# TANOR: A TOOL FOR ACCELERATING N-BODY SIMULATIONS ON RECONFIGURABLE PLATFORM

*J. S. Kim, P. Mangalagiri, K. Irick, M. Kandemir, V. Narayanan*
Department of Computer Science and Engineering
Pennsylvania State University
email: jskim, mangalag, irick, kandemir, vijay@cse.psu.edu

*K. Sobti, L. Deng, C. Chakrabarti*
Department of Electrical Engineering
Arizona State University
email: Kanwaldeep.Sobti, Lanping.Deng, chaitali@asu.edu

*N. Pitsianis, X. Sun*
Department of Computer Science
Duke University
email: nikos, xiaobai@cs.duke.edu

## ABSTRACT

Algorithm-architecture co-exploration is hindered by the lack of efficient tools. As a consequence, designers are currently able to explore only a limited set of points in the whole design space. Therefore, a tool that can allow fast exploration of algorithmic and architectural tradeoffs in an automated manner is highly desired. In this paper, we describe TANOR an automated tool targeted for designing hardware accelerators for the class of N-body interaction problems. The design flow, starting from a high level (MATLAB) description, configures the entire system automatically. We describe the design of TANOR and demonstrate the effectiveness and adaptability of our tool using three different target applications, namely, the gravitational kernel used in astrophysics, the gaussian kernel common in image processing applications, and a force calculation kernel applied in molecular dynamics. Our results demonstrate that TANOR generates hardware accelerator that are competitive with existing custom accelerator.

## 1. INTRODUCTION

Traditionally, ASICs have been used to achieve a performance speedup in scientific applications. However, in most of these scientific applications, the precise algorithmic specifications are often subject to change making reconfigurability desirable. Recent advances in reconfigurable hardware, especially field programmable gate arrays (FPGAs), have facilitated the use of reconfigurable hardware for such computationally intensive scientific applications. The computationally intensive tasks of such applications are accelerated through the reconfigurable hardware. Most of the scientific application designers work at algorithmic level and use high-level languages to model their applications. To realize such an application in reconfigurable hardware, the high-level description has to be translated into a hardware description language (HDL). Manually translating the high-level descriptions into a HDL description that fits into the traditional RTL design flow is a clearly very tedious and error-prone process. One of the main goals of our automated tool flow is to aid the designer in seamlessly transforming the high-level description of an application in Matlab to HDL.

Several design tools that automate the conversion between high-level languages and RTL have been developed to reduce the design cycle and provide scope for algorithmic optimization and architectural exploration. The input format of the design to these tools can be broadly classified into three different categories, high-level graphical languages, algorithmic languages, and behavioral hardware descriptions[1]; Algorithmic languages like SRC Computer C/Fortran[2], Impulse C[3], Catapult C (Mentor Graphics)[4], and Xilinx AccelDSP Matlab subset[5] have been developed and tailored into the respective tool-flows developed by several FPGA vendors. While languages such as SRC and Impulse high-level specifications reduce the complexity of code development for reconfigurable platforms, the process of porting an existing scientific code to one of these languages is quite tedious - something scientific software developers are not acquainted with.

In this work, we focus on building our tool to support the class of N-body interaction problems that supports both algorithmic and architectural exploration. The classical n-body problem simulates the evolution of a system of N bodies, which interact. It has diverse applicability for studying particle dynamics in areas as varied as Astrophysics, Molecular Modeling, and Quantum Chemistry[6]. These tasks are very computationally intensive. The wide range of application domains and the computationally intensive sub-tasks involved in n-body simulations require reconfigurable hardware accelerators. One of the prominent works in this area has been the GRAPE project which implements the gravitational N-body interaction problem[7]. The GRAPE system used ASICs to accelerate astrophysical n-body simulations, and was able to attain a peak performance of 63.4 Tflops from GRAPE-6 system in 2003. However, as recently noted in [7], there is a need for an automated flow to support architectures that can be configured for various N-body problems. The GRAPE PGR system[8] is an effort in this direction to support reconfigurable systems. Also, some early efforts have been made towards modeling molecular dynamics using SRC Computers in [9, 10]. However, this system requires use of a specialized language and requires manual intervention in many key steps of translating specification to implementation such as extracting timing information.

