

Heterogeneous Reconfigurable Design for TreePM N-body Simulation

Linlin Zheng*, Xi Jin*, Bo Peng*, Tianqi Wang*

*Department of Physics, University of Science and Technology of China, Hefei, Anhui Province, China

linlinoo@mail.ustc.edu.cn, jinxi@ustc.edu.cn, pengball@mail.ustc.edu.cn, tqwang@mail.ustc.edu.cn

Abstract— In this paper, we describe our implementation for N-body simulation with the aid of FPGA, GPU and CPU. TreePM, which is a common algorithm for N-body simulation and meets our demands well, is chosen here. We split treePM algorithm into several parts, and assign them to different accelerators depending on the features of the operation. Moreover, partially dynamically reconfigurable FPGA architecture is used here for fine-grained scheduling, making the assignments more balanced. Experiments showed that it has improvement in speed and energy efficiency over previous design with the same scale. Besides, it's potential for advances with multiple clusters also been demonstrated here.

Keywords—N-body, treePM, heterogeneous, partial configuration, FPGA

I. INTRODUCTION

N-body simulations are fundamental for the research of large scale system. Such system include astrophysical N-body systems, like solar systems, galaxies, clusters of stars and so on. The evolution of N-body systems are determined by the gravitational interactions forces between all the bodies [1].

The most direct algorithm, which is the so called PP (Particle-Particle) algorithm, is to calculate the acceleration of a body by computing the potential forces come from the rest of the bodies in the system. Obviously, the cost of it is $O(N^2)$. It is a heavy burden for computing when N is large. So it is always regarded as a benchmark for high performance computing. In this situation, the memory, the power and the instruction level parallelism wall should be under consideration.

There are many algorithms with approximation have been used for the study, among which tree code is the most widely used algorithm for the simulation. The tree algorithm reduces the cost of calculation from $O(N^2)$ to $O(N \lg N)$. Many Gordon-Bells were given to the projects which had made efforts to accelerate the tree code (1992, 1995-2001, 2003, and 2009-2010)[13]. Some made use of a series of ASIC-based special-purposed computers, named GRAPE (GRAVity Pipe), for the acceleration of N-body system. Others, which is also newer, used GPUs (Graphics Processing Units) instead and outperformed GRAPEs to some extent [4].

Besides, the PM (Particle Mesh) algorithm is also a popular choice for the cosmological N-body simulations. It is the fastest method for computing the gravitational field with

$O(N_g \lg N_g)$ cost. Here N_g is the size of the grid, which is always much smaller than N. However, it is difficult for PM algorithm to reach high spatial resolution [3].

In order to improve the accuracy of PM and the speed of the tree code, treePM algorithm was put forward. The main idea of the algorithm is to split the force into two parts, the short-range and the long-range force. The former is calculated by tree code to improve the spatial resolution, while the latter is calculated by PM to satisfy the need of speed. Some of the GRAPEs also implemented treePM algorithm for the study of N-body system. Besides, the newest Gordon-Bell winner related N-body problem adopted treePM on K computer, a supercomputer with a total of over 640,000 cores, to simulate the evaluation.

In this paper, we present a novel heterogeneous method to implement treePM. FPGA (Field Programmable Gate Array) shows superiority for its customizable and high energy efficiency. Being dynamically reconfigurable makes it more flexible. Most of our tasks are implemented on FPGA. Whatever, GPU is also a preferred accelerator in many cases for its ability of high parallelism. We implement treePM here on a heterogeneous system with FPGA, GPU and CPU. FPGA is mainly applied here, while CPU is used to transmit data and GPU is used to build the dynamic tree data. To verify the idea, we employ Xilinx Kintex-7 and NVIDIA TEGRA K1 to construct the system. Zynq-7020 is also used for comparison.

In the following we introduce the treePM algorithm in detail and review the related work in Section II. Then we present our design for treePM in section III. Section IV gives the results for our design and compares it with related work. At last, section V draws conclusions.

II. BACKGROUND AND RELATED WORK

A. Algorithm

Tree code is presented by Barnes & Hut in 1986 [6]. The basic idea of it is that when some neighbouring bodies are far enough from the target body, they can be seen as a particle in the calculation.

To make the description easily, as Figure 1 showed, we set quad-tree as an example. Square is equally split into four parts recursively when there are more than one body left in the square. The gravity center of each cell is determined by the position and the mass of all the sub cells and set as a tree node.

