An FPGA Computing Demo Core for Space Charge Simulation

Jinyuan Wu and Yifei Huang

Abstract— **In accelerator physics, space charge simulation requires large amount of computing power. In a particle system, each calculation requires time/resource consuming operations such as multiplications, divisions, and square roots. Because of the flexibility of field programmable gate arrays (FPGAs), we implemented this task with efficient use of the available computing resources and completely eliminated non-calculating operations that are indispensable in regular micro-processors (e.g. instruction fetch, instruction decoding, etc.). We designed and tested a 16-bit demo core for computing Coulomb's force in an Altera Cyclone II FPGA device. To save resources, the inverse square-root cube operation in our design is computed using a memory look-up table addressed with nine to ten most significant non-zero bits. At 200MHz internal clock, our demo core reaches a throughput of 200M pairs/s/core, faster than a typical 2GHz micro-processor by about a factor of 10. Temperature and power consumption of FPGAs were also lower than those of micro-processors. Fast and convenient, FPGAs can serve as alternatives to time-consuming micro-processors for space charge simulation.**

Index Terms—**FPGA Firmware, Reconfigurable Computing**

I. Introduction

In accelerator physics, simulating charged particles' movement and positions requires large amounts of computing power[1,2]. In simulations, determining the magnitude and direction of the particles' movement requires calculating force using Coulomb's law. The following calculation shows the force on one particle if there are *n* particles in the system (where **r** is the position vector and q is the charge):

$$\mathbf{F}_i = \frac{q_i}{4\pi\varepsilon_0}\sum_{j=1}^{n}\frac{q_j\left(\mathbf{r}_i-\mathbf{r}_j\right)}{\left|\mathbf{r}_i-\mathbf{r}_j\right|^3} \quad (1)$$

The same calculation is repeated for all other particles before the positions are updated for each time step. Consequently, the number of total computation is $O(n^2)$, i.e., when number of particle increases by a factor of 10, computation time increases by a factor of 100.

The efforts of simulation can be classified into brute force approaches and non-brute force approaches. An example of a brute force approach is using graphic process units (GPU) as computing hardware to calculate all forces [3]. The tree code [1,2] is an example of a non-brute force approach in which particles far away from the seeding particle are combined together and the number of total computations is reduced to O(n*log(n)). For applications with large number of particles ($>10^6$) such as accelerator designing or high brightness electron sources, reduction of total computations is likely to be the ultimate technique. However, validating the algorithms by cross-checking with the results from brute force calculations is still useful.

In this paper, we discuss implementing the Coulomb forces computing with FPGA. A 16-bit demo core is implemented in an Altera Cyclone II device (EP2C8T144C6) [5]. The computing module with the FPGA devices is shown in Fig. 1.

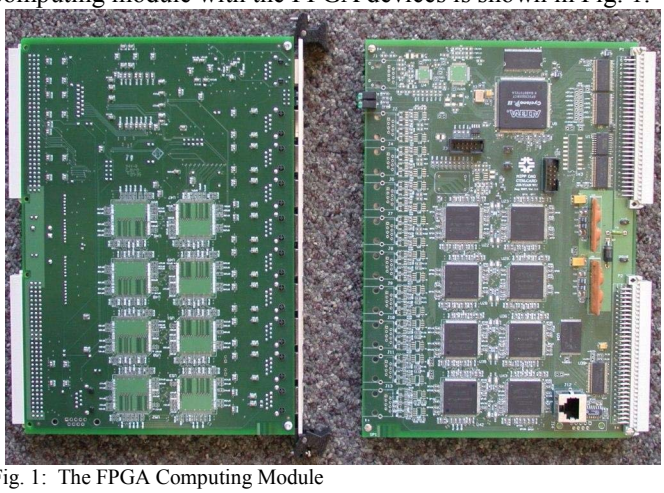

Fig. 1: The FPGA Computing Module

Each module is designed to host up to 16 FPGA devices for massive computing tasks. Another FPGA is used to interface with VMEbus, Ethernet, a synchronous dynamic RAM (SDRAM) and a FLASH memory. The interface FPGA also broadcast initial data to and collect results from the calculation FPGA chips.

For simplicity at the early stage, the computing of Coulomb forces with FPGA that we tested in this work is a brute force approach. However, it is possible to implement other algorithms in the future given that FPGA devices are flexible and reconfigurable.