TANOR(A Tool for Accelerating N-body Simulations on Reconfigurable Platform) was motivated by the need for an automated system generator for N-body interaction problems and driven by the following key targets,

- Support for kernel specification of the N-body problem in a high-level language (MATLAB) making the algorithm-architecture exploration accessible to application designer.

- Provide the end-user control over the design optimizations and a fully automated design flow starting from MATLAB specification to HDL synthesis.
- Realize designs competitive in terms of power, area, accuracy, and performance to the existing N-body hardware accelerators.

In section 2, we present an overview of TANOR. In section 3, the details of key modules of our tool are explained. In section 4, we provide a case study to demonstrate the use of TANOR. We compare the achieved performance of our implementation with previous efforts in this direction. Finally, in section 5, we provide conclusions.

## 2. TANOR DESIGN FLOW

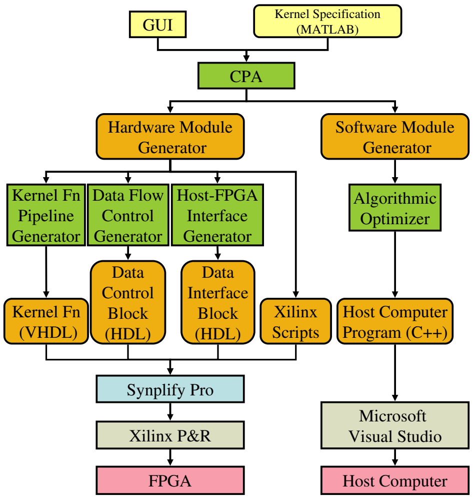

GUI
Kernel Specification (MATLAB)
CPA
Hardware Module Generator
Software Module Generator
Kernel Fn Pipeline Generator
Data Flow Control Generator
Host-FPGA Interface Generator
Algorithmic Optimizer
Kernel Fn (VHDL)
Data Control Block (HDL)
Data Interface Block (HDL)
Xilinx Scripts
Host Computer Program (C++)
Synplify Pro
Xilinx P&R
FPGA
Microsoft Visual Studio
Host Computer

**Fig. 1**. TANOR Design Flow

Figure 1 shows a block diagram of our automated design flow. The input to TANOR consists of a high-level description of a particular interaction kernel function for the N-body problem, provided in MATLAB, and constraints on accuracy, latency, area, and power, provided by a graphical user interface.

By using **MATLAB** as the specification language, we do not require a separate software emulator for functional verification of the design at early stages of the design flow. Once the kernel specification along with the constraint information is provided by the user, they are processed by the code parser and analyzer**(CPA)**. This phase parses the kernel functions into an intermediate format which is used to perform a series of optimizations. These optimizations remove any redundancies, as a redundancy in the code can subsequently translate to extraneous processing units in the hardware.

After the CPA phase, the flow is partitioned into the hardware and software modules. The task of the hardware module is to generate the bit file for configuring the FPGA system. This module generates the HDL description of different blocks of the architecture along with the scripts required by the back-end FPGA compiler. Our system is currently targeted to support Synplify Pro 8.6.2 and Xilinx ISE 8.1.03i as back-end tools to generate the bit file to be configured on the FPGA.

The generated architecture is composed of three main blocks (See Figure 2): Kernel Function block, Data flow control block, and Host-FPGA interface block. The Kernel Function block is implemented using the kernel function pipeline generator, which generates the pipeline architecture of the kernel function in HDL based on the information from CPA phase. Based on the resource limitations of the target FPGA and the resource size of the generated kernel function, the number of parallel pipelines to be included in one FPGA is inferred. The Data flow control block includes the state machines which control the data steering between the host and the generated hardware. Moreover, it includes storage buffers and queues. The automated HDL generation for this block uses HDL templates that contain various *parameter* and *generate* sentences supported by Verilog-2001 standard. This parameter value setting and instantiation mapping is automatically done by our tool after gathering information from CPA and kernel function code generator. The Host-FPGA interface block, generates the control signals for PCI Express core[1] and serves as an interface between RX/TX FIFO of the PCI Express core and the user application.

