2010 13th IEEE International Conference on Computational Science and Engineering

Revealing Feasibility of FMM on ASIC: Efficient Implementation of N-Body Problem on FPGA

Zhe Zheng, Yongxin Zhu, Xu Wang, Zhiqiang Que, Tian Huang, Xiaojing Yin, Hui Wang, Guoguang Rong
School of Microelectronics
Shanghai Jiao Tong University
Shanghai, China
{zhengzhe, zhuyongxin, xuwang, quezhiqiang, huangtian, yinxiaojing, wanghui, guoguangrong}@ic.sjtu.edu.cn

Meikang Qiu
Department of Electrical and Computer Engineering
University of Kentucky
Lexington, USA
mqiu@engr.uky.edu

Abstract—**FPGAs have been improved significantly in terms of performance and capacity over the last 20 years. The scale of FPGA based design also sparked off the demands for high-level synthesis to handle complicated applications. A well known intricate application is the FMM (Fast Multipole Method) algorithm of N-body problem, which is so complicated that it was not implemented on FPGA as reported in literature. In this paper, we take high level modeling and design tools, i.e. Simulink and System Generator to implement major modules in FMM algorithm to solve the N-Body problem on FPGA. Besides the impressive performance speedup on FPGA, we improve the efficiency by merging the circuits for the common logic among modules in the algorithm. Our experience in efficiently implementing the FMM algorithm will be taken as a useful reference for researchers working on FPGA applications as well as high performance computing.**

I. INTRODUCTION

N-Body problem is a classic high performance computing application to simulate the evolution of *N* celestial bodies, given the initial positions, masses, and velocities of particles. Their subsequent motions as determined by classical mechanics, i.e., Newton's laws of motion and Newton's law of gravity [1]. It can be applied to macroscopic celestial objects and also to microcosmic molecules or atoms. A large number of physical systems can be studied by simulating the interactions between the particles. Each particle influences every other particle in a certain system, which is based on an inverse square law such as Newton's law or Coulomb's law. The system described above is widely used in astrophysics, plasma physics, molecular dynamics, fluid dynamics and so on. Since the simulation involves following the trajectories of motion of a collection of N particles, the issue is called N-Body problem.

Through studying and researching the N-Body problem continuously, many mature and reliable algorithms are proposed these years. PP (Particle-Particle) algorithm, which requires $O(N^2)$ work per iteration, is to calculate the interaction between each particle directly. The advantage of PP algorithm is high accuracy and easy to parallel-processing. However, it has disadvantage obviously, which is the complexity of calculation especially when the scale of problem is very large. In order to reduce the time consumption and improve the computational performance, other algorithms are proposed. The $O(N^2)$ cost can be reduced to $O(N_g \log N_g)$ by using the PM(Particle-Mesh) algorithm [2] where N_g represents the size of the discrete-mesh or to O(NlogN) by using the BH(Barnes–Hut) algorithm [3]. But the computational accuracy of the BH algorithm is 0.01 and the scaling coefficient is considerably large. Another algorithm used frequently is FMM [4] Greengard and Rokhlin proposed, which is to calculate the potential of each point through level-classification and multipole expansion of potential function. This algorithm reduces the time complexity to O(N) and attains arbitrary precision. So the calculation complexity and computational accuracy in FMM algorithm are dominant compared with other algorithms in high performance computing. Furthermore, some hybrid algorithms are forwarded such as P3M(Particle-Particle Particle-Mesh) algorithm [5] and TreePM algorithm [6] which are consisted of PM algorithm calculating the long-range interaction and PP algorithm calculating the short-range interaction.

N-Body problem is expanding unceasingly and has reached the scale of millions and even billions. Therefore, this problem has gradually challenged the calculating ability and processing performance and also become one of the most influential and most challenging problems in the field of high-performance computer. Many super-computers are intended to implement this problem. The most representative super-computer is GRAPE-DR, which is joint-developed by the Tokyo University, National Astronomical Observatory and Physical-Chemical Research Institute in Japan. It consists of 4096 GRAPE-DR processor chips, each of which contains 512 cores operating at the clock frequency of 500MHz and the peak speed of a processor chip is 512Gflops in single-precision type and 256Gflops in double-precision type [7]. Another team researching this problem practically performs a stellar simulation with 1.6 billion particles on a GPU cluster using 256 NVIDIA GeForce 8800GTS GPUs and obtains a sustained performance of 42.15Tflops [8]. It is also known that many several successful implementations using SIMD parallel processing such as Illiac-IV [9], Goodyear MPP [10], TMC CM-1/2 [11] and so on.

In this paper, we shall present our implementation of some key modules in FMM algorithm on FPGA using Simulink and SysGen. Simulink provides an integrated environment for system modeling, simulation and synthesis with many basic modules available to users. SysGen can convert the algorithm into a synthetic and reliable hardware system. These two tools play an important role in FPGA application design. Jonathan Phillips [12] has used LJP(Lennard-Jones Potentials) to solve this problem based on dynamic load-balancing processor architecture on FPGA. Gerhard Lienhart [13] also has researched this problem focusing on the floating-point algorithm.