II. Implementation

The block diagram of the 16-bit demo core for the space

Manuscript received October 30, 2008. This work was supported in part by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy and by Illinois Math and Science Academy Student Inquiry and Research Program.

J. Wu is with Fermi National Accelerator Laboratory, Batavia, IL 60510 USA. Y. Huang is with Illinois Math and Science Academy, Aurora, IL 60506, USA (phone: 630-840-8911; fax: 630-840-2950; e-mail: jywu168@fnal.gov).

charge simulation is shown in Fig. 2.

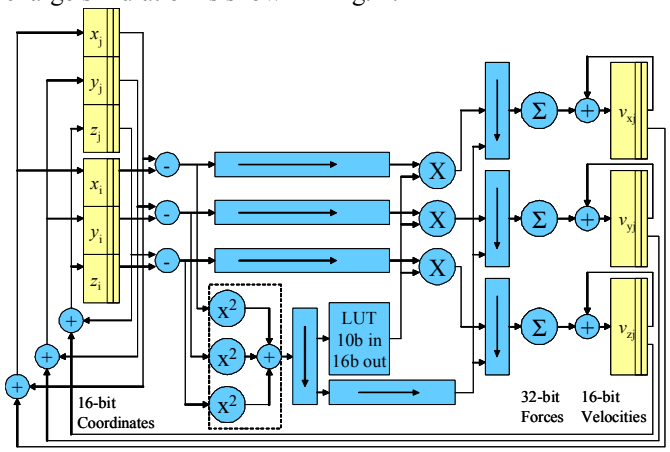

Fig. 2. The demo core for space charge simulation

The demo core is designed in pipeline fashion with an internal clock of 200 MHz. The 16-bit coordinates of 256 particles are stored inside the FPGA.

In each clock cycle, a pair of particles marked with "i" and "j" is chosen and the force between them is calculated through the pipeline. Two counters control the computing sequence by circulating the "i" index one count per clock cycle and then incrementing the "j" index after all i-particles have been traversed through. The coordinates of each pair of the particles are subtracted to find the differences Δx, Δy and Δz as the components of $(\mathbf{r}_j-\mathbf{r}_i)$. The differences are then fed into a sum-of-squares block to calculate the square of the distance between the two particles $|\mathbf{r}_j-\mathbf{r}_i|^2$. This value is used to address a lookup table (LUT) of inverse square-root cubed. The output of the lookup table is then multiplied with the Δx, Δy and Δz to calculate the component of the force. Note that differences of the coordinates are delayed in a pipeline with the number of stages matching the stages of the sum-of-squares and the LUT. The components of the force on the seed particle marked with "j" are accumulated internally in 32-bit precision.

Because an extensive amount of computing resources and time would be needed to calculate the inverse square-root cubed, the value is calculated by checking a memory lookup table. Instead of using 32-bit r-squared values to address in our table, we utilize only the nine or ten most significant bits to save memory resources. (If 32 bits were used to address the lookup table, a memory with 4G words would be needed, which would be too large for typical FPGA devices.) This is performed by removing higher bits of logic 0, narrowing the value stage-by-stage from 32 bits, to 16 bits, to 12 bits, to 10 bits with a logarithmic shifter. (Sometimes it is called "barrel shifter". The term "logarithmic shifter" is chosen here to emphasize its implementation scheme and small resource usage.) The 16-bit output value from the table is multiplied by Δx, Δy, or Δz to form force components. Finally, the force components are shifted back to the corrected scale by padding logic 0 bits (or logic 1 in higher bits if the value is negative.). The input for the inverse square-root cubed LUT is shifted downward in increments of 2. If it is shifted downward by 2n bits, i.e., divided by $2^{(2n)}$, the inverse square-root cubed LUT output is over-scaled by a factor of $2^{(3n)}$. The force components are to be shifted downward by 3n bits to recover back to the corrected scale.

The lookup table is shown in Fig. 3.