The generated hardware configuration exploits both spatial and temporal parallelism. Spatial parallelism is supported by multiple instantiations of the pipelines of kernel function. The temporal parallelism is exploited by taking into account the latency of various arithmetic units. For instance, if we choose the single-precision floating point adder for the accumulation in a MAC(Multiplier Accumulator) block, each instance of the accumulated partial sum has a latency of 14 clock cycles. Since the architecture is pipelined, the instructions for accumulating different target data are scheduled simultaneously to maximize the throughput of the accumulator.

The software module generates the host computer program (in C++) that interfaces the FPGA with the host computer. A key task of the software module is to communicate data between the host computer and the FPGA accelerator. The communication of data to and from the FPGA board is synchronized using interrupts. The input data is processed and the data flow partitioning is done by inserting commands that aid the hardware in distinguishing between source and target sets in the N-body interaction problem. The input sequence is then read into an input buffer which is read by the PCI Interface in bursts. The hardware generates an interrupt after processing is done, and the partial results are written back to the output buffer. These results are read by the host computer to be accumulated and re-ordered to fit the result format. This module aids in algorithmic optimization by modifying the data sequence fed to the accelerator. The algorithm optimizations are discussed in detail in section 3.3.

## 3. TANOR MODULES

### 3.1. Kernel Function Pipeline Generator

The HDL code generation process for the kernel function is illustrated in Figure 3. The module starts with a MATLAB kernel specification as input. Only a subset of Matlab functions having matching hardware primitives in our library are supported. Our current primitive set is tailored to support a variety of N-body interaction kernels. The code parser and analyzer phase parses this

[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 [11].

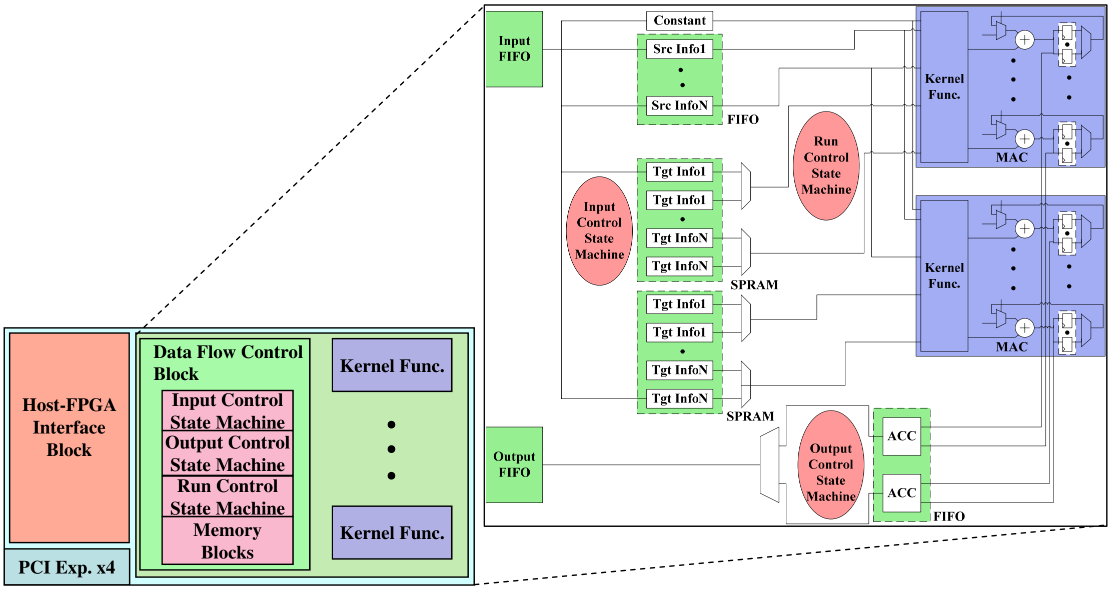


**Fig. 2**. TANOR Architecture

transformed algorithm into an intermediate representation known as an ASG (Abstract Syntax Graph). Each node of this graph represents either an input or a computation, and edges capture the datapath. More importantly, this graph also reveals the opportunities for optimizations such as CSE (common sub-expression elimination) which help the tool to come up with minimal hardware design for the target application. For example, some kernel functions may have only one output, while some kernel functions can have multiple outputs. In the case of multiple outputs, the common computations between the different outputs can be eliminated for the optimal resource usages. Timing information necessary for HDL code generation is computed by traversing the data-path graph. It utilizes the timing information of primitive hardware modules from the hardware library to generate the timing for the entire computation core. Once the timing information is available, delay elements are introduced into the data path to synchronize the operations. Finally, the data-path graph is translated into a HDL description of Kernel Functions.