As far as we know, there is no report on complete implementation of FMM algorithm on FPGA, not to mention FMM on ASIC. Hence doubts arise over the feasiblity of FMM on FPGA or ASIC. In this paper, we shall show how to solve N-Body problem by covering major issues regarding feasibility, i.e. key modules design, hardware architecture, resource utilization and difficulty handling in FMM algorithm. The specific processor chip GRAPE-DR just mentioned got much better performance than previous implementation on the N-Body problem even if algorithms other than FMM were adopted by GRAPE-DR. Therefore we think it is a must to implement FMM algorithm on ASIC. Our work will reveal the feasibility of FMM algorithm on the N-Body problem with efficient implementation on FPGA before implementation on ASIC. The results we got demonstrate that the implementation of FMM on FPGA can not only accelerate in performance but also save the resources to improve efficiency. The research we have endeavoured would be valuable for future research in high performance computing.

The remainder of this paper is organized as follows. In Section II, we present the principle of the FMM algorithm. Some key modules in FMM algorithm will be proposed and the details of the design of these modules will be shown in Section III. Further research and innovation about these key modules will be shown in Section IV. Results of implentation on FPGA through hardware co-simulation will be presented in Section V. In Section VI, we will analyze the result and make a conclusion.

II. THE PRINCIPLE OF FMM ALGORITHM

FMM algorithm sorts the particles into hierarchy boxes, which can be divided into long-range and short-range. For the particles in short-range, we just use PP algorithm directly to calculate the interaction. On the other hand, we calculate potentials or forces between boxes by ME(Multipole Expansion) and LE(Local Expansion) for the particles in long-range. We get the sum of short-range force and long-range force finally. This algorithm belongs to a classical fast summation algorithm and the steps of FMM algorithm can be described as followed [4].

Step1: Construct Tree

Divide the space into long-range and short-range.

Step2: P2M (conversion of particles into multipole expansion)

Form multipole expansions of potential field due to particles in each box about the box center at the finest mesh level.

Step3: M2M (Multipole to Multipole)

Form multipole expansions about the centers of all boxes at all coarser mesh levels, each expansion representing the potential field due to all particles contained in one box.

Step4: M2L (conversion a multipole expansion into a local expansion)

Form a local expansion about the center of each box at each mesh level. This local expansion describes the field due to all particles in the system that are in the interaction list, which means the particles are not contained in the current box or its nearest neighbors.

Step 5: L2L(translation of a local expansion)

While the local expansion is obtained for a given box, it is shifted, in the second inner loop to the centers of the box's children, forming the initial expansion for the boxes at the next level.

Step 6: L2P(conversion of a local expansion into particles)

Evaluate local expansions at particle positions.

Step 7: PP(particle-particle)

Compute potential or force due to nearest neighbors directly.

Step 8: Summation

For each particle, add direct and far-field terms together.

The flow of FMM algorithm is shown in Fig. 1. The left part of the tree is described the process of upward sweep including P2M and M2M. The right part of the tree depicts the process of downward sweep including M2L, L2L and L2P. Considering the Quad-Tree is overcrowded to be shown, so we use the Binary-Tree in Fig. 1 to illustrate the FMM algorithm, which serves as the division in 1D space. FMM algorithm is an approximate algorithm because the result and time-consumption is proportional to coefficient p, which is the order of Taylor Formula. The source of error in FMM is introduced by approximating the kernel and truncating the expansions [14].

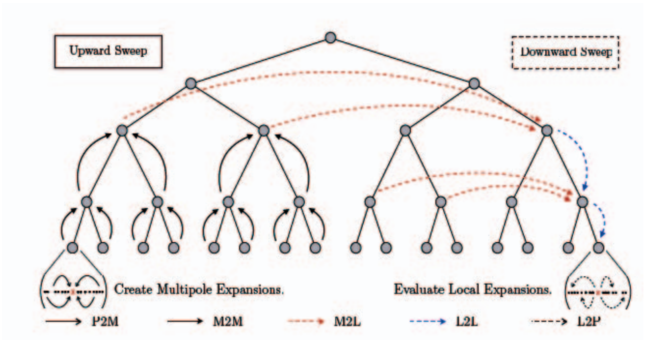

Figure 1. Schematic workflow of FMM algorithm.

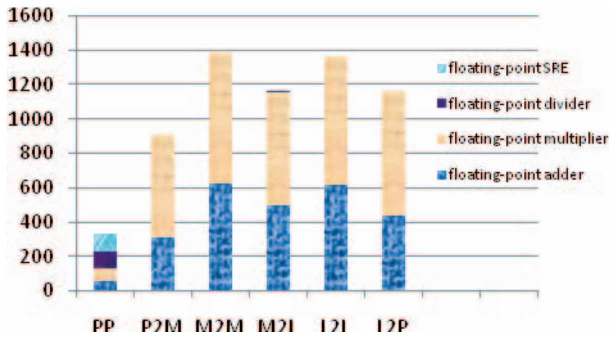

Figure 2. Resource utilizations for modules in FMM (p = 20).

