

Design Methodology of the Heterogeneous Multi-core Processor With the Combination of Parallelized Multi-core Simulator and Common Register File-Based Instruction Set Extension Architecture

Bingbing Xia¹, Fei Qiao², Huazhong Yang², Hui Wang²

Department of Electronic Engineering, Tsinghua University, Beijing, China

Email: xbb07@mails.tsinghua.edu.cn¹, {qiaofei, yanghz, wangh}@tsinghua.edu.cn²

Abstract—The era of multi-core processor has come with the development of semiconductor technology, and heterogeneous multi-core processor is better than homogeneous one for both performance and power. In such circumstance, an efficient design methodology for such heterogeneous multi-core processor is given in this paper. At the simulator level, parallelized simulator is used to obtain the high simulation speed and conflict detection ability, at the RTL level, the common register file-based instruction set extension architecture is taken to speedup the application in multi-core systems. And simulation results at the RTL-level show that with such design methodology, taking JPEG encoding as a case study, the heterogeneous multi-core designed gains 5.44X speedup than homogeneous one and the energy cost is only 22.9% of the homogeneous one. What's more, the extra hardware logic cost is less than 25% compared with the homogeneous one, taking both the hardware logic cost and the performance into consideration, such methodology is better than popular XTensa for such architecture exploration based on instruction set extension.

Index Terms—heterogeneous multi-core, JPEG encoder, instruction set extension

I. INTRODUCTION

With the continuous scaling of the transistor size in each process technology, integrating billions of transistors onto a single die has become possible. Along with this trend, more and more processors and hardware accelerators can be implemented onto a single die to build a complex SoC. Such advantages are fully taken by the microprocessor design to make high-performance and low-power processors. Because of the limitation for the power cost, one single core can't work at a higher frequency, so the parallelization of the task is made use of to realize the high performance. Meanwhile, since the clock frequency of each single core is reduced, the total power cost is also reduced accordingly since the power cost is proportional to the square of the clock frequency. Therefore, multi-core processor is preferred today for its high-performance and low-power cost. However, not all the applications will benefit from such homogeneous

multi-core processors. According to Amadal's Law shown in equation 1 (here, f means the portion of the program that can't be parallelized, N means the number of cores), it can be drawn that, for particular application, the speedup ratio for these homogeneous multi-core processor depends on the portion that can't be fully parallelized, especially when the core number is very large.

$$Speedup = \frac{1}{f + \frac{1-f}{N}} \quad (1)$$

To overcome this limitation and further improve the performance, heterogeneous multi-core will be applied for particular applications since it will improve the performance of both the portion that can't be parallelized and the portion that can be parallelized. The modified speedup ratio is shown in equation 2, in this equation, a means the speedup ratio for the heterogeneity of the portion that can't be parallelized, b means the speedup ratio for the heterogeneity of the portion that can be parallelized.

$$Speedup = \frac{1}{\frac{f}{a} + \frac{1-f}{N*b}} \quad (2)$$

It can be drawn that heterogeneity exploration can be applied for the further improvement of the particular application. In such circumstance, for high-performance applications such as real-time video processing, heterogeneous multi-core processors are preferred for its high-performance and low-power cost. Typical architecture for such heterogeneous multi-core processors is shown in Figure 1 [1]. It can be seen that such complex chips consist of several different processor cores (such as the dynamically reconfigurable processor and the matrix processor) and many hardware accelerators (such as the video processing unit and the sound processing unit). Such multi-core processor get at the performance of 37.3GOPS/W which is much better than homogeneous multi-core processors [2].

In academic field, many proposals are given for the realization of the heterogeneous multi-core, since the processor architecture consists of two main aspects, one

An Example of Heterogeneous Multi-core Processor

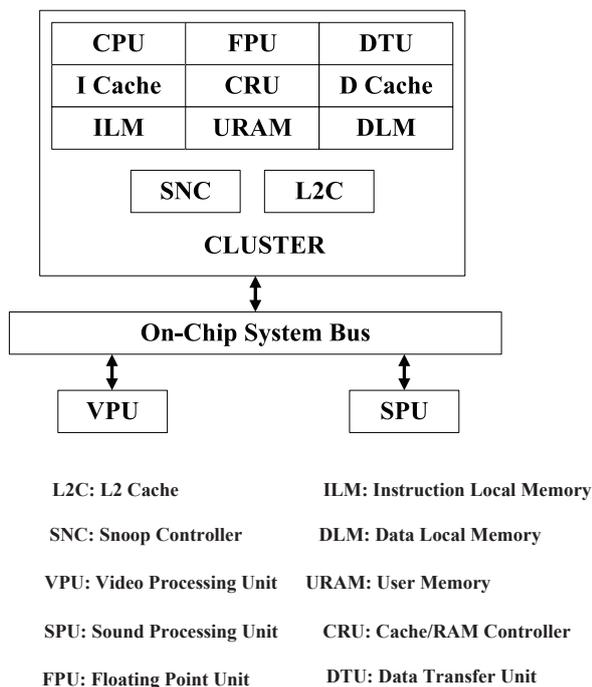


Figure 1. Typical Heterogeneous Multi-core Processor for Multimedia Applications.

is processor architecture, the other is the instruction set. Depending on the difference of these two kinds, heterogeneity is exploited in three main aspects.