Our implementation supports *IEEE-754 Floating Point Standard* and parameterized precision (user defined exponent and mantissa widths) arithmetic units. The supported operations include addition, subtraction, multiplication, division, and square root. In addition, we can support look-up-table based implementation of arithmetic functions. Area and latency of these operational units varies depending on the bit-width of the supported operations. TANOR automatically selects the arithmetic units with optimal set of parameters based on the design constraints specified by the user in the input user specification. Currently Xilinx CORE Generator is used for Floating-Point Operator[12]. Therefore the latency and resource information of Xilinx CORE Generator are used for the timing calculation and delay insertion. For this block, VHDL is used because CORE Generator only provides the simulation model of VHDL and the structural mapping is much easily implemented by *generate* sentence in VHDL for this automatic generation.

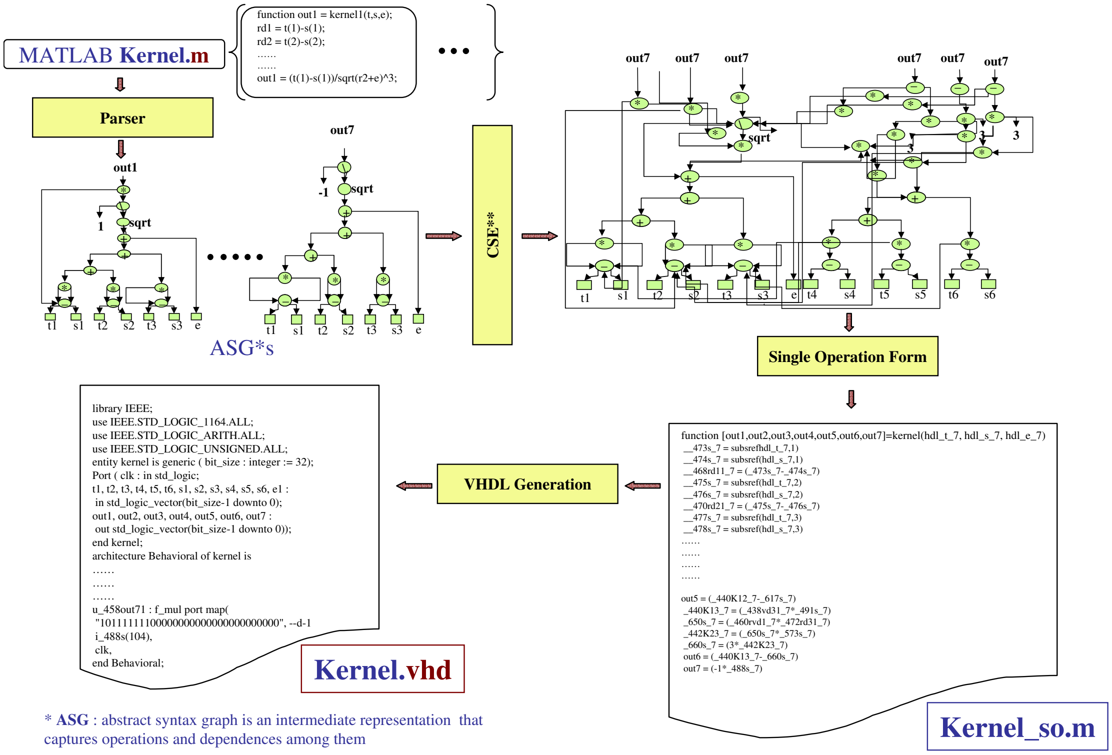
Since kernel function outputs are commonly multiplied and accumulated in several N-body interaction problems, our generator also supports a pipelined MAC block. Unless specified by the user, MAC block is implemented in single precision floating point arithmetic. However, the precision can be varied according to the requirement of the application. And the outputs of MAC block are stored in separate buffers for the accumulator to be shared during the time of floating point operation latency. MAC block can be inserted or not according to the characteristic of kernel function.