From the principle of FMM algorithm introduced above, we know the core of this algorithm is to calculate multipole expansions, local expansions and conversion between them to obtain all the pair-wise forces. FMM algorithm has distinct advantages that it only costs O(N) to finish all the calculation mentioned above. Furthermore, the precision of FMM algorithm can be adjusted by the number of expansion terms, namely the order of Taylor Formula. These two main characters of FMM algorithm bring out that it can be used to solve the N-body problem efficiently. Thus, we choose FMM algorithm to do our research due to its suitability for typical application analysis in the field of high performance computer.

Besides, through the analysis on the computing formula and pipeline stages of all calculation modules, we added up the resource utilization of floating-point arithmetic unit in each module in FMM algorithm, which is related to p value, i.e. the order of Taylor Formula. Fig. 2 shows the statistics of resource utilization of different modules when p is 20. Except that PP module which is to calculate the near-force in short-range uses floating-point SRE(Square Root Extractor) and floating-point divider, other five modules which are to calculate the far-force in long-range just use floating-point adder and floating-point multiplier. We can know clearly from Fig. 2 that the resource utilization of these five modules are alomost the same. Moreover, the comuting formula of these five modules are similar. Based on these similarities, we decide to focus on two key modules L2L and L2P, which we shall elaborate below, because these two modules are representative both in calculation performance and resource utilization. It is feasible that other modules including M2M and M2L can be designed and implemented through the same method. In later part of this paper, we shall show our design of L2L module and L2P module and these two modules' implentataion on FPGA in later sections. Also we explore the parallelism in these module calculation in FMM algorithm by implementing it with a flexible design based on a hardware pipeline. Considering that the computing formula and the resource utilization are similar, other modules in FMM algorithm such as P2M or M2M can also be implemented in the same way which we shall not discuss in this paper.

III. Design of L2L Module and L2P Module

TABLE I. All Data Information Storage List

Particle Information	**Box Information**	**ME**	**LE**	**Force**
Number	Number	Box Number	Box Number	Particle Number
Coordinate	Center Coordinate	Value	Value	Long-range force
Mass	Base Address	Level of Tree	Level of Tree	Short-range force
Velocity	Level of Tree			Summation
Box Number	Neighbour List			
	Interaction List			
	Amounts of Particle			

A. Data Structure and System Diagram

At the beginning of the design and implementation, we should comprehend the data structure of the algorithm including data transmission and data storage. The input data of the whole algorithm are particle information and box information. Particle information involves particle coordinate, particle mass, particle velocity and so forth. Box information includes box number, box center coordinate, level of tree, neighbour list, interaction list and so on. During the course of calculation, ME and LE coefficients should be stored correctly as well as the result of calculation. These information is listedby the following data structure shown in Table I.

The format of data in Table I above is specified as followed. The number of boxes and particles is 32-bit unsigned integer. The coordinates of particle and box, the velocity of particle and the force on particle are all of 128-bit double-precision type, where one 64-bit coordinate is on behalf of X-direction and the other one is on behalf of Y-direction. The particle mass is of 32-bit single-precision type. ME and LE coefficients are both p-dimensional double-precision type complex vector.

According to the flow of FMM algorithm introduced in Section II and data structure described above, the whole system logic diagram could be shown in Fig. 3. The whole system is composed of input memory interface, memory on chip, PP module which calculates the short-range force, P2M、M2M、M2L、L2L and L2P modules which calculate the long-range force and a summation module which outputs the final result. We shall focus on the L2L and L2P modules, which are in the red border block. They belong to downward sweep in the FMM algorithm and the goal of the two modules is to calculate LE coefficient of each particle in the box, which is at the finest level and to convert into the long-range force affecting the target particle.

B. Design of L2L Module

1) Calculation Formula: The calculation formula of L2L module is the following expression,

$$b_l' = \sum_{k=l}^{P-1} b_k \times C_k^l \times (Z_1 - Z_c)^{k-l} \quad (1)$$

where b_l' is the result of L2L module, which is LE

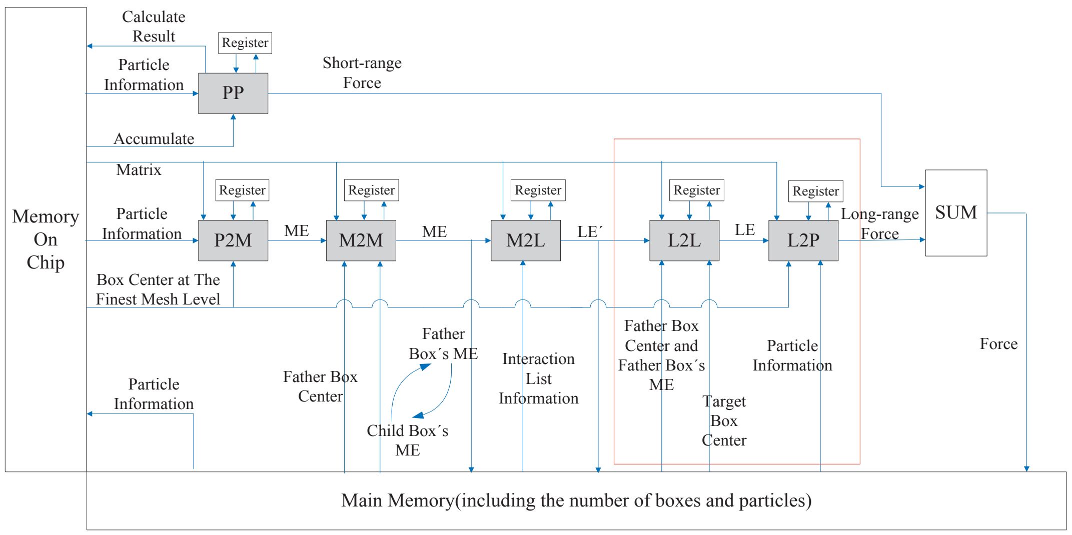

Figure 3. Diagram of the overall system logic.