The node in the tree with no child means that it is a body. The oct-tree is analogous to quad-tree. Secondly, to get the force impacted on a body, a tree is traversed from the root. In each level, if the gravity center is sufficiently far away, the tree is truncated here and the center is added to the list waited to be summed up the interaction.

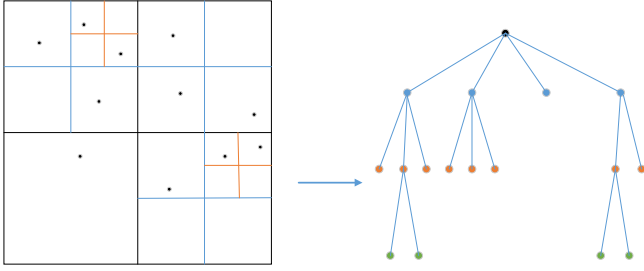


Figure 1. The building of the hierarchical quad-tree. Different colour of the dots in the tree means different level of the tree.

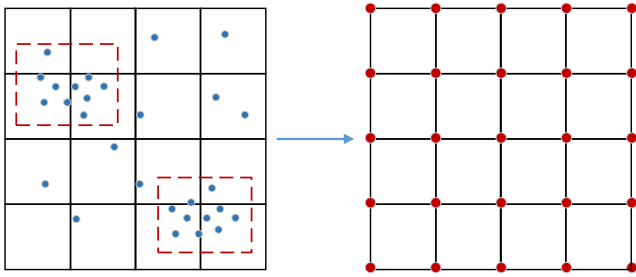


Figure 2. The distribution of all the bodies on the mesh. As it shows above, the bodies cluster strongly in our study. So PM algorithm may have low resolution if the grid size isn't small enough.

PM code is the fastest choice for computing gravitational force. As Figure 2 shows, it puts the calculation on grid. Firstly, the density is distributed to the grid point. Here we used CIC (clouds in cells) algorithm. Secondly, to get potential of the grid point, the Poisson equation for the potential will be solved. The forward FFT of the density is applied, followed by multiplying with the Green function, and then the inverse FFT will be transformed. Finally, the force field is determined and the force on a body position is obtained by differentiating and interpolating the force field on the grid point. Obviously, spatial force resolution is restricted by mesh size [6].

In our study, cosmological simulations cluster strongly and have a very large dynamic range. PM algorithm is not an efficient way to solve our problem. Using tree algorithm will cause lots of empty leaves in the tree, wasting computing resources. So treePM becomes our preferred solution.

$$\begin{aligned}\phi_k &= \phi_k^{long} + \phi_k^{short} \\ \phi_k^{long} &= \phi_k \exp(-k^2 r_s^2) \\ \phi_k^{short} &= \phi_k [1 - \exp(-k^2 r_s^2)]\end{aligned}\quad (1)$$

Here r_s is the scale for the force splitting. And k is the distance between source body and target body. ϕ_k^{long} becomes smaller and decreases to zero when k getting smaller, while ϕ_k^{short} goes to the opposite side [3].

After the gravitational force on all the bodies have been obtained, and the bodies have moved to the new position, the computation for one time step is finished. Then new potential will be calculated.

B. Related work

TreePM has been deeply studied in recent decades. In 2004, treePM was implemented on CPU by Ray using parallel way. But the task of load balance becomes nontrivial and gives rise to the complexity of the computing. In addition, Gadget-2, which is a massively parallel TreeSPH code and also can be configured as treePM code, finds a way to assign the tasks to arbitrary number of processes. It used space-filling Peano-Hilbert curve to decompose the bodies for oct-tree code. And that is instructive for our study. However, Tomoaki ISHIYAMA in 2009 put forward a solution to balance the load for parallel TreePM and achieve better scaling than Gadget-2. But it was also not efficient for repeating some calculation, e.g. FFT calculation [2].

There are also implementations of treePM on GRAPEs, which are ASIC-based accelerators. But obviously, they are expensive and less flexible.

As for the similar work on GPU, most of them focus on tree algorithm. [4] presented an adjusted way to fit the tree in GPU in 2010. But the tree was constructed on CPU, neglecting any kind of acceleration. Besides, it fulfil parallelism by grouping neighbouring particles to share same tree traversal list. It is not a complete efficient classical Barnes Hut tree. In addition, in 2011, [8] described a method to implement the whole classical tree entirely on GPU. But its maximum simulation size is limited by the size of memory on GPU.