### 3.2. Data Flow Control Generator

The Data Flow Control Generator shown in figure 2 contains instantiations of memory blocks and state machines that control data steering. The memory block contains one FIFO and two single-port RAMs for storing input information. In addition to these, there are several FIFOs to store the result and the number of these FIFOs depends on the number of kernel pipelines used. There are three different state machines controlling the data flow. The first state machine takes care of the input data to be stored into correct memory locations. The second state machine controls the hardware execution and generates signals for reading the data from input FIFO and RAM, and writing the data to the output FIFOs. The third state machine decides the order of output FIFO to avoid the collision.

### 3.3. Algorithmic Optimizer

Key aspects of algorithmic optimizations that influence the hardware efficiency in N-body interaction problems are data traversal sequence and data representation. Since N-body interaction problems involve large data sets, the operations are performed in a tiled fashion. The ordering of these tiles has a significant impact on the accuracy of the computed result. To reduce computational complexity, data can be compressed to aggregate multiple source-target interaction computations into one computation[13]. When done intelligently, this can be achieved without significant reduction in numerical accuracy. Our algorithmic optimizer module can generate efficient data traversal sequences based on kernel properties,



```
function out1 = kernel1(t,s,e);
rd1 = t(1)-s(1);
rd2 = t(2)-s(2);
……
……
out1 = (t(1)-s(1))/sqrt(r2+e)^3;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity kernel is generic ( bit_size : integer := 32);
Port ( clk : in std_logic;
t1, t2, t3, t4, t5, t6, s1, s2, s3, s4, s5, s6, e1 :
 in std_logic_vector(bit_size-1 downto 0);
out1, out2, out3, out4, out5, out6, out7 :
 out std_logic_vector(bit_size-1 downto 0));
end kernel;
architecture Behavioral of kernel is
……
……
……
u_458out71 : f_mul port map(
 "10111111100000000000000000000000", --d-1
 i_488s(104),
 clk,
end Behavioral;
```

```
function [out1,out2,out3,out4,out5,out6,out7]=kernel(hdl_t_7, hdl_s_7, hdl_e_7)
 __473s_7 = subsrefhdl_t_7,1)
 __474s_7 = subsref(hdl_s_7,1)
 __468rd11_7 = (_473s_7-_474s_7)
 __475s_7 = subsref(hdl_t_7,2)
 __476s_7 = subsref(hdl_s_7,2)
 __470rd21_7 = (_475s_7-_476s_7)
 __477s_7 = subsref(hdl_t_7,3)
 __478s_7 = subsref(hdl_s_7,3)
……
……
……
……

out5 = (_440K12_7-_617s_7)
 _440K13_7 = (_438vd31_7*_491s_7)
 _650s_7 = (_460rvd1_7*_472rd31_7)
 _442K23_7 = (_650s_7*_573s_7)
 _660s_7 = (3*_442K23_7)
 out6 = (_440K13_7-_660s_7)
 out7 = (-1*_488s_7)
```

* **ASG** : abstract syntax graph is an intermediate representation that captures operations and dependences among them

**CSE: common sub-expression elimination

**Fig. 3**. Kernel Function Pipeline Generation

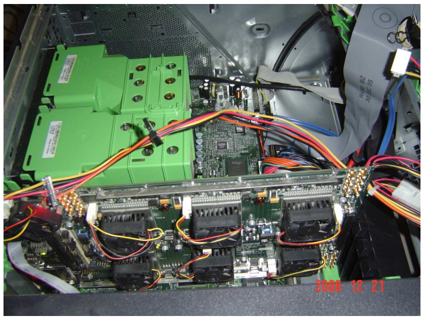


**Fig. 4**. TANOR System

desired numerical accuracy, and computational resource availability. Due to space constraints, we only elaborate the data traversal techniques used to improve numerical accuracy.