The first one is that the processors use the same instruction set, but different architectures, the typical example of such kind is the architecture given in [3]. In this architecture, EV5, EV6 and EV6+ are the cores used, they use the same instruction set, but they are different in issue width, cache size and branch prediction strategy. Simulation results show that such heterogeneity would get the power reduction at about 39% compared with homogeneous multi-core processor. This kind of heterogeneity is easy to integrate and develop since the compilers are the same and the processors are all off-the-shelf, but the performance improvement beyond the homogeneous multi-core is not sufficient enough.

The second kind is that all the processors use various instruction sets and have different architectures. Such as the CELL processor which has PPE and SPE two different processors, nowadays, the popular CPU+GPU architecture also belongs to such architecture [4] [5]. Such architecture can get at about 100X than homogeneous multi-core processor for high data density processing applications, but the development of such applications needs an integrated environment and it's not suitable for application-specific designs.

The third kind of architecture is that the architectures of the cores are similar but the instruction sets are different. Such as the architectures proposed in [6] [7], such architecture usually takes the configurable processor as the processor core, and modify the instruction set based on

this configurable processor and extensible instruction set, they always get a relatively high speedup ratio since such instruction set extension is application-specific. They are easy to develop, but the processor should be configurable and the instruction set should also be extensible. Among these, taking the design complexity into consideration and to obtain a higher speedup ratio, the third type is the most appropriate one for application-specific designs and is easy to use for users, therefore the proposed architecture in this paper belongs to this type.

The proposed design methodology gives a design flow from High-Level(algorithm) to the RTL-level(hardware design), the main contribution of this design methodology is that it integrates the high-level description of algorithm, the cycle-accurate parallelized multi-core simulator and the RTL-level heterogeneous multi-core design using common register file-based instruction set extension architecture into a systematic flow. Taking the JPEG encoder as a case study, the realized heterogeneous multi-core system has much better performance than homogeneous one and achieves lower energy cost as well. Meanwhile, the simulation results show that the performance achieved is better than the similar architectures as shown in [6] [7]. Therefore, it can be used to realize the future application-specific heterogeneous multi-core designs.

The next section gives an overview for this design methodology. Section 3 gives a detailed description of the this design methodology. A case study of JPEG encoding algorithm implementation based on this methodology is given in section 4. At last, conclusion and future work are given in section 5.

II. OVERVIEW OF THE DESIGN METHODOLOGY

Although there are several design methodologies at the high level for such heterogeneous multi-core designs, such as the ones in [6] [7], it's difficult to prove its efficiency without the RTL-level implementation. What's more, although there are also such chips including the reconfigurable processor such as XTensa [8], they lack the high-level simulation which will be used to estimate the total performance before such chip is taped out. The proposed methodology is a top-to-bottom design flow which includes both the high-level simulation and the RTL-level implementation. Three main steps are taken in this flow, including the cycle-approximate transaction-level simulation, cycle-accurate simulator-level modeling and the final RTL-level implementation, it goes from top to bottom to build the final application-specific heterogeneous multi-core processor. The transaction-level simulation is used to estimate the parallelization of the tasks, the cycle-accurate simulator is taken to obtain the detailed running cycles and get the hot-spot functions of the particular algorithm. The RTL-level implementation is the final design of such multi-core chips. In detail, this flow is shown in Figure 2. It mainly consists of 8 steps:

1. Input the algorithm and analyze it for the initial estimate of the task-level parallelization.

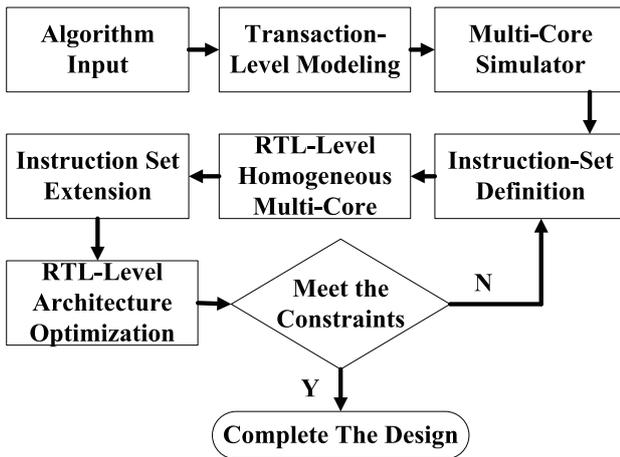


Figure 2. Design Flow for Heterogeneous Multi-core Design.

