	8	2
# operation	Add +	3	6	8	27	8
	Mult *	1	3	6	31	6
Lookup table	# entries	1.6e+6	12802	2403	864	282
	size	4.3MB	34.4kB	6.45kB	2.32kB	0.76kB
Occupied slices		N.A.†	382	716	4416	766
18×18bit multipliers		N.A.†	12	24	132	24
Power(mW) @125MHz		N.A.†	804	866	1664	894
Accuracy		1.23e-5	1.08e-5	1.35e-5	1.49e-5	3.83e-6

† Indicates that this configuration cannot fit on the target FPGA

Table 8 shows the number of additions and multiplications used for address calculation and interpolation. It also lists the memory size, which is expressed in terms of number of table entries to store the pre-computed derivatives used in Taylor series expansion. Each individual configuration is labeled PT _{n} (for plain tiling) and GT _{n} (for geometric tiling), where n stands for degree of the Taylor polynomial.

The simplest way to satisfy the accuracy requirement is to apply a direct lookup scheme, PT₀, which uses a large lookup table because of the high sampling density. Such a table cannot fit into a single XC2VP100 device and so we make use of PT₁ which uses a first degree Taylor polynomial to significantly reduce the table size, at the cost of additional multipliers and adders. PT₂ and PT₈ show that the table size can be further reduced if we continue to increase the degree of the Taylor polynomial.

Next, assume that the target architecture imposes the constraints of 3KB for table size and 1K for number of occupied slices. From Table 8, we see that PT₁ achieves the accuracy and slice constraints but violates the memory constraint. The Taylor polynomial degree is increased and the memory constraint is satisfied for degree 8 corresponding to PT₈. However this requires a large number of slices that far exceeds the area constraint. Since it is impossible to find a balance between area, accuracy and memory only by changing the degree of the Taylor polynomial in conventional PT schemes, a GT based scheme GT₂ is applied. It uses a lower degree of the Taylor polynomial (degree 2 in this example) and thus requires fewer slices and multipliers, fewer number of lookup table entries, and much smaller memory.

6.6.3 Model Validation

The results presented in Section 6.6.1 and 6.6.2 were generated from actual synthesis, P&R steps. For larger designs, this may easily take hours. In comparison, the area and power estimates using the models described in Section 4.2 take a few seconds or less. In this section, we compare the estimation results with those obtained by actual synthesis followed by P&R for a representative set of examples.

TABLE 9

Resource estimation results for Gaussian and $J_0(x)$ functions

Config.	Slices			Total Power (mW)		
	Estim.	Synth.	Err.	Estim.	Synth.	Err.
Gauss(3,64,15)	459	465	1.29%	432	424	1.83%
Gauss(3,64,20)	724	722	0.28%	498	481	3.60%
Gauss(9,64,15)	1011	1121	9.81%	574	612	6.24%
Gauss(9,64,20)	2125	2079	2.21%	883	867	1.33%
J ₀ (3,64,15)	779	759	2.64%	508	482	5.50%
J ₀ (3,64,20)	1258	1113	12.99%	629	583	7.99%
J ₀ (9,64,15)	2057	2437	14.97%	845	945	10.09%
J ₀ (9,64,20)	4779	4315	10.57%	1589	1437	10.62%

Table 9 shows the results for Gaussian kernel (Gauss) and Bessel function ($J_0(x)$) [38]. Both functions are implemented using LUT-based interpolation schemes in fixed point; the notation (n, X, Y) means n is the degree of Taylor polynomial, X is the maximum number of total bits and Y is the maximum number of fraction bits. For the 8 configurations, the average error is 6.84% for the number of slices and 5.90% for the total power. The results of block RAM and 18x18-bit Multipliers are not shown because there is no mismatch between the estimated and synthesized results. The accuracy of our model is quite high even for large designs. For instance, for 8-point FFT implemented in floating point format, the average error is 1.86% for the number of slices and 3.49% for the total power [38].