Two different traversal sequences, namely *plain and geometric tiling* are supported by our tool. In *plain tiling* the source and target information is partitioned into sets of pre-defined lengths. These sets are then compiled into a sequence of tiles in which every target set is followed by all possible source sets. For instance, during the force calculations of an N-body simulation, a summation of the force exerted on a certain target by all possible sources is performed. In astrophysical simulations the gravitational force is inversely proportional to the square of the distance between two particles. The lack of homogeneity in the spatial configuration of the particles marginally varies the amount of force exerted by different sources. This variation may result in the lack of precision if summation is not performed in a sequence that is sorted in the increasing order of magnitude, due to arithmetic rounding. Hence, traversing algorithms like *plain tiling* which do not take this factor into consideration are prone to a certain loss of accuracy in their results. *Geometric Tiling* addresses such an issue by creating an interaction list for every target particle. This interaction list contains source particles sorted in terms of their distance from the target under consideration. When tiling is performed, for each target, the source elements are ordered in the decreasing sequence of their distance from the target. Additional details of the geometric tiling are explained in [14].

## 4. CASE STUDY

### 4.1. System Configuration

The DN6000k10PCIe-4[15] logic emulation system depicted in Figure 4 is used as the target hardware platform for evaluating TANOR. We use two Xilinx Virtex2Pro-100 devices. On one of the FPGAs, the PCI Express block and data interface FIFO block are configured to provide the PCI Express 4 Lane DMA mode and the other blocks, specific to the kernel function and data flow, are configured on the second FPGA.

In the following sections we demonstrate the effectiveness and adaptability of our automated tool by developing applications using three different kernel functions, the gaussian kernel common in image processing applications, the gravitational kernel used in astrophysics, and a force calculation kernel applied in molecular dynamics.

| Kernel | Precision | Number of Operation | # of Pipelines per FPGA | Latency (Clk cycle) | Power (W) | Performance (Gflops) |
|---|---|---|---|---|---|---|
| Gravitational | e8m16 | 56 | 3 | 129 | 6.5 | 22.2 |
| Gaussian | e8m16 | 19 | 8 | 92 | 5.2 | 19.2 |
| Force Cal. in MD | e8m23 | 23 | 4 | 124 | 5.0 | 11.8 |

**Table 1**. Resource utilization, power and performance evaluation

### 4.2. Gaussian Kernel

The gaussian kernel is one of the popular smoothing algorithms for image processing and reconstruction[16]. A two-dimensional image is used for the test set. Only one kernel function is provided to specify the Gaussian kernel.

$$\mathbf{h}(\mathbf{x_i}, \mathbf{x_j}) = e^{\frac{-\|\mathbf{x_i}-\mathbf{x_j}\|^2}{2\sigma^2}}, \tag{1}$$

$x_i$, $x_j$ are the positions of $i_{th}$ pixel and $j_{th}$ pixel respectively. A taylor series expansion is used for implementing on the hardware. A second order taylor series expansion is specified for the Gaussian kernel description.

### 4.3. Gravitational Kernel

The gravitational kernel, commonly used in astrophysical N-body simulations, has been implemented. At every time step of the simulation, the gravitational acceleration force, its first time derivative, and the potential of every particle in the input domain are calculated. There are seven kernel functions calculating the magnitude of these parameters at every target particle. An input test set consists of the source and target particle locations in a three-dimensional space $R^3$. In a gravitational N-body simulation problem the gravitational force, its time derivative, and the gravitational field potential exerted at a target set **T** of particles due to the mass at a source set **S** of particles is computed as follows,

$$\mathbf{a_i} = c \cdot \sum_{\mathbf{s_j} \in \mathbf{S}} \frac{m(\mathbf{s_j}) \cdot (\mathbf{t_i} - \mathbf{s_j})}{\|\mathbf{t_i} - \mathbf{s_j}\|^3}, \tag{2}$$

$$\dot{\mathbf{a_i}} = c \cdot \sum_{\mathbf{s_j} \in \mathbf{S}} \left[ \frac{\mathbf{v_{ij}}}{\|\mathbf{t_i} - \mathbf{s_j}\|^3} - \frac{3(\mathbf{v_{ij}} \cdot (\mathbf{t_i} - \mathbf{s_j}))(\mathbf{t_i} - \mathbf{s_j})}{\|\mathbf{t_i} - \mathbf{s_j}\|^5} \right], \tag{3}$$

$$\theta_{\mathbf{i}} = \mathbf{c} \cdot \sum_{\mathbf{s_j} \in \mathbf{S}} \frac{\mathbf{m}(\mathbf{s_j})}{\|\mathbf{t_i} - \mathbf{s_j}\|}, \qquad \mathbf{t_i} \in \mathbf{T}. \tag{4}$$