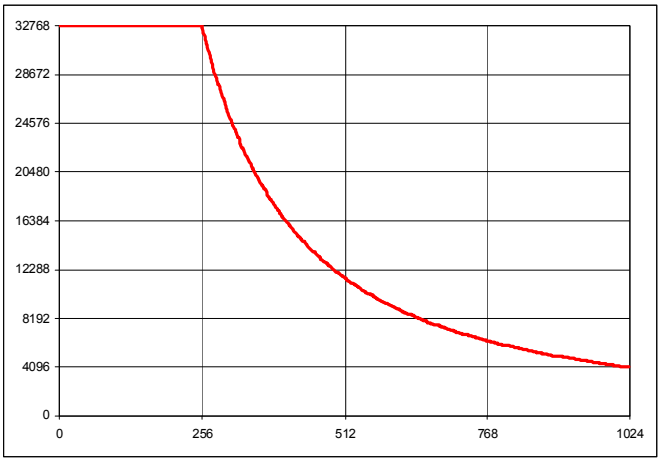

Fig. 3. The lookup table of the inverse square-root cubed

Since the input for the inverse square-root cubed LUT is shifted downward in increments of 2, the address for the LUT is in the range from 256 to 1023 if the raw sum-of-squares is larger than 255. The only possible case for the LUT address to be 0-255 is when the two particles are placed too close than physically possible. In order to account for small r-squared values that would cause the inverse square root cubed value to approach infinity, output values are limited at the hexadecimal 7FFF. Points with the same x, y, or z position always result with a force of 0 because Δx, Δy and Δz are multiplied into the table output result.

Since the velocity change in a time step is directly proportional to the acceleration or the force, the accumulated net force components are added together and stored in the 16-bit velocity memories.

The x, y, and z positions are updated after all particles' velocities have been calculated.

The two counters control the entire sequence. One updates each clock cycle while the other updates after calculating the net force for a particle and updating its new velocity. After this two-layer nested loop, interactions between all particles are calculated. Then a single loop runs to update position memories, and the iteration for a time step is complete.

In this demo core, the updating of velocity components and position coordinates is performed inside the calculating FPGA as shown above. However, in the future implementation, the accumulated force components are to be read back to the interface FPGA to update the velocity components and coordinates of the seed particle since this process is only performed once after the interactions between the seed particle and all other particles are calculated.

It should also be pointed out that 1/4 of total memory locations between 0-255 can be used more efficiently. One possibility is to increase input resolution when the input value is 256.0 to 511.5 by adding an extra bit into the LUT address lines, which could be an improvement for the future implementations.

III. Test Results

We have used our demo core to compute a 2-electron or a 2-proton system to check with results calculated in conventional computers. A 256-partile system with same charge is also computed as a demo.

A. The 2-Particle System

Two identical particles are initially separated by 256 nm at rest and the separation increases in time as shown in Fig. 4(a).

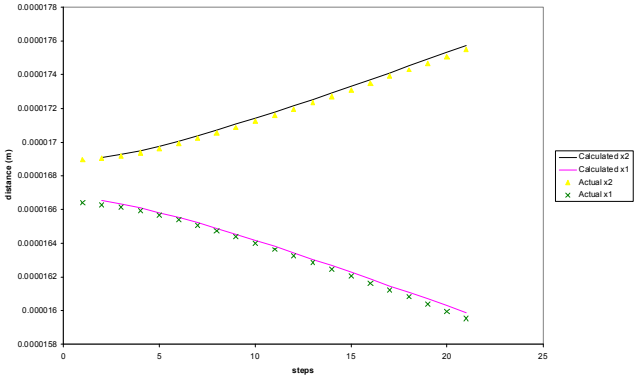

Fig. 4. Separations of a 2-particle system

The points represent the coordinates calculated with the FPGA core of the particles and the curves are coordinates calculated with regular computer. From the rate of separation, the scale of the time step can be calculated as shown in Table I. For electrons the each time step represents 1.4ps while the proton time step is 61ps.

TABLE I
PARAMETERS OF PARTICLES AND TIME STEP SIZE

Particle Type	Charge (C)	Mass (kg)	Time Step (s)
Electron	1.602E-19	9.109E-31	1.42807E-12
Proton		1.673E-27	6.12017E-11

The values calculated in FPGA are systematically lower than that calculated in regular computer. A possible reason is that the rounding scheme in the FPGA is bit truncating.

B. The 256-Particle System

As an initial study of the performance of the FPGA demo core, we have computed the expansion of a 256-particle system. 256 identical charged particles are placed in a 4x4x16 grid and their initial positions are as shown in Fig. 5(a). The initial velocities of all particles are set to be at rest. After 20, 30 and 35 time steps, the particle system expands as shown in Fig. 5(b)-(d). The particles are packed closer initially in horizontal dimension. Therefore, the outer particles gain higher horizontal velocity components, which cause the system to expand more rapidly in horizontal direction.