So the contributions of this paper are as follows:

- Present a heterogeneous system that trades off the advantages of several accelerators to practically accelerate the calculation of N-body simulation.
- Use dynamically reconfigurable platforms to balance the tasks of PM and tree.
- Create a node that can be extent to become flexible clusters to implement treePM algorithm.

III. SYSTEM ARCHITECTURE

Our accelerator platform designed to be combined with FPGA, CPU and GPU. In this section, we will analyse our implementation of treePM algorithm on the platform.

A. The Assignment of the Tasks

The most computing-intensive part of treePM algorithm is implemented on FPGA. CPU here was used to do some preparation work, transmit the data between GPU and FPGA, and merge the results. Tree is a dynamic data structure which is not suitable for FPGA to implement, while GPU was proved to be efficient in building the tree [8]. But for the potential calculation of the tree algorithm, FPGA performed to be more energy-efficient in our experiments. As for the PM calculation, in a time step, PM will be calculated just one time while tree should be computed much more than that. It is convenient for FPGA to use dynamical configuration to combine the calculation together. Figure 3 shows the calculation procedure.

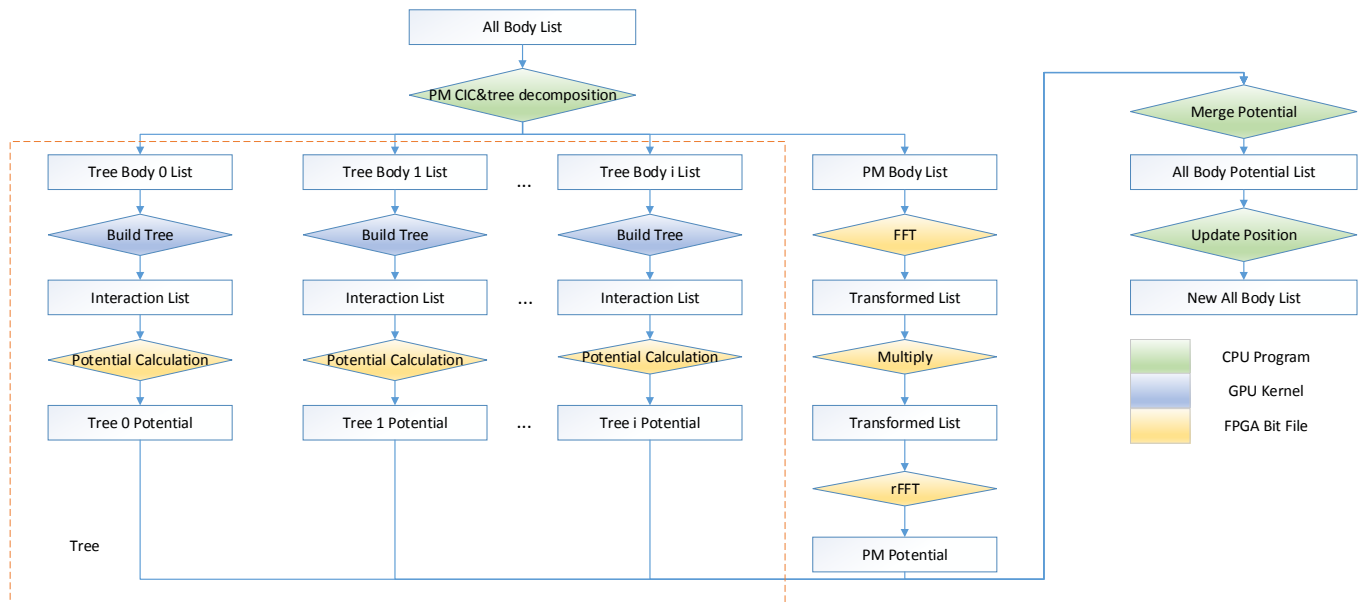


Figure 3. The distribution of the tasks.

B. Algorithm on CPU

CPU was used as a host controller in our design. The calculation procedure of the treePM algorithm in a time step is as follows:

1) The host CPU calculates CIC and divides the neighbouring bodies into groups for tree building. Peano-Hilbert curve is used for the domain decomposition.