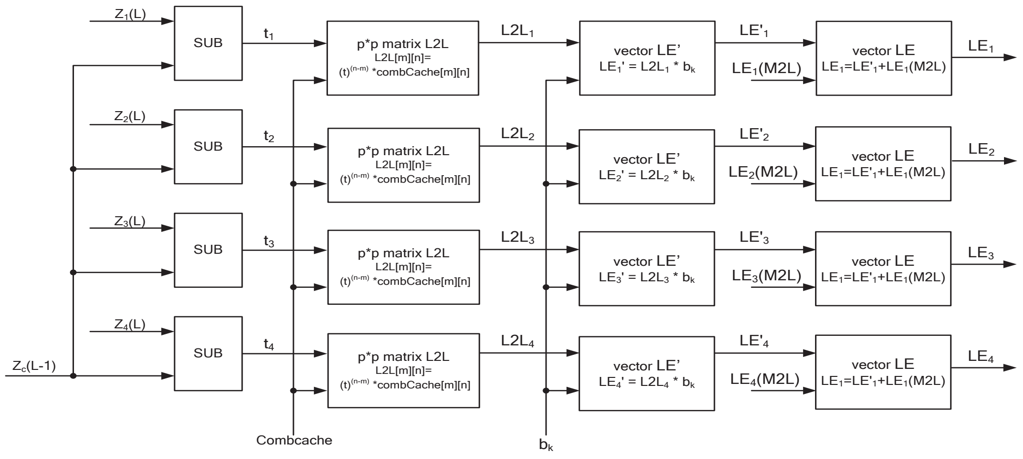
Z1(L)
Z2(L)
Z3(L)
Z4(L)
Zc(L-1)
SUB
t1
t2
t3
t4
p*p matrix L2L L2L[m][n]= (t)(n-m) *combCache[m][n]
L2L1
L2L2
L2L3
L2L4
vector LE' LE1' = L2L1 * bk
LE'1
LE1(M2L)
vector LE LE1=LE'1+LE1(M2L)
LE1
LE2
LE3
LE4
Combcache
bk

Figure 4. Hardware implementation diagram of L2L module.

coefficient of target child box and b_k is the LE coefficient of father box. $C(k,l)$ is the combination, which is stored in memory on chip. Z_C and Z_1 are the center coordinates of father box and child box respectively.

2) Calculation Process: We can know that the inputs of L2L module are the center coordinates of father box and child box, which are represented as Z_C and Z_1 in (1) as well as LE coefficient of father box, which is represented as b_k in (1) from Fig. 3. The output of L2L module is LE coefficient of child box, which is described as b_l in (1). At the beginning of L2L module calculation process, the center coordinate of father box is subtracted from the center coordinate of child box to calculate the distance between them. Then we would get a set of data, which is from square value to p-th power value of the difference just obtained. These data are multiplied with the matrix whose elements are the combinations, which are already stored in memory on chip. And the product would also be stored in a matrix and be multiplied with the LE coefficient of father box which is a vector. Finally, the result of multiplication is added with another LE coefficient from M2L module. The whole process completes a calculation between a father box and one of its child boxes. Each father box has four child boxes in 2-D, so four identical processes could be operated concurrently. An index representing the relative position of the target box and source box is calculated while the process described above has completed. In addition, from the view of tree structure, the process would be started from the third level to the last level where child box located. After all child boxes finished L2L transform in L level, these coefficients obtained are rotated back to the memory and the LE coefficient of L-1 level would be removed from the LE storage list in order to conserve resource.

3) Hardware Architecture Design: Fig. 4 shows the sketch of hardware implementation diagram of L2L module. It illustrates the calculation process between one father box

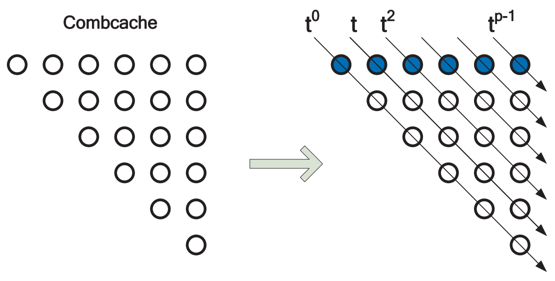

Figure 5. Schematic of matrix calculation.