7 CONCLUSION

We have presented TANOR, a framework for automatic generation of efficient system architectures for accelerating numerical computations on a reconfigurable platform. TANOR generates the desired hardware modules and a software data communication interface, starting from a high-level MATLAB description. It incorporates a high level synthesis flow and supports custom fixed point and floating point configurations. It is also capable of supporting many transcendental functions used in scientific simulation and signal and image processing through LUT generation. Finally, TANOR enables the joint exploration of algorithmic and architectural options.

TANOR can be applied to a wide spectrum of DSP applications though here we have demonstrated TANOR's capabilities with three different N-body applications. TANOR generated accelerators are shown to be competitive with other existing custom designed hardware accelerators such as GRAPE-6. Further, by co-exploration of variable precision architectures and data traversal schemes, TANOR automation flow is able to achieve a 20% reduction in resource utilization, and a 19% reduction in power consumption while maintaining comparable accuracy.

ACKNOWLEDGMENTS

This work is supported in part by grants from DARPA W911NF-05-1-0248 and NSF CAREER 0093085.

REFERENCES

- [1] Y. Atat and N.-E. Zergainoh, "Simulink-based MPSoC design: new approach to bridge the gap between algorithm and architecture design," 2007 IEEE Computer Society Annual Symposium on VLSI, pp. 9–14, 2007.