2) If the domain decomposition has constructed a new group, the host CPU will send the bodies' data of the group to GPU for the building of the tree when GPU is ready.

3) After GPU has built the tree, the tree structure is sent back to CPU. CPU then constructs an interaction list for the group and sends another group of bodies to GPU. Partial BIT file, which configures the FPGA to calculate purely tree interaction force, or with a fraction of resources for PM, will be chosen according to the estimation made by CPU. Grid data for PM, if PM haven't been calculated yet, and the interaction list, will be sent to FPGA when FPGA is ready. And the partial BIT file, if new one is needed, will be loaded to FPGA, too.

4) Repeat 3 for all groups.

5) CPU merges the results of tree and PM, updating the position and the velocities of all the bodies.

Different partial BIT files determine different percentage of resources used for PM. And we can estimate the need for the calculation by the size of grid and the size of the tree interaction list.

C. Algorithm on FPGA

Figure 4 shows the basic structure of the FPGA design. The potential calculation for the tree and FFT for PM algorithm, which are the most time-consuming parts in treePM, are calculated on FPGA. To split the calculation stage of tree potential for pipeline, it is essential to ensure every stage of

the pipelines has same latency. So the equation of body-to-node potential calculation for the tree is transformed as followed:

$$\phi(r_i) = m_i \sum_{j < k} \frac{m_j (R_{ij}^2 + 1.5\epsilon^2)}{(R_{ij}^2 + \epsilon^2)^{\frac{3}{2}}} \quad (2)$$

$$\phi(r_i) = m_i \sum_{j < k} \frac{m_j}{R_{ij}^2 + \epsilon^2} \sqrt{R_{ij}^2 + \epsilon^2} * \left(\frac{0.5\epsilon^2}{R_{ij}^2 + \epsilon^2} + 1 \right)$$

Here, $R_{ij}^2 = (r_i - r_j)^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2$. And ϵ is a softening parameter constant for adjusting the results at small distance. k is the number of the interaction nodes for target body i . Due to the complexity of different calculation, the addition, subtraction and multiplication units are set to the same latency cycle, so as the division and square root units.

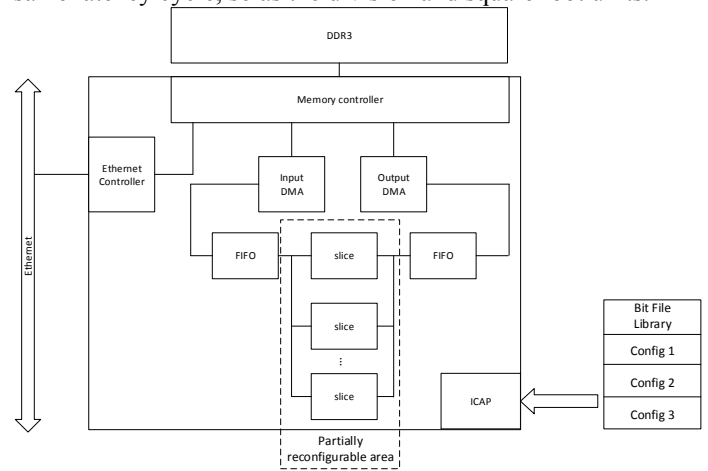


Figure 4. The reconfigurable architecture of FPGA in our design

As for the calculation of PM potential, FFT is the most essential part, and we referred to [7] published in 2014 here. A

3D FFT can be decomposed into 3 sets of 1D FFTs. We used Xilinx LogiCORE FFT v8.0 IP generator, for our FFT pipelines. However, it is essential to find a way to map data onto FFT IP. So crossbars are used here. The design for PM on FPGA is showed in Figure 5. Firstly, FFT transformation and the parameter is computed. Secondly, the needed multiplication for the Green function is done. At last, the inverse FFT will be calculated to get the potential.