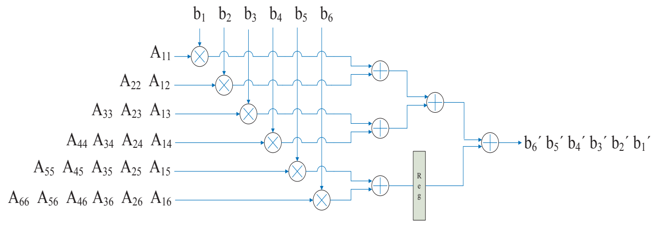
b_1 b_2 b_3 b_4 b_5 b_6
A_{11}
A_{22} A_{12}
A_{33} A_{23} A_{13}
A_{44} A_{34} A_{24} A_{14}
A_{55} A_{45} A_{35} A_{25} A_{15}
A_{66} A_{56} A_{46} A_{36} A_{26} A_{16}
Reg
b_6' b_5' b_4' b_3' b_2' b_1'

Figure 6. Architecture of MAC.

and its four child boxes clearly as well as some mathematical operation modules, which are submodules of L2L.

The center coordinates of father box and child box from main memory are subtracted as complex number in the formx+yi which x and y reprensent horizontal and vertical coordinates respectively. Here x and y are both 64-bit double-precision type so 256-bit width of input data interface is needed. From the point of view of resource, complex subtraction requires two double-precision type adders, which correspond to real and imaginary parts. The next to calculate is the power of the difference from subtraction, which is a 128-bit complex number. We could expand the power-calculating process in order to achieve pipeline operation, which means getting the (p-2) power values from square value to p-th power value in one cycle. So it requires 4*(p-2) double-precision type multipliers and 2*(p-2) double-precision type adders in all. After all powers are ready, they multiply the combination matrix, which is stored in memory on chip. This combination matrix is an upper triangular matrix because m is less than n in C(n, m) as Fig. 5 shows. The capacity of memory required for this matrix is (p(p+1)/2)64 bits. Each power would be multiplied with the elements which are diagonal direction in the matrix. Then the capacity of memory required for the result is another (p(p-1)/2)64 bits because the elements of main diagonal direction are unchanged. From the point of view of resource, it requires 2(p-1) double-precision type multipliers to achieve multiply between a real and a complex. The product of previous submodule and the LE coefficient of father box are MAC to figure out LE coefficient of child box. The procedure of this operation is shown as Fig. 6. A_{ij} means the product of previous submodule, and b_k stands for the LE coefficient of father box, and b_l' represents the result of the current submodule. The elements of each row in the matrix and the LE coefficient of father box are both input in parallel. The summation of these products is exactly an LE coefficient of child box which is output in serial. In this submodule, it is required 4p double-precision type multipliers and (4p-2) double-precision type adders in all.

The last step in L2L module is an addition between LE coefficient of child box just obtained and another LE coefficient from M2L module. The output is a 128-bit width. What we have demonstrated above are details of design of L2L module. The hardware architecture of this module is built using Simulink and SysGen as Fig. 7 shows while coefficient p, the order of Taylor Formula, is five.

C. *Design of L2P Module*

The calculation formula of L2P module is the following expression,

$$b_1 = \sum_{k=1}^{P-1} a_k \times k \times (Z_i - Z_1)^{k-1} \tag{2}$$

where b_1 is the result of the L2P module, which is long-range force on the target particle. And a_k is the LE coefficient of the box where the target particle is located. The constant k is stored in memory on chip. Z_1 and Z_i are the center coordinates of the box and the target particle respectively.

As L2P formula is similar with that of L2P, we just aim at the design of L2P, which is different from what we have analyzed in last subsection. It is clearly that the inputs of L2P module are the center coordinates of the box, which is at the finest level and the target particle as well as LE coefficient of the box. The output of L2P module is long-range force on the target particle. Due to the fact that the calculation process of L2P module is similar to that of L2L, we no longer elaborate the whole process and the hardware implementation other than the differences.

One difference emerges in the submodule, which is to achieve the multiplication between power results and data in the memory on chip. In L2P module, it only requires (p-1) data to store in the memory on chip, namely, the capacity of memory on chip is 64*(p-1) bits. Besides, these data multiply with the output of previous submodule without any control signal. Another difference is that L2P module is not required the last add submodule because the long-range force on target particle is calculated after MAC module. The hardware architecture of L2P module is almost same to that of L2L module. We would not give the unnecessary details of design only to present the hardware architecture of L2L module, which is built through using Simulink and SysGen as Fig. 8 shows while coefficient p, the order of Taylor Formula, is five. Considering these two hardware architectures are very similar and the resource utilization is almost the same as well, we could combine them to compose a merged module, which could accomplish L2L calculation and L2P calculaton by means of some control signals.