Where $t_i$ and $s_j$, denote the position of the target and source particles in T and S, respectively $\|t - s\|$ denotes the Euclidean distance between the $t$ and $s$, $m(s_j)$ is the mass of the particles at location $s_j$, and $c$ is a constant. Equations 2 and 4 compute the acceleration and potential of a target particle $t_i$, and equation 3 computes the time derivative of the particle acceleration.

### 4.4. Force Calculation in Molecular Dynamics

Since the force calculation is the most time-consuming part in Molecular Dynamics simulation, it can be accelerated by implementing it in the hardware[10]. Three dimension input test is used in our evaluation, and the force calculation formula implemented is as follows.

$$\vec{f_i} = \sum_{j \neq i} \left( \frac{12A}{r_{ij}^{14}} - \frac{6B}{r_{ij}^{8}} + \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}^{8}} \right) \vec{r}_{ij}, \tag{5}$$

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

**Table 2**. The comparison with GRAPE-6

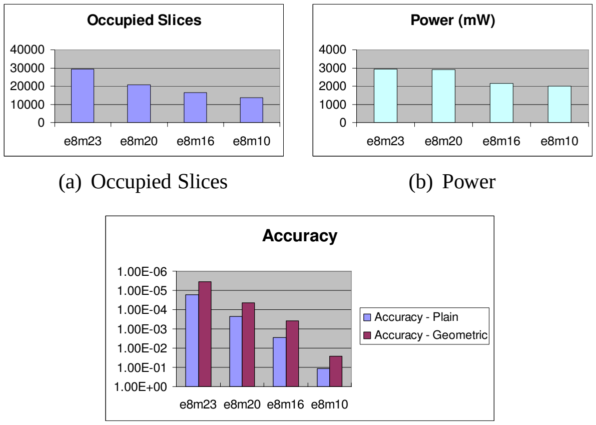
$\vec{r}_{ij}$ is the distance vector between atoms $i$ and $j$, and $r_{ij}$ is the magnitude of $\vec{r}_{ij}$. A and B are constants and $q_i$ and $q_j$ is the charge on atom $i$ and $j$.

### 4.5. Results

First, we demonstrate the versatility of TANOR by generating hardware for three different N-body interactions. TANOR requires between 2 to 3 seconds for translating the Matlab specifications to a HDL code on a Pentium 4 2.4GHz with 1 GigaByte Memory. In addition, the hardware synthesis time ranges from 1 to 7 hours for the explored designs. Table 1 shows the resource utilization, power and performance evaluation for the above three kernels. The precision column is the chosen precision for this application, e and m denote the exponential and mantissa bit widths respectively. Configurations can also be selected based on accuracy, power, or performance considerations. Some variants are shown later for the gravitational kernel. The number of operations is the total number of primitive operations used to implement each kernel function. The gravitational kernel shows the highest number of operations per data-path pipeline in Table 1 because 7 kernel functions are described for the gravitational kernel.

Number of pipelines per FPGA denotes the maximum number of the pipelined functional units that can be included in one FPGA. The gravitational kernel has a small number of pipelines per FPGA because large numbers of operations are used to implement a single pipeline. Even though the difference of number of operations is not too high between Gaussian kernel and MD force calculation, the difference in precision of the implemented units varies. The MD force calculation units which require higher precision need more resources per unit and hence have smaller number of data path pipelines in their implementation. Latency denotes the number of clock cycles per operation. In the current system, a common 125MHz clock is used for the kernel function, data flow and PCI express blocks. Power consumption numbers for the different configurations are reported using XPower[17]. Performance numbers reported were obtained through actual time measurements on our target platform for a problem size of 5K particles averaged over 20 different executions.