- [2] D. Soderman and Y. Panchul, "Implementing C designs in hardware: a full-featured ANSI C to RTL Verilog compiler in action," *Proceedings of International Verilog HDL Conference and VHDL International Users Forum*, pp. 22–29, 1998.
- [3] P. Banerjee, M. Haldar, and et al., "Overview of a compiler for synthesizing MATLAB programs onto fpgas," *IEEE Transactions on VLSI systems*, vol. 12, pp. 312–323, March, 2004.
- [4] MATLAB, "The MATLAB website," <http://www.mathworks.com>, 2007.
- [5] J. S. Kim and et al., "TANOR: A tool for accelerating N-body simulations on reconfigurable platform," *International Conference on Field Programmable Logic and Applications - FPL'2007*, pp. 68–73, August 2007.
- [6] N.-E. Zergainoh, K. Popovici, A. Jerraya, and P. Urard, "IP-block-based design environment for high-throughput VLSI dedicated digital signal processing systems," *Proceedings of Asia and South Pacific Design Automation Conference - ASP-DAC'05.*, pp. 612–618, 2005.
- [7] N. Zergainoh, L. Tambour, P. Urard, and A. Jerraya, "Macrocell builder: IP-block-based design environment for high-throughput VLSI dedicated digital signal processing systems," *EURASIP Journal on Applied Signal Processing*, pp. 1–11, 2006.
- [8] SysGen, "The Xilinx web page," http://www.xilinx.com/ise/optional_prod/system_generator.htm, 2007.
- [9] Impulsec, "The impulsec web page," <http://www.impulsec.com>, 2007.
- [10] Catapult, "The Mentor web page," http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/, 2007.
- [11] Cameron, "Cameron: Compiling high-level programs to FPGA configurations," <http://www.cs.colostate.edu/cameron/>, 2002.
- [12] Calypto, "Calypto's sequential analysis technology," <http://www.calypto.com/>, 2008.
- [13] S. Vasudevan, J. Abraham, V. Viswanath, and J. Tu, "Automatic decomposition for sequential equivalence checking of system level and RTL descriptions," *Proceedings of International Conference on Formal Methods and Models for Co-Design - MEMOCODE '2006*, pp. 71–80, July 2006.
- [14] AccelDSP, "Xilinx AccelDSP synthesis tool," http://www.xilinx.com/ise/dsp_design_prod/acceldsp, 2007.
- [15] MATCH, "MATCH: A MATLAB compilation environment for distributed heterogeneous adaptive computing systems," <http://www.ece.northwestern.edu/cpdc/Match>, 2002.
- [16] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim, and R. Uribe, "Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design," *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM '2003*, p. 263, 2003.
- [17] J. Reif and S. Tate, "The complexity of N-body simulation," *Proceedings of the 20th International Colloquium on Automata, Languages and Programming - ICALP '93*, pp. 162–176, 1993.
- [18] J. Makino, "The GRAPE project," *Computing in Science and Engineering*, vol. 8, no. 1, pp. 30 – 40, 2006.
- [19] T. Fukushige, M. Taiji, J. Makino, T. Ebisuzaki, and D. Sugimoto, "A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: MD-GRAPe," *Astrophysical Journal*, vol. 468, no. 1, pp. 51 – 61, 1 Sept. 1996.
- [20] R. Susukita, T. Ebisuzaki, B. Elmegreen, H. Furusawa, K. Kato, A. Kawai, Y. Kobayashi, T. Koishi, G. McNiven, T. Narumi, and K. Yasuoka, "Hardware accelerator for molecular dynamics: MDGRAPE-2," *Computer Physics Communications*, vol. 155, no. 2, pp. 115 – 131, 2003.
- [21] S. Toyoda, H. Miyagawa, K. Kitamura, T. Amisaki, E. Hashimoto, H. Ikeda, A. Kusumi, and N. Miyakawa, "Development of MD engine: High-speed accelerator with parallel processor design for molecular dynamics simulations," *Journal of Computational Chemistry*, vol. 20, no. 2, pp. 185 – 199, 1999.
- [22] T. Amisaki, S. Toyoda, H. Miyagawa, and K. Kitamura, "Development of hardware accelerator for molecular dynamics simulations: A computation board that calculates nonbonded interactions in cooperation with fast multipole method," *Journal of Computational Chemistry*, vol. 24, no. 5, pp. 582 – 592, 2003.
- [23] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 197–206, 2004.
- [24] Y. Gu, T. VanCourt, and M. C. Herboldt, "Accelerating molecular dynamics simulations with configurable circuits," *Proceedings of IEE Computers and Digital Techniques*, vol. 153, no. 3, pp. 189–195, 2006.
- [25] T. A. Cook, H.-R. Kim, and L. Louca, "Hardware acceleration of N-body simulations for galactic dynamics," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 2607, pp. 115 – 126, 1995.
- [26] W. Smith and A. Schnore, "Towards an RCC-based accelerator for computational fluid dynamics applications," *Journal of Supercomputing*, vol. 30, no. 3, pp. 239 – 261, 2004.
- [27] T. Hamada, T. Fukushige, A. Kawai, and J. Makino, "PROGRAPE-1: a programmable special-purpose computer for many-body simulations," *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 256–257, 1998.
- [28] T. Hamada and N. Nakasato, "PGR: a software package for reconfigurable super-computing," *International Conference on Field Programmable Logic and Applications*, pp. 366–373, 2005.
- [29] D. P. V. Kindratenko, "A case study in porting a production scientific supercomputing application to a reconfigurable computer," in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.
- [30] R. Scrofano, M. Gokhale, F. Trouw, and V. K. Prasanna, "A hardware/software approach to molecular dynamics on reconfigurable computers," *Proceedings of 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 23–34, 2006.
- [31] K. Sobti, L. Deng, C. Chakrabarti, N. Pitsianis, X. Sun, J. Kim, P. Mangalagiri, K. Irick, M. Kandemir, and V. Narayanan, "Efficient function evaluations with lookup tables for structured matrix operations," *IEEE Workshop on Signal Processing Systems - SiPS '2007*, pp. 463–468, Oct. 2007.
- [32] G. Chen, L. Xue, J. Kim, K. Sobti, L. Deng, X. Sun, N. Pitsianis, C. Chakrabarti, M. Kandemir, and N. Vijaykrishnan, "Using geometric tiling for reducing power consumption in structured matrix operations," in *IEEE International SOC Conference*, 2006, pp. 113 – 114.
- [33] R. A. Walker and R. Camposano, *A Survey of High-Level Synthesis Systems*, 1st ed. Boston, MA: Kluwer Academic Publishers, 1991.
- [34] L. H. de Figueiredo and J. Stolfi, *Self-Validated Numerical Methods and Applications*, ser. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, Rio de Janeiro, Brazil, 1997.
- [35] D. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy guaranteed bit-width optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1990–2000, 2006.
- [36] K. Sobti, "FANTOM: A fixed point framework for algorithm architecture co-design," M. S. Thesis, Arizona State University, Tempe, Arizona, August 2007.
- [37] LogicoreAddSub, "The Xilinx webpage," www.xilinx.com/ipcenter/catalog/logicore/docs/addsub.pdf, 2007.
- [38] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of FPGA implementations," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing - ICASSP '2008*, pp. 1417–1420, April 2008.
- [39] P. H. Sherrod, "Nonlinear Regression Analysis Program," <http://www.nlreg.com/NLREG.pdf>, 2002.
- [40] PCIe, "Xilinx PCI Express Endpoint LogiCORE," http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-PCIEXP, 2007.
- [41] Dinigroup, "Dinigroup dn6000k10pcie-4," <http://www.dinigroup.com/index.php?product=DN6000k10pcie>, 2007.
- [42] R. C. Gonzalez and R. E. Woods, *Digital image processing (2nd ed.)*. Prentice Hall, 2002.
- [43] Xpower, "The Xilinx webpage," http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm, 2007.