2. Parallelize the algorithm at the transaction level with the help of SystemC and OpenMP [9] [10].

3. Parallelize the algorithm at the multi-core simulator level.

4. According to the cycle-accurate simulation results obtained in the previous level, extension instructions are defined for the hot-spot functions.

5. Build the homogeneous multi-core at the RTL-level according to the simulation results obtained at step 3. For such design, the processor type and the basic instruction set used are the same with the simulator, MIPS processor and MIPS instruction set are used. The total system for homogeneous multi-core processor is shown in Figure 3. Such multi-core processor consists of AMBA bus, two MIPS processors, data cache, instruction cache and the related memory units which store the instruction and the data. There are private data cache and instruction cache, and the cache coherency is implemented using the directory-based protocol. The cache parameters and the bus configuration parameters are defined in advance.

6. Implement the instruction-set extension structure at the RTL-level to realize the heterogeneity for the particular algorithm. At this step, the heterogeneity of each core is implemented so that each of them matches well with the tasks allocated to it. Based on the algorithm analysis at the transaction level and the hot-spot functions obtained at the simulator level, the hot-spot functions are realized at the RTL-level using the instruction set extension method. After such instruction set extension simulation, the heterogeneous multi-core platform is built and the simulation results will help to make decision on the modification of the selected custom instructions.

7. Optimize the total architecture for the further improvement of the total algorithm.

8. Evaluate the performance of the heterogeneous multi-core, if the heterogeneity really achieves the improvement than the homogeneous multi-core, then complete the total design, otherwise, modify the instructions to be extended until the best heterogeneous multi-core is obtained.

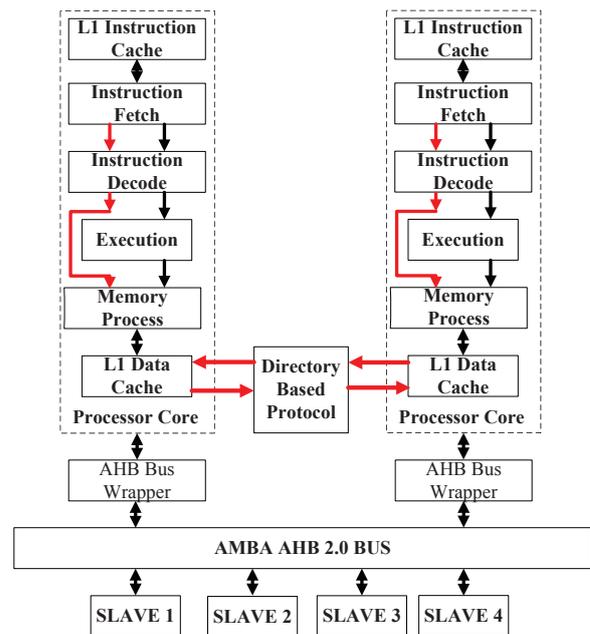


Figure 3. Homogeneous Multi-core Processor.

III. CONTRIBUTIONS OF THE PROPOSED METHODOLOGY

The main contributions of the proposed methodology are shown as follows: cycle-accurate and parallelized multi-core simulation and coarse-grained instruction-set extension architecture.

A. Cycle-accurate and Parallelized Multi-core Simulation

For the realization of such simulation, parallelism is exploited at the two-step simulation.

The first step is at the transaction level. At this step, the parallelization strategy will be estimated at the high level which is not cycle-accurate. For this level simulation, both SystemC and OpenMP are used. SystemC is used for transaction-level multi-core simulation, such transaction-level simulation has both the advantages of object-oriented programming and hardware description ability, it's especially useful for high-level hardware simulation, such as [11] [12]. What's more, to better parallelize the tasks and evaluate whether the task can be really parallelized, OpenMP is also used for such transaction-level design which can help to parallelize the tasks. Based on these tools, the total design flow at this level is shown in Figure 4. As shown in this figure, the algorithm is input and the tasks are parallelized with OpenMP according to the analysis of the algorithm. Afterwards, the simulation of each task is done with SystemC. After such transaction-level simulation, various task-level parallelization strategies of the algorithm are estimated and evaluated.