The lengths in the simulation are integers and the unit can be chosen by users. If 1 nm is chosen as integer unit, the time step scale is given in Table I.

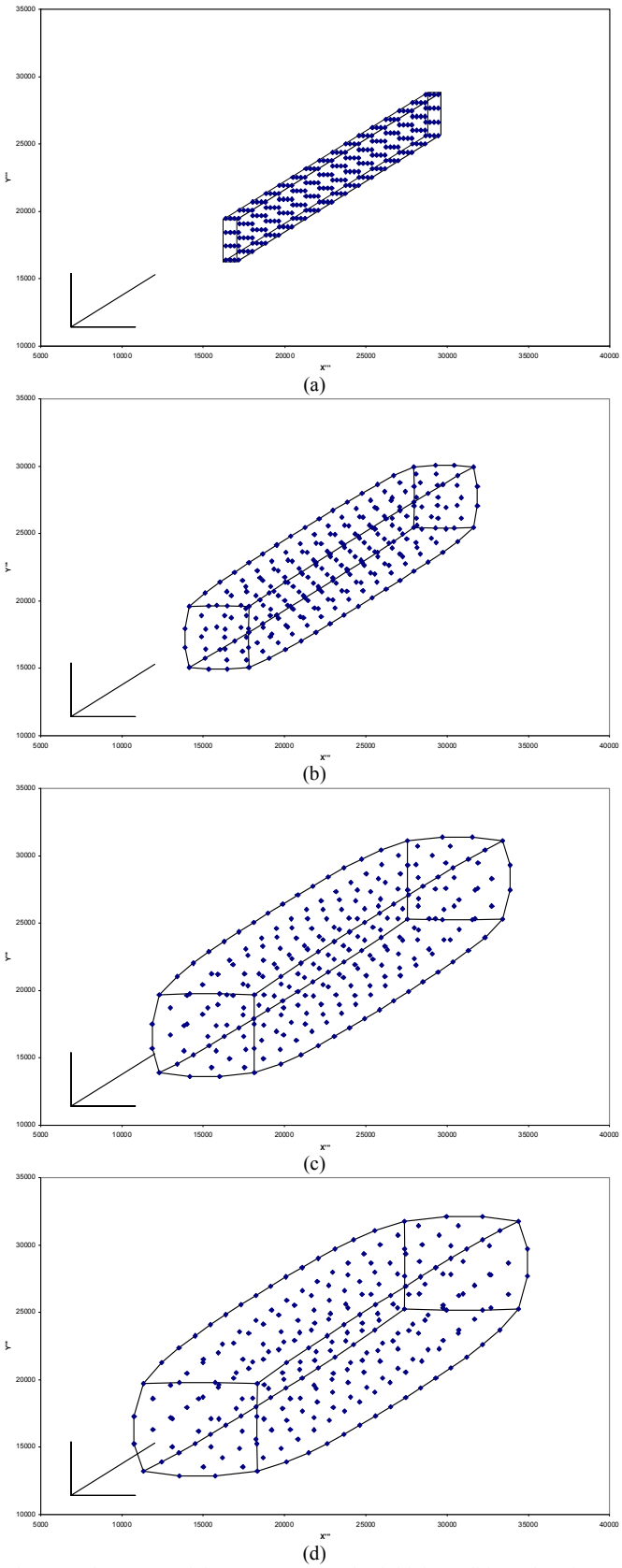

Fig. 5. The 256-particle system (a) at the initial condition, (b) after 20 iterations, (c) after 30 iterations, (d) after 35 iterations

IV. Comparison with a Microprocessor

A similar computation has been bench marked in a microprocessor and compared with the FPGA core. The power consumption and operating temperature of the FPGA are also measured.

A. Computation Speed

A short C-code calculating the Coulomb forces between particles is written and compiled for an Intel Core 2 Duo CPU running at 2.2 GHz. The compiling and running environment is the native UNIX system in an Apple MacBook. The computing times for various numbers of particles are shown in Fig. 6. The computing time for FPGA core running at 200 MHz internal clock is also shown.