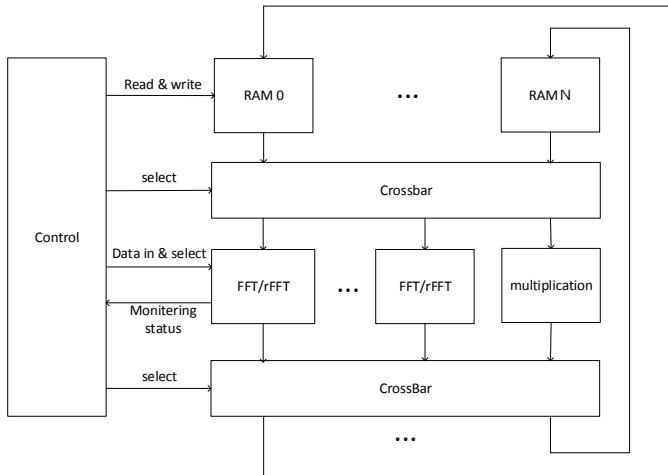


Figure 5. Data flow of the potential calculation in PM

D. Algorithm on GPU

The only job GPU needs to do is building oct-tree, which seems to be much more complex for FPGA because of the dynamic data structure. We mainly referred to [8] here. But since the potential calculation is left to FPGA, the sorting of the body for the acceleration, which seems to be significant in [8], is eliminated here to save more time. So the procedure is as followed:

- 1) Compute the range around all bodies
- 2) Build a tree by inserting each body into octree.
- 3) Summarize tree node information and transmit them to CPU.

IV. EVALUATION

We implemented our design on a combination of NVIDIA TEGRA K1 and Xilinx Kintex-7. NVIDIA TEGRA K1 is consist of NVIDIA 4-Plus-1TM Quad-Core ARM Cortex-A15 "r3" and 192 CUDA cores. It is fine that the GPU and CPU seems to be weak compared to other related work. For the tasks on them are at most, which is much smaller than the tasks on FPGA. Besides, we also implemented tree potential calculation on Zynq-7020 for test.

A. Resource Utilization

We evaluated the resource usage on FPGA here. For most of the time, FPGA purely computed the potential of the tree. Table 1 shows the comparison of pipelines for tree potential calculation.

As shown in Table 1, we compared the pipeline we designed and the pipeline HLS (high level synthesizer) generated with the maximum frequency. And Xilinx Zynq-

7020 is used as test platform for FPGA. And all the results in it is generated by Vivado 2015.2 with default options. We can lead to the conclusion that our design for tree potential pipeline saved resource. So more pipelines can be integrated. Resource utilization comparison of Kintex-7 and Zynq-7020 are showed in Table 2. 32 pipelines can be implemented on Kintex-7.

TABLE 1. ZYNQ-7020 RESOURCE UTILIZATION

entity	LUTs	Registers	DSP48E1
Pipeline(HLS)	7014(13%)	12132(11%)	24(11%)
Pipeline(designed)	4417(8%)	7761(7%)	24(11%)
Accelerator(8 pipelines)	39616(74%)	64777(61%)	196(89%)

TABLE 2. FPGA RESOURCE UTILIZATION

Device	pipelines	LUTs	Registers	DSP48E1
Kintex-7	32	92%	67%	88%
Zynq-7020	8	74%	61%	89%

B. Performance Comparison

It is not instructive to compare our results with large scale super computers or ASIC, like GRAPEs. There is few work we can discover to implement treePM on a heterogeneous system. So we compare our work with that on CPU, like Gadget-2. We run Gadget-2 on 8 cores Intel i7-4940 CPU and 32 cores Intel Xeon E5-2667 CPU. MPI was used here for parallelism. The treePM performance comparison is showed in Table 3. The speedup is not so impressive, for the CPU and GPU here are weak compared to similar products. The impact will reduce as N increases.

TABLE 3. TREEPM PERFORMANCE COMPARISON PER TIMESTEP

	threads	N = 30,000	N = 300,000
Gadget-2(8 cores Intel i7-4940)	1	380.3(ms)	3678.0(ms)
	8	145.4(ms)	2356.9(ms)
Gadget-2(32 cores Intel Xeon E5-2667)	1	476.9(ms)	6136.3(ms)
	8	80.3(ms)	943.1(ms)
Our design	none	35.9(ms)	279.6(ms)

For the results of our solution, it is a case satisfying our demands. The parameters are not exactly the same as Gadget-2. To make our work more convincing, we compared parts of our design with related work implemented on single device. So the summation of the tree potential, which is the most time-consuming calculation in our solution, was compared with similar work in Table 4.