The second step is at the simulator level. A cycle-accurate multi-core simulator is proposed and used at this step. Such multi-core simulator will supply the cycle-accurate level simulation results for various task allocation strategies. To model the algorithm at the simulator level, a

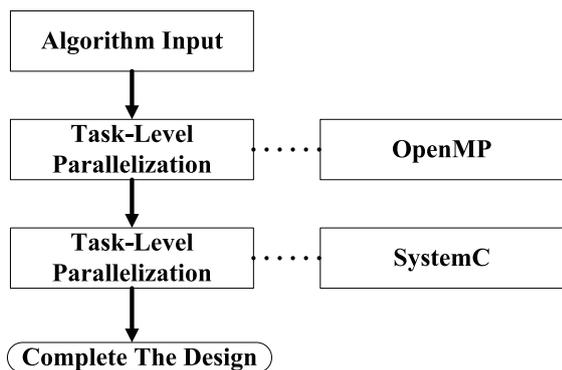


Figure 4. Design Flow at Transaction Level.

TABLE I. COMPARISON BETWEEN SERIALIZED SIMULATION AND PARALLELIZED SIMULATION

Core Number	Simulation Speed			
	1	2	4	8
Serialized Simulation	200KIPS	200KIPS	200KIPS	200KIPS
Parallelized Simulation	200KIPS	200KIPS	207KIPS	316KIPS

multi-core simulator is designed based on the extension of SimpleScalar 3.0. This multi-core simulator is named as Tsinghua Multi-Processor Simulator, thus simulator supports both time and power simulation [13]. The simulation flow for timing is shown in Figure 5 while the simulation flow for power is shown in Figure 6. For the timing simulation, each thread (named as pid N) simulates one core, they run in parallel and the simulation results are gathered together to obtain the performance of the total multi-core processor. In such simulation, the multi-thread running library and the runtime library are both used to realize the 'real parallelized' multi-core simulation. For the power simulation, after the initiation of the multi-core processor, the static power can be obtained. Afterwards, when the simulation starts, the dynamic power is obtained based on the Watch toolset to calculate such dynamic power cost [14]. At last, these two parts of power are gathered together to acquire an estimation of the total power. Simulation results show that there is only 11% difference between such simulator and the RTL-Level implementation. After this level simulation, the hot-spot functions can be obtained which take most of the time.

The best advantage of the parallelized simulator is that it makes full use of the practical multi-core processor to speedup the simulation and get the conflicts that can't be detected by the serialized simulator. The comparison between the serialized simulator and the parallelized simulator is shown in Table I. Based on this table, it can be seen that the proposed parallelized simulation has the higher simulation speed than the serialized simulation especially when the core number is large, and since the tasks run in real parallel, the conflicts between the tasks can be detected through the debugging of the code.

Since transaction-level modeling is popular for dig-

TABLE II. COMPARISON AMONG VARIOUS SIMULATION METHODS

Method	Transaction-level	Simulator-level	Simulation Strategy
[15]	✓		✓
[16]		✓	✓
[17]-[20]		✓	✓
Proposal	✓	✓	✓

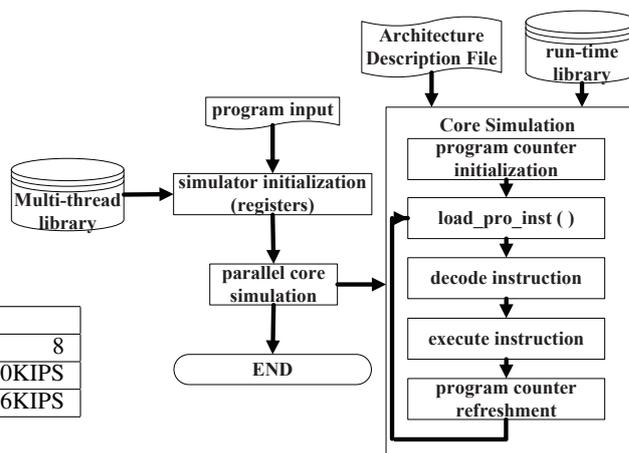


Figure 5. Tsinghua Multiprocessor Simulator Flow Chart.

ital design and SimpleScalar is popular for processor simulation, there are several other similar simulation platforms for such high-level simulation. P. Dziurzanski gives a flow for the conversion from C (with OpenMP) to SystemC [17], it is used to realize the behavioral simulation of C code, but OpenMP is used to obtain the behavioral SystemC. In the proposed methodology, OpenMP is used with SystemC to realize the real parallelization for the multi-core simulation and the conflicts for the common memory input and output can be obtained. What's more, G. Beltrame gave a transaction-level MPSoC simulation platform [15], it used both SystemC and Python to implement the multi-core simulation. It used trap emulator to manage the concurrence and the code can be written with the OpenMP directives, however, the simulation doesn't run in parallel for each core. N. Manjikian gave a modeling and simulation methodology for multi-core processor [18], it used SystemC to build the system and the simulation was trace-driven, which was not appropriate for cycle-accurate simulation such as the execution-driven simulator used in this paper. S. Stattelmann gave a conflict simulation strategy for SystemC based conflict simulation, but it's not appropriate for coarse-grained simulation since the Synchronization Phase and the Scheduling Phase would take a lot of time [19]. Through the combination of SystemC and OpenMP, for coarse-grained multi-core simulation, the proposed simulation has the better performance since the conflicts can be obtained by analyzing the output frames. R. Bergamaschi gave a high-level component-based analysis tool for multi-core system named SLATE (System-Level