To evaluate the quality of the generated hardware, we used GRAPE-6 as a comparison point. We observed that the hardware



(a) Occupied Slices

(b) Power

(c) Accuracy

**Fig. 5**. Accuracy Trade-offs

generated by TANOR achieves a performance comparable to the GRAPE-6 chip in terms of FLOPS and power consumption as shown in Table 2. It should be noted that a direct comparison is difficult due to differences in technology (0.25$\mu m$ vs 0.13$\mu m$) and implementation style (ASIC vs FPGA). The results only serve as an indicator that the quality of generated output is competitive.

To demonstrate the ability to use TANOR for algorithmic and architecture exploration, we show area, power and accuracy trade-offs of different configurations for the gravitation kernel. In Figure 5(c) we show the trade-offs in accuracy achieved by the choice of two different algorithmic variants of data traversal, plain and geometric tiling. We use a double precision result from a Matlab implementation as a reference to compute the relative error of each parameterized precision implementation. It can be observed that through geometric tiling using a mantissa bit width of 16 we can achieve accuracy comparable to that of plain tiling for a mantissa bit width of 20. The single pipeline constraint is implemented to show the comparison of slices and power. Note that this decrease in the bit width translates into a 20% reduction in utilized area on the FPGA, and a 21% reduction in power consumption as shown in figure 5(a) and 5(b) due to algorithmic-architecture co-exploration.

## 5. CONCLUSION AND FUTURE WORK

We have developed a tool to automatically generate hardware for accelerating N-body simulation on reconfigurable platform. Our tool generates the required hardware modules and a software data communication interface, starting from a high-level MATLAB description. Using TANOR, we have generated a pipelined hardware accelerator that can be optimized to meet user design constraints for three different N-body interaction applications. The performance comparison with GRAPE-6 shows that the quality of generated output is quite competitive. Through variant precision and data traversal sequences, our automation flow was able to achieve a 20% reduction in resource utilization, and a 21% reduction in power consumption while maintaining comparable accuracy. We are currently working to extend TANOR to multi-FPGA architecture through the serial IO interface.

## 6. REFERENCES

[1] M. Gokhale and P. S. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Springer, 2006.

[2] "Carte programming environment," http://www.srccomp.com/CarteProgEnv.htm.

[3] "Impulsec," http://www.impulsec.com.

[4] "Catapult c synthesis," http://www.mentor.com/products/cbased_design/catapult_c_synthesis/index.cfm.

[5] "Xilinx acceldsp synthesis tool," http://www.xilinx.com/ise/dsp_design_prod/acceldsp.

[6] R. Stevens, "Future directions in computer and systems architecture for scientific computing," in *Institute for Theoretical Atomic and Molecular Physics Workshop*, 2000.

[7] J. Makino, "The GRAPE project," *Computing in Science & Engineering*, vol. 8, pp. 30–40, 2006.

[8] T. Hamada and N. Nakasato, "PGR: a software package for reconfigurable super-computing," in *International Conference on Field Programmable Logic and Applications*, 2005, pp. 366–373.

[9] 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.

[10] M. G. R. Scrofano, F. Trouw, and V. K. Prasanna, "A hardware/software approach to molecular dynamics on reconfigurable computers," in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[11] "Xilinx pci express endpoint logicore," http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-PCIEXP.

[12] "Xilinx logicore floating-point operator v3.0," http://www.xilinx.com/bvdocs/ipcenter/data_sheet/floating_point_ds335.pdf.

[13] J. Carrier, L. Greengard, and V. Rokhlin, "A Fast Adaptive Multipole Algorithm for Particle Simulations," *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 4, pp. 280–292, July 1988.

[14] 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.

[15] "Dinigroup dn6000k10pcie-4," http://www.dinigroup.com/index.php?product=DN6000k10pcie.

[16] R. C. Gonzalez and R. E. Woods, *Digital image processing (2nd ed.)*. Prentice Hall, 2002.

[17] "Xilinx xpower," http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm.

[18] A. K. G. Lienhart and R. Manner, "Using floating-point arithmetic on PGAs to accelerate scientific n-body simulations," in *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 182–190.

[19] J. F. Makino, Toshiyuki, K. Masaki, and N. Ken, "GRAPE-6: Massively-parallel special-purpose computer for astrophysical particle simulations," *Publications of the Astronomical Society of Japan*, vol. 55, pp. 1163–1187, 2003.