As Table 4 shown, our solution for tree potential is competitive in performance over CPUs. And compared with Intel Xeon E5-2660, the power consuming is about 97W under 2.2GHz, which means that implementation on FPGA

defeat CPU also in performance per Watt. As for GPU, FPGA is proved to be more energy-efficient than GPU in many cases. Set Tegra K80, a high performance GPU we used here as an example. The Power consuming estimated by the frequency is

about 100 Watt, which is about 25 times larger than that of Kintex-7, and about 80 times larger than Zynq-7020. All of these results are similar to the results presented on [5].

TABLE 4. CALCULATING THE DIRECT POTENTIAL WITH DIFFERENT DEVICES

type	Computing unit	parallelism	Freq.(MHz)	M pairs/s	Power(Watt)	Speedup
CPU	ARM Cortex A9	2 cores	667	3.106	NONE	1
CPU	Intel i5-2467m	2 cores	1600	26.97	NONE	12.8
CPU	Intel Xeon E5-2660	8 cores	2200	68.17	NONE	21.9
GPU	Tegra Kepler	192 cores	950	192.0	NONE	61.8
GPU	Tegra K80	2*2496 cores	562	6312	NONE	2032.2
FPGA	Zynq-7020	8 pipelines	200	866.6	1.7	279.0
FPGA	Kintex-7	32 pipelines	344.9	5977.8	4.427	1924.6

V. CONCLUSION AND FUTURE WORK

In this paper, we present a method to calculate treePM, which is a common algorithm for N-body simulation. Not only the speed but also the energy-efficiency of our design is competitive. Using dynamically configurable structure of FPGA, our solution also solve the problem of load balance to some extent, which is a significant problem in treePM calculation [2]. So the loss of performance due to non-optimal load balancing reduces. Besides, it also provides a method to calculate algorithm in a cluster of FPGA, CPU and GPU. With the aid of GPU and CPU, FPGA can take advantage of its strengths better. It is an efficient way for dealing with the wall of power in super computing. For the future work, we may extent the implementation to multiple nodes for the simulation. Penao-Hilbert curve makes the distribution of the trees much easier. And the FFT calculation proved to be convenient for parallelism between multiple FPGAs.

REFERENCES

- [1] Warren, Michael S., and John K. Salmon. "Astrophysical N-body simulations using hierarchical tree data structures." Proceedings of the 1992 ACM/IEEE Conference on Super-computing. IEEE Computer Society Press, 1992.
- [2] Ishiyama, Tomoaki, Toshiyuki Fukushima, and Junichiro Makino. "GreeM: massively parallel TreePM code for large cosmological N-body simulations." Publications of the Astronomical Society of Japan 61.6 (2009): 1319-1330.
- [3] Bagla, Jasjeet S. "TreePM: A code for cosmological N-body simulations." Journal of Astrophysics and Astronomy 23.3-4 (2002): 185-196.
- [4] Hamada, Tsuyoshi, and Keigo Nitadori. "190 tflops astro-physical n-body simulation on a cluster of gpus." Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society, 2010.
- [5] Hamada, Tsuyoshi, et al. "A comparative study on ASIC, FPGAs, GPUs and general purpose processors in the $O(N^2)$ gravitational N-body simulation." Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on. IEEE, 2009.
- [6] Barnes, Josh, and Piet Hut. "A hierarchical $O(N \log N)$ force-calculation algorithm." (1986): 446-449.

- [7] Humphries, Benjamin, et al. "3D FFTs on a Single FPGA." Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on. IEEE, 2014.
- [8] Burtcher, Martin, and Keshav Pingali. "An efficient cuda implementation of the tree-based barnes hut n-body algorithm." GPU computing Gems Emerald edition (2011): 75.



Linlin Zheng received her B.S. degree from University of Science and Technology of China, and she is currently a M.S. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. Her current research work is mainly on FPGA-based Hardware Accelerator design.



Xi Jin received the B.S. degree from University of Science and Technology of China, Anhui, China, and he is currently an associate professor in Department of Physics in University of Science and Technology of China, Anhui, China. His research interests include SOC design technology, VLSI design, computer-aided design methodologies for SoC system integration and FPGA-based Hardware structure design.



Bo Peng received his B.S. degree and Ph.D. from University of Science and Technology of China, and he is currently a lecturer in Southwest University of Science and Technology. His current research work is mainly on FPGA-based Hardware Accelerator design and network.



Tianqi Wang received his B.S. degree from University of Science and Technology of China, and he is currently a Ph.D. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work is mainly on FPGA-based Hardware Accelerator and computing architecture.