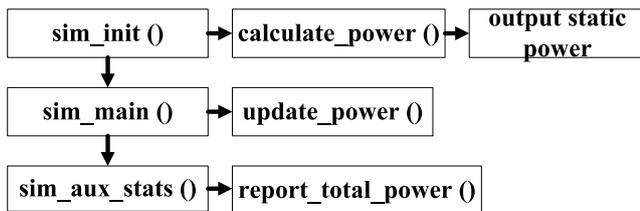


Figure 6. Power Simulation Flow.

Analysis Tool for Early Exploration) [16]. The error rate for such tool is less than 16%, which is acceptable for high-level simulation. However, such simulator is based on the popular components written in SystemC and the simulation is trace-driven, and it can't model the detailed architecture for complex multi-core chips. R. Zhong gave a multi-core simulator based on SimpleScalar, it makes use of SystemC and the shared memory to build such simulator [20]. Although such architecture can model the multi-core processor, it has the following disadvantages: 1, Lack the capability to simulate the instruction-level parallelization. Such architecture for multi-core simulation uses task-level parallelization instead of instruction-level parallelization, so the detailed simulation can't be realized for such simulator, such as cache coherency. 2, Not real parallel simulation. Based on the synchronization scheme in the paper, the simulation record the access cycles of the cores one by one, which will not be appropriate for the system-level simulation since the memory is accessed in parallel instead of one by one. 3, No integrated simulation framework. Based on the analysis of such simulator, the assembler codes for the cores are produced one by one, if there are two cores, two C codes are input to such platform, which is not convenient for parallelized code such as the C code written with the synchronization primitives. Recently, several other multi-core simulators are used for academic research, such as M5, Simics and GEMS, all of them are cycle-accurate, but the simulation for multi-core is serialized [21] [22] [23]. The comparison between the above simulators and the proposed simulator is shown in Table II. It can be seen that our proposed simulator has the advantage of the parallelized simulation over the other ones and is most suitable for the simulation of the real multi-threaded applications.

B. Efficient Coarse-grained Instruction-set Extension Architecture

Such instruction-set extension architecture is shown in Figure 7. In detail, the instructions are dispatched through the dispatcher, afterwards, the general instructions and the extension instructions go through two different data-paths, the general register file stores the data for general instructions and the custom register file stores the data for extension instructions. Data can be transferred between these two register files. The custom logic in such instruction set extension architecture is the particular hardware accelerator for the particular functions which are selected according to the hot-spot functions obtained.

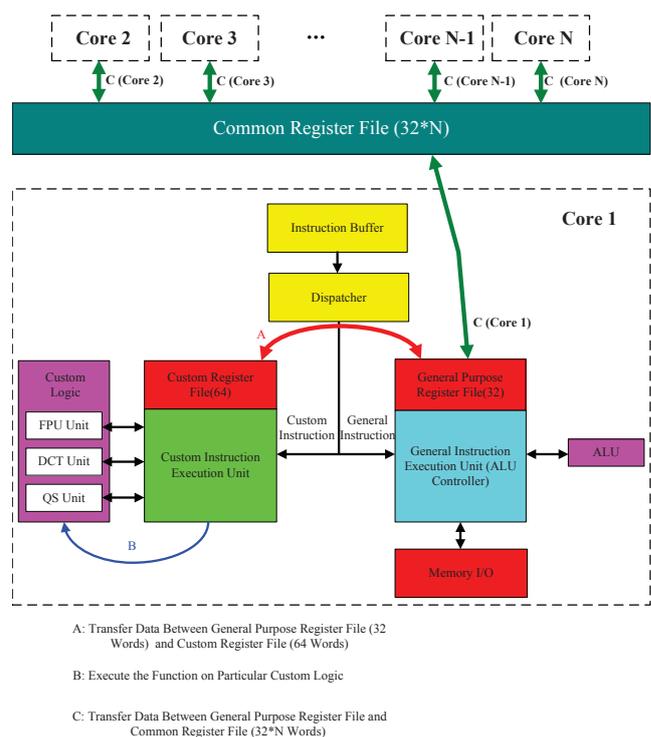


Figure 7. Instruction Set Extension Structure.