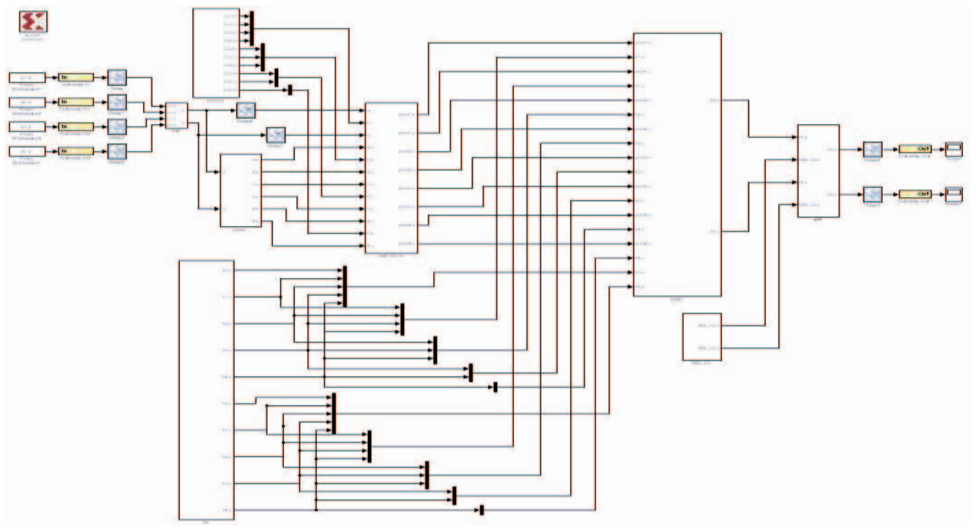

Figure 7. Overview of the whole L2L module (p = 5).

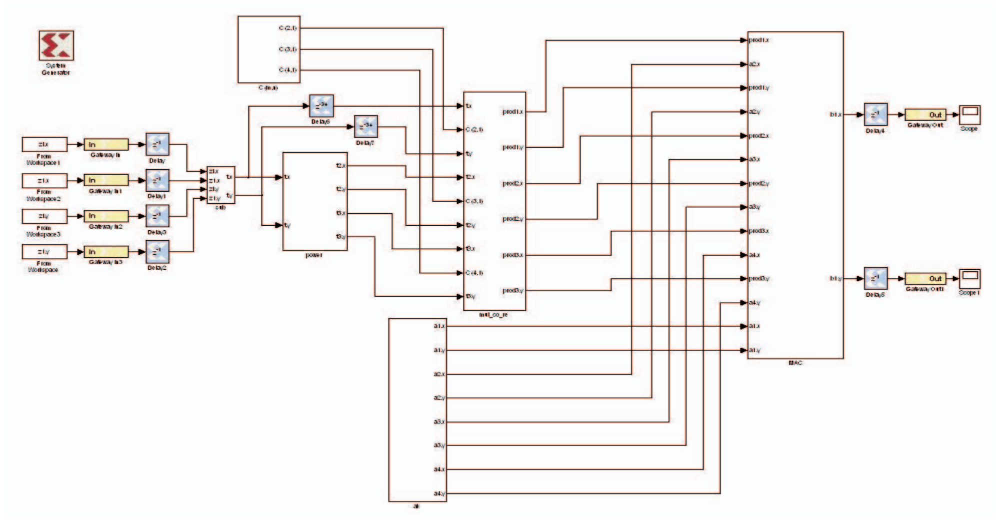

Figure 8. Overview of the whole L2P module (p = 5).

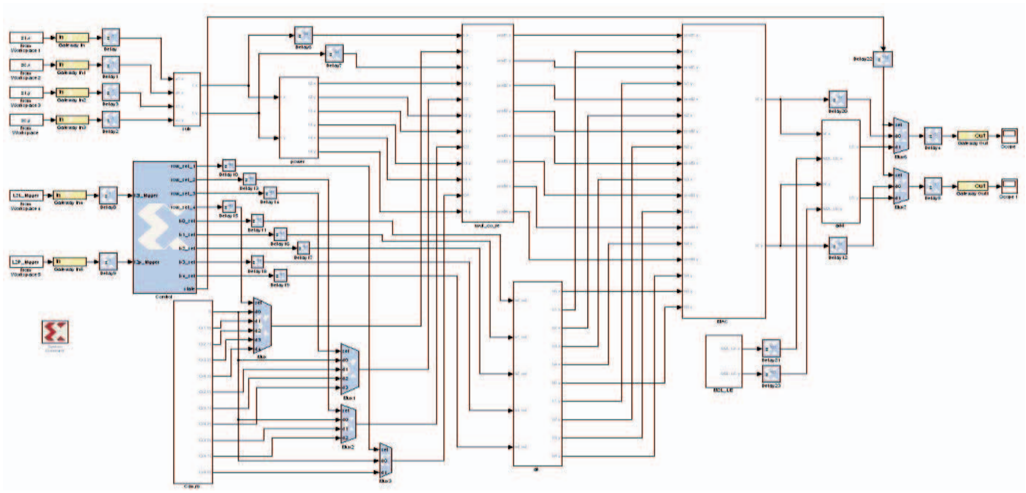

Figure 9. Overview of the whole merged module (p = 5).

IV. Design of the Merged Module

For the sake of synthesis of these two calculation modules we have introduced, we shall add a control module, which includes some control signals to distinguish which calculation function the merged module realizes. According to the difference between these two modules analyzed above, a counter choosing which row of data in the combination matrix to calculate is designed to control this module. In the state of L2P module, it is unnecessary to use a counter for choosing data because the data are just in the second row in the combination matrix. Therefore, the counter starts from zero after a trigger signal being fed to it as the start signal of L2L calculation. Some delays that match the calculating cycles, and required by subtraction and power should be inserted before the counter. That the counter outputs one means data in the first row of the combination matrix would be selected to involve in the calculation. It is easy to deduce the rest operation when the counter increases continuously. The other function of the counter is to detemine which LE coefficient of input to multiply with the corresponding result of the previous multiplier. For example, the product which is obtained from the m-th power of the difference and the data in the combination matrix would multiply the (m+1)-th LE coefficient of input while calculating the first row of the matrix, multiply the (m+2)-th LE coefficient of input while calculating the second row of the matrix and so forth. The strobe signals generated by the counter in the control module decide which LE coefficient of input is selected to multiply, as Fig. 10 shows.