JungSub Kim (S/07) received his BS and MS degrees in Electrical Engineering from Yonsei University, Seoul, Korea in 1995 and 1997 respectively, and PhD Degree in Electrical Engineering from the Pennsylvania State University in 2008. From 1997 to 2003, he worked in the Embedded System Laboratory, R&D Center, LG Industrial system, Anyang, Korea. Currently he is a Senior Engineer with Architecture Research Lab, DMC R&D Center, Samsung Electronics, Suwon, Korea. His research interests include

high-performance reconfigurable systems design and reliable circuit design.



Mahmut Kandemir is an associate professor in the Computer Science and Engineering Department at the Pennsylvania State University. His research interests are in optimizing compilers, runtime systems, embedded systems, I/O and high performance storage, and power-aware computing. He is a recipient of NSF Career Award and the Penn State Engineering Society Outstanding Research Award.



Lanping Deng received his BS and MS degrees in Electrical Engineering from Tsinghua University, Beijing, China in 2003 and 2005, respectively. He is a PhD candidate in Electrical Engineering Department, Arizona State University, Tempe, AZ. His research interests include hardware-software codesign, FPGA-based accelerator design and EDA tool design.



Vijaykrishnan Narayanan is a Professor of Computer Science and Engineering at The Pennsylvania State University, University Park. His research interest are in Computer Architecture, Embedded Systems and Nanoarchitectures.



Prasanth Mangalagiri received the dual(BS and MS) degree in computer science from the Indian Institute of Technology, Madras. He is working toward the doctorate degree in the Department of Computer Science and Engineering, Pennsylvania State University, University Park. His current research interests include reliable system design, hardware-software codesign, and high-performance computing on FPGAs.



Chaitali Chakrabarti (SM/02) is a Professor with the Department of Electrical Engineering, Arizona State University, Tempe. Her research interests are in the areas of low power embedded systems design and VLSI architectures and algorithms for signal processing, image processing, and communications.



Kevin Irick received the BS degree in Electronics Engineering Technology from DeVry University, Atlanta, Georgia. He received the MS degree in Computer Science and Engineering and is currently working toward the doctorate degree in the Department of Computer Science and Engineering at The Pennsylvania State University, University Park, Pennsylvania. His research interests include application specific hardware accelerators, hardware assisted image processing and recognition, and high-performance computing on FPGAs.

puting on FPGAs.



Nikos Pitsianis is an Assistant Professor with the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Greece and an adjunct professor with the Departments of Computer Science and Electrical and Computer Engineering of Duke University, Durham, NC. His research interests include high performance algorithms and architectures for signal and image processing.



Kanwaldeep Sobti received his MS degree in Electrical Engineering from Arizona State University, Tempe in 2007. He is a Senior Design Engineer with Advanced Micro Devices (AMD) in the field of structural testability and DFT techniques for AMD's next generation processor. His interests lie in microprocessor design, low power design, and CAD design, especially for ESL techniques.



Xiaobai Sun is a Professor of Computer Science at Duke University. Her research interests and efforts focus on numerical algorithm design and analysis, especially, in bridging and blending mathematical models and computer architectures for scientific simulation and signal processing.