This architecture is built based on the MIPS processor. At the instruction dispatcher, the instructions are dispatched to two different instruction execution units according to the type of the instruction, therefore, the 5-stage pipeline is not affected. The advantage of this architecture is that it has the common register file for the data transferring to and from the cores, therefore, such architecture is more suitable for block-level processing applications since no bus transfers are required. For such realization of the coarse-grained instruction set extension method, three types of instructions are used as follows:

Type A instructions: Transfer data between general purpose register file and custom register file.

Type B instructions: Execute the function on particular custom logic.

Type C instructions: Transfer data between common register file and general purpose register file in each core.

In comparison, the instruction set extension architecture taken in XTensa is shown in Figure 8 [8], the structure consists of several parts, including the base ISA architecture, the configurable function, the optional function and the extensible structure. Compared with the proposed instruction set extension architecture, XTensa has no block-level data transferring structure such as the common register file, so the instruction set extension efficiency of the proposed architecture is higher than XTensa. The JPEG Encoder case shows that the efficiency of the proposed architecture is about 27% higher than XTensa.

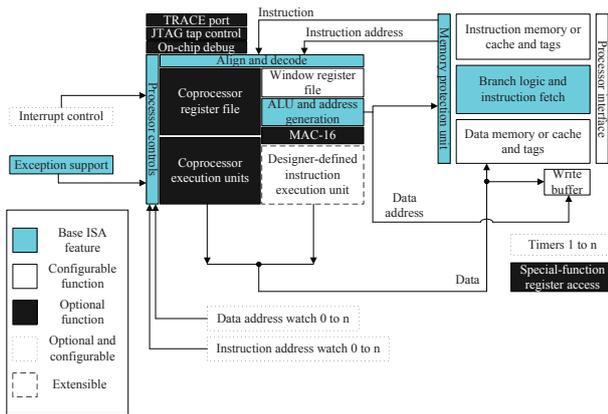


Figure 8. XTensa Architecture.

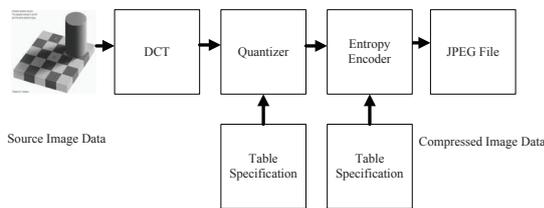


Figure 9. JPEG encoding algorithm.

IV. CASE STUDY : JPEG ENCODER BASED ON THE PROPOSED METHODOLOGY

To exploit the advantages and disadvantages of such design flow, the case study of JPEG encoder is given in this section on the proposed design flow as illustrated in Figure 2.

JPEG encoding algorithm is popular for image compression [24], it's widely used in multimedia applications, the algorithm consists of the following parts: discrete cosine transform(DCT), zigzag scan, quantizer and entropy encoder, just as shown in Figure 9.

A. Transaction-Level Simulation

At this level, according to the algorithm and based on the simulation using OpenMP and SystemC, the 4 major tasks can be parallelized so that they can run in parallel, as shown in Figure 10.

Based on this diagram, it can be seen that, apart from header writing, tail writing and entropy encoding, other parts can be parallelized. According to the analysis of this encoding procedure, the pseudo code for the functions that can be parallelized are shown as follows :

```

Pseudo Code for Transaction-Level Simulation
#pragma omp parallel sections
#parallel section 1
    Task_Core1:DCT/Zigzag
    Time Caculate()
    Wait()
#parallel section 2
    Task_Core2:DCT/Zigzag
    Time Calculate();
    Wait();
    
```

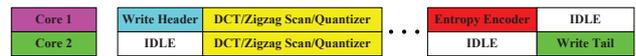


Figure 10. Task Allocation at Transaction Level.

TABLE III. HOT-SPOT FUNCTIONS FOR JPEG ENCODING ALGORITHM

Function Name	Cycle Percentage
dct	40%
ac_encode	39%
quant/zigzag	7%
dc_encode	6%
code_conversion	2%
put_header	1%
put_tail	1%

B. Simulator-Level Simulation

At this level, the cycle-accurate simulator is used. According to the task-allocation shown in Fig. 10. The DCT and Zigzag functions are chosen as the tasks to be parallelized. Therefore, the pseudo code for the total algorithm is shown as follows:

```

Pseudo Code for Simulator-Level Simulation
Serial Part1
    Put_Header()
Parallel Function
    Get Thread_ID
    Get Relative Data according to the Thread_ID
    TASK Execution:DCT/Zigzag
    Barrier(2 Threads)
Serial Part2
    Entropy Encode()
    Put_Tail()
    
```

Based on simulation the percentage of the cycle cost for each function of JPEG encoding algorithm is shown in Table III. As shown in this Table, dct,ac_encode and quant/zigzag are the three hot-spot functions for such JPEG encoding algorithm.