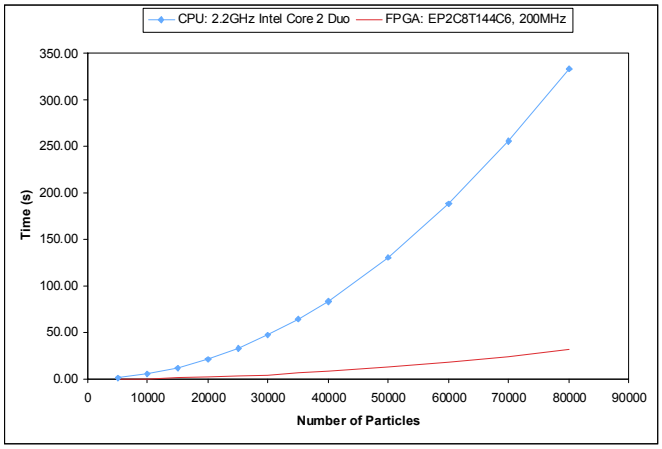

Fig. 6. Comparison between the FPGA demo core and a microprocessor

During the run, the calculation task is assigned to one of the two CPU cores and the task typically takes more than 97% of the core time during the process. If two processes are submitted simultaneously, the two processes are assigned to two cores and the computing times for each process are about the same. So the comparison given in Figure 6 is between a CPU core and an FPGA core.

The insignificance of system overhead can also be seen from the dependence of the computing time on the number of particles. It follows quadratic curve without large offset or linear term that large system overhead would produce.

It can be seen that the FPGA core is about a factor of 10 faster than a CPU core. The FPGA core calculates the force between a pair of particles each clock cycle through a pipeline. The CPU software make the same calculation over multiple clock cycles (It seems to be >100 clock cycles). Although the FPGA clock is a factor of 11 slower than the microprocessor, the FPGA core is still faster than the CPU in terms of useful calculation.

B. Power Consumption and Temperature

Power consumption of entire FPGA computing module and the FPGA operating temperature are measured and shown in Table II.

The current is measured in the wire supplying entire module before the on board DC to DC converter that produces +1.2V and +3.3V for FPGA devices from +5V. The power consumption includes 8 calculation FPGA chips, a data interface FPGA and several other chips (SDRAM, FLASH RAM and interface buffer etc.). For comparison, typical microprocessors at 2 GHz consume 40 to 100 W.

TABLE II
Power Consumption of Module and FPGA Temperature

Power Supply Voltage: 5.19 V, Room Temperature 23.1 °C			
Operating Condition	Current	Power	Temperature
Idle	0.66 A	3.4 W	24.5 °C
Continuous Computing	0.87 A	4.5 W	36.8 °C

The FPGA temperature is measured in open air without cooling fan or thermo sink.

V. Summary and Discussions

Unlike what is claimed in some literature, FPGA devices are not as cheap and fast as their counter part of micro-processors. As the cost of flexibility, the transistor usage of implementing logic functions is not efficient in FPGA. Therefore directly repeating schemes in micro-processors is not a winning line. It is necessary to search for the architecture suitable for FPGA implementation by taking advantage of ultra-flexibility of the FPGA devices.

VI. Conclusion

A 16-bit FPGA demo core for computing Coulomb force for space charge simulation has been implemented and tested. The core is about a factor of 10 faster than a 2.2 GHz CPU core and consumes about 0.5 W per core. Additional future work includes implementing the computing function to higher precision (e.g., 32-bit coordinates) and integrating multiple FPGA cores and modules into a larger system.

Acknowledgement

The authors would wish to express thanks to the Student Inquiry and Research program at the Illinois Math and Science Academy and Education Office of Fermi National Accelerator Laboratory for guidance and service through the research process.

References

[1] J. Barnes and P. Hut, "A Hierarchical O(NlogN) Force-Calculation Algorithm," *Nature*, vol. 324, pp. 446–449, Dec. 1986.

[2] C. Coleman-Smith, H. Padmore, & W. Wan, "Limits to Electron Beam Emittance from Stochastic Coulomb Interactions" submitted to *Physics Review Letter,* 2008.

[3] T. Hamada & T. Iitaka, "The Chamomile Scheme: An Optimized Algorithm for N-body simulations on Programmable Graphics Processing Units," ArXiv Astrophysics e-prints astro-ph/0703100, 2007.

[4] K. Asanovic, "Advanced CISC Implementations: Pentium 4," Presentation, Massachusetts Institute of Technology Course 6.823, 2002.

[5] Altera Corporation, "*Cyclone II Device Handbook*", (2007) available via: {http://www.altera.com/}