Another control signal needs to be generated to decide whether or not the last add submodule is available. We shall insert a multiplexer to select the data from MAC or Adder before the terminal port. This signal is equivalent to a module-switcher, which can indicate whether the merged module is calculating L2L or L2P. We create this control module whose RTL code is written by ourselves to control some necessary signals mentioned above. The hardware architecture of this merged module, which is built using

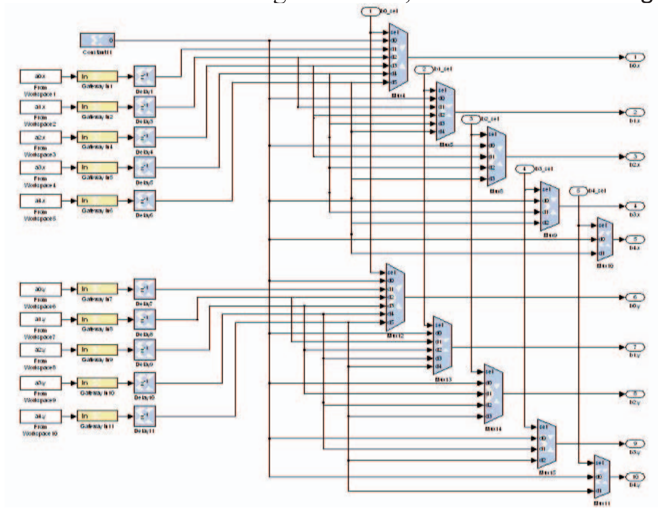

Figure 10. Control circuits of LE coefficients.

Simulink and SysGen as Fig. 9 shows with coefficient p, the order of Taylor Formula, being five. The whole merged module implements L2L calculation and L2P calculation with almost the same resources which the module uses when implementing one calculating function.

V. Hardware Co-simulation and Implementation

We shall accelerate the design simulation process through gererating a hardware co-simulation block after the whole system is created using Simulink and SysGen. Hardware co-simulation is wiring development board and PC together and downloading the hardware design module to the development board for simulation. When finishing setting the reasonable parameters and connecting the JTAG-Based interface line, the whole model will gererate the object files including HDL code and netlists. Meanwhile, a new library will be created and a hardware co-simulation block will be added, which is shown in Fig. 11. During the simulation process, the hardware co-simulation module completes the chip configuration and data transmission through interacting with the FPGA platform. It can be driven by either Xilinx fixed-point data type or by Simulink double-precision data type. In our co-simulation system, input data are generated from Matlab workspace and output data are displayed by scope. When the simulation begins, the input data from MATLAB workspace will be fed to the FPGA by JTAG connection and the results through whole module calculation will be fed back to the MATLAB workspace and shown in scope.

After finishing building hardware system of these modules, we use the Xilinx XST tool to synthesize the whole design and to generate the bitfile. As the PC operating system is 32-bit and the PC memory is 2G in our research environment, which cannot support the synthetical implementation of double-precision type calculation, we generate the input data which is single-precision type through random function. Besides, the computing resources and memory requirements are square proportional to the p value, the order of Taylor Formula. If the p value is too big, the memory requirement would be enormous. So we define the p value as 4 and 5 in our design and implementation in consideration of the PC memory. The development board we used is XUPV5 SX50T board bearing a Virtex-5 FPGA.

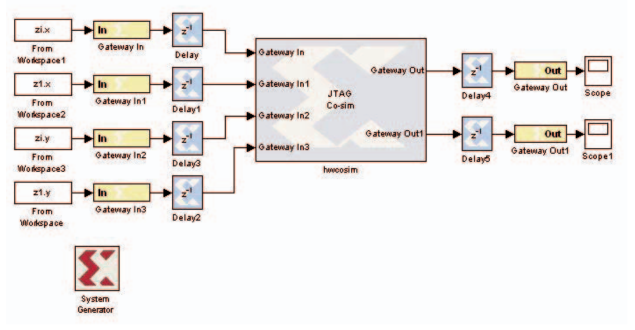

Figure 11. Setting hardware co-simulation

TABLE II. PERFORMANCE AND RESOURCE UTILIZATION COMPARISON

		p = 4		p = 5	
L2L and L2P		*Separated*	*Merged*	*Separated*	*Merged*
FPGA	FPGA chip	xc5vsx50t-1ff1136			
	Maxinum Frequency	160.46 MHz	157.31 MHz	144.26 MHz	133.87 MHz
	Flip-Flops	13978 (43%)	5665 (17%)	18393 (56%)	7433 (22%)
	LUTs	10092 (31%)	3471 (10%)	12828 (39%)	4412 (13%)
	DSP48e	104 (36%)	104 (36%)	144 (50%)	144 (50%)
	Running Time	24.928ns	25.428ns	34.66ns	37.35ns
CPU	CPU Model	Intel Pentium Dual E2200 @2.2GHz			
	Running Time	1.48us		2.1us	