C. RTL-Level Simulation

According to the homogeneous multi-core architecture shown in Figure 3, the tasks are allocated onto these two cores according to the Figure 10. Based on the simulation results at this level, since cycle cost of the ac_encode function takes only about 7% and the quant/zigzag function takes about 10% of the total algorithm, therefore, DCT and quant/zigzag are chosen as the custom modules to realize the instruction set extension. After this definition of these extension instructions, the architecture of the whole heterogeneous multi-core processor is shown in Figure 11. In this heterogeneous multi-core architecture, the DCT and quant/zigzag functions are realized based on the instruction-set extension structure as proposed in Figure 7. Since the instruction-set is extended, the task allocation strategy is modified accordingly as shown in Figure 12. As shown in this figure, the block-level processing of DCT and quant/zigzag run in pipelined fashion, core 1 takes responsibility for DCT function and the entropy encoding, and the other functions are

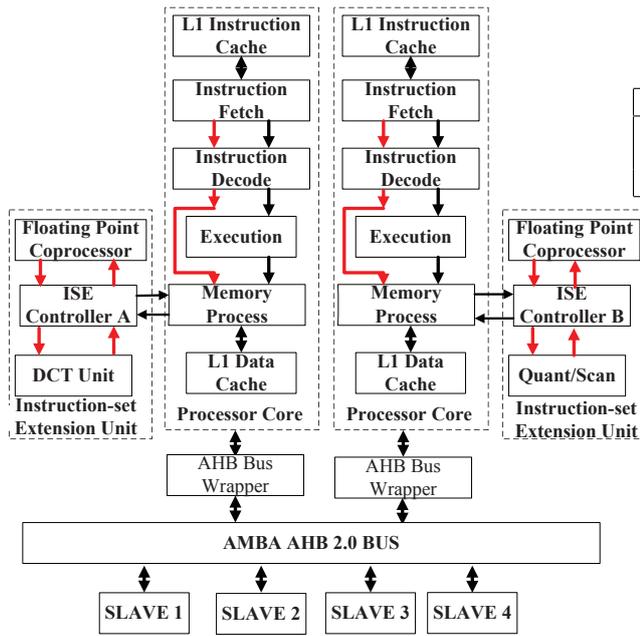


Figure 11. Heterogeneous Multi-core Architecture.



Figure 12. Task Reallocation for Heterogeneous Multi-core Processor.

implemented on core 2, in this way both the cores make full use of the instruction-set extension structures.

D. Experimental Results

Based on such design methodology, the heterogeneous multi-core processor is built. The simulation results based on the DC compiler are obtained, three kinds of comparisons are made in this part to show the efficiency of the design methodology. The first one is the comparison between the homogeneous multi-core processor and the heterogeneous multi-core processor which shows the effectiveness of this methodology, the second one is the comparison for the deviation between different levels of the methodology which shows the accuracy of it. The last one is the comparison between the Instruction Set Architecture and the popular ones (such as XTensa) which shows the efficiency for our selected instruction-set extension architecture.

1) *Architecture Comparison:* The comparisons between the single core, homogeneous multi-core and heterogeneous multi-core for speed, power and energy are shown in Table IV. For the implementation of instruction-set extension, the time overhead for the insertion of the instruction set extension architecture can be overlooked. Taking JPEG encoder as an example, the integration of the DCT module and Quant/Scan module takes less than 5 hours. The time for the selection and insertion of the instruction for extension is much less since the time cost

TABLE V. COMPARISONS BETWEEN DIFFERENT ARCHITECTURES(HARDWARE LOGIC COST)

Architecture	Hardware Logic Cost	Normalized
Single-Core	1,119,793	1
Homogeneous Multi-core	2,326,755	2.0778
Heterogeneous Multi-core	2,886,820	2.5780

of the simulator is less than 1 minute. As shown in this table, Cycle Cost means the cycle cost for the JPEG encoding algorithm at RTL-level. It can be seen that the speedup ratio of the homogeneous multi-core over the single-core is 1.55, and the speedup ratio of the heterogeneous multi-core over the single core is 8.45. Power cost means the total power cost of the architecture(Unit:mW), since the instruction set extension is used, the power cost of the heterogeneous multi-core is somewhat larger than homogeneous multi-core. Energy cost is the product of power and running time, and the major advantage of such heterogeneous multi-core processor is its efficiency for the energy cost. However, since the running time is much reduced the energy cost for JPEG encoding at heterogeneous multi-core has the best performance among these, as shown in this Table, the energy cost of heterogeneous multi-core for JPEG encoding only takes about 27% than the single core. From this table, it also can be seen that the speed of proposed heterogeneous multi-core architecture is 5.44X than the homogeneous one, and the energy cost of such heterogeneous multi-core is only 22.88% of the homogeneous one. The comparison for hardware logic cost is shown in Table V. The unit for the hardware logic is the gate, it can be seen that with the instruction set extension unit and the modification on the architecture for the reconfigurability, the hardware logic cost for the heterogeneous multi-core is about 1.25X than the homogenous multi-core processor. Taking that the time cost for Huffman encoding (by software) and writing head and tail for the JPEG file takes about 12% of the total algorithm, such speedup ratio of 8.45 versus the single core achieved by heterogeneous multi-core processor is high enough for this application.(According to Amadahl’s Law, the highest speedup ratio that can be achieved by homogeneous multi-core processor is less than 8.33. Through the optimization of the architecture, the heterogeneous multi-core processor also improves such serial part)