Table II shows the performance and resource utilizations of merged module we designed compared with L2L and L2P module implemented separately. The running time shown in Table II refers to the calculate time that is spent on the calculation process between one father box and one child box. We can know from the Table II that the performance of the merged module has almost been unchanged in compared with L2L and L2P module implemented separately. But the resource utilization of the merged module is greatly reduced. Except for the same in DSP48e, other resource utilization such as FFs and LUTs are almost one-third less than that of L2L and L2P, which is obviously improved. Furthermore, we also compared the performance on FPGA board and on CPU using C code on Visual C platform. The result is shown in Table II as well. When the p value is 5, the running time of the merged module with the software code on CPU is 2.1us, while the running time of the same module with the hardware design on FPGA is 37.35ns. Namely, the speedup ratio is almost 60.Through implementing in hardware, we are able to accelerate the program. This comes from both moving the logic into the hardware and being able to run the calculation in parallel.

VI. CONCLUSION

To evaluate the feasibility of ASIC design for FMM algorithm, we have presented our practice of developing hardware architecture design using Matlab Simulink and Xilinx SysGen by implementing major modules in FMM algorithm on FPGA. The work we have researched presents an evaluation of the technique implementation. The result we obtained is valuable to future work. We explored the feasibility of parallel partitions of calculation process and merger of modules to achieve high resource utilization. We verified our design through running hardware co-simulation on FPGA, whose data were fed by Simulink, and implementing on FPGA XUPV5-SX50T board. Both the simulation result and the output from FPGA board demonstrated the correctness of our design. We endeavored to design a merged module, which has both calculating function of L2L module and L2P module. The merged module brought much better resource utilization while the performance is not compromised. The performance of hardware implementation we obtained boasts a speedup of almost 60 times than an implementation on a dual-core Intel 2.2GHz processor. Further work will include the integration of other calculation modules such as M2M and M2L to make the first complete implementation of FMM algorithm on FPGA in academia. As per our finding and evaluation, we are confident that ASIC design for FMM algorithm is achievable.

ACKNOWLEDGMENT

This research was supported in part by Grant No.2009AA012201 from China 863 program (Hi-tech key project).

REFERENCES

[1] Y.Hagihara, Celestial Mechanics, Vol.I, V, MIT Press, Cambridge, 1970～1976.

[2] Dominique Aubert, Mehdi Amini, Romaric David. A Particle-Mesh Integrator for Galactic Dynamics Powered by GPGPUs. ICCS 2009, Part I, LNCS 5544, pp. 874–883, 2009.

[3] Barnes J, Hut P. A hierarchical O(NlogN) force calculation algorithm. Nature 324: 446–449,1986.

[4] L.Greengard and V.Rokhlin. A fast algorithm for particle simulations. J.Comput. Phys. 73(2): 325-348, 1987.

[5] H. M. P. Couchman, “Mesh-refined P3M: A fast adaptive N-body algorithm,”Astrophys. J. Lett., vol. 368, pp. 23–26, 1991.

[6] Chhatnag Road, Jhunsi, TreePM: A Code for Cosmological N-Body Simulations, J. Astrophys. Astr. (2002) 23, 185–196

[7] J. Makino, K. Hiraki, and M. Inaba. GRAPE-DR:2-Pflops massively-parallel computer with 512-core,512-Gflops processor chips for scientific computing. InSC '07: Proceedings of the 2007 ACM/IEEEconference on Supercomputing, pages 1–11, New York,NY, USA, 2007. ACM.

[8] Tsuyoshi Hamada, Rio Yokota, Keigo Nitadori, Tetsu Narumi, Kenji Yasuoka, Makoto Taiji, 42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence, In SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (2009), pp. 1-12

[9] W.J. Bouknight, S.A. Denenberg, D.E. McIntyre, J.M.Randall, A.H. Sameh and D.L. Slotnick, The Illiac IV system, Proc. IEEE, vol. 60, no. 4, April 1972, pp.369-388

[10] J. L. Potter. The Massively Parallel Processor. The MIT Press, Cambridge, Massachusetts, 1985.

[11] W. D. Hillis. The Connection Machine. MIT Press, Cambridge, Massachusetts, 1985.

[12] Jonathan Phillips, Matthew Areno, Brandon Eames, and Aravind Dasu, An FPGA-Based Dynamic Load-Balancing Processor Architecture for Solving N-body Problems. at the 10th High Performance Embedded Computing (HPEC) workshop, MIT Lincoln Labs. Sept 2006. Page(s): 1

[13] Gerhard Lienhart, Andreas Kugel and Reinhard Männer, Using Floating-Point Arithmetic on FPGAs to Accelerate Scientific N-Body Simulations, In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing machines (FCCM 2002), pages 182–191,April 2002

[14] Felipe A. Cruz, L. A. Barba, Characterization of the errors of the Fast Multipole Method approximation in particle simulations, eprint arXiv:0809.1810 [cs.DS]