2) *Simulation Accuracy Comparison:* According to this methodology, three levels of simulation are realized. The accuracy comparison among these levels is shown in Table VI. It can be seen that although the code sizes for JPEG encoding algorithm are similar on these platforms (C Code or the Assembler Code), the simulation time for transaction-level is the least among the three and the RTL-level is the highest among these. So system-level parallelization and simulation is fast to simulate the total design to reduce the modification time cost at the RTL level. As shown in Table VI, a modification

TABLE IV.
COMPARISONS BETWEEN DIFFERENT ARCHITECTURES(SPEED/POWER/ENERGY)

Architecture	Speed		Power(Unit:mW)		Energy	
	Cycle Cost	Speedup	Power Cost	Normalized	Energy	Normalized
Single-Core	48,210,150	1	500.8	1	0.2414	1
Homogeneous Multi-core	31,000,000	1.55	919.5	1.84	0.2850	1.18
Heterogeneous Multi-core	5,700,000	8.45	1144.3	2.28	0.0652	0.27

at RTL level will introduce a 2-4h simulation time cost, while for transaction level, this is only 0.015s. Therefore, using the transaction-level simulation and the simulator-level simulation is really a need for such high-complexity designs. Based on this table, it also can be seen that along with this design methodology, the inaccuracy goes near to zero. The accuracy of this methodology is enough for such designs.

3) Instruction-Set Extension Strategy Comparison:

Since the heterogeneity is realized based on the configurable processor and the instruction set extension, the comparisons with other popular instruction set extension methods are shown in Table VII. The instruction set extension methodologies are divided into two kinds: one is coarse-grained instruction set extension architecture, such as XTensa [8], the other one is the fine-grained instruction set extension architecture, such as ASIG [25]. For the first type, the speedup ratio is always high while the hardware overhead is always not small. While for the other type, the speedup ratio is always not high, but it needs much less hardware logic resource. According to this table, since the ASIG is fine-grained instruction set extension, it can't achieve a high speedup ratio as the other two, and in comparison with XTensa our proposed architecture has better performance (we get the performance improvement at 1.2545 Per 1%Extra Hardware Logic, and the number for XTensa is 1.2 Per 1%Extra Hardware Logic). The disadvantages of the proposed architecture is that it has not been a product like XTensa, so the integrated design and develop environment has not be built thoroughly yet.

V. CONCLUSIONS

A design methodology for heterogeneous multi-core processor with instruction set extension architecture is proposed in this paper. Through the transaction-level modeling, simulator-level simulation and RTL-level design, the final heterogeneous multi-core is built. The RTL-level simulation results show that with such design methodology, take JPEG encoding as a case study, the heterogeneous multi-core designed has 5.44X speedup than homogeneous one and the energy cost is only 22.9% of the homogeneous one. What's more, the extra hardware logic cost is less than 25% compared with the homogeneous one, taking both the hardware logic cost and the performance into consideration, such methodology is better than popular XTensa for such architecture exploration. In the future, the total design methodology will be improved and the integrated design and development environment will be built to better design such heterogeneous multi-core processors.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China under Grant #60871005; in part by Ph.D. Programs Foundation (Young Scholar) of Ministry of Education of China under Grant #200800031073; and Research Platform of Graduate Course in Tsinghua University, Structural Integrated Circuit Design (70230183).

REFERENCES

- [1] Y. Yuyama, M. Ito, Y. Kiyoshige, Y. Nitta, S. Matsui, O. Nishii, A. Hasegawa, M. Ishikawa, T. Yamada, J. Miyakoshi, *et al.*, "A 45nm 37.3 gops/w heterogeneous multi-core soc," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*. IEEE, pp. 100–101.
- [2] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*. IEEE, 2008, pp. 88–598.
- [3] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," 2003.
- [4] D. Pham, S. Asano, M. Bolliger, M. Day, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, *et al.*, "The design and implementation of a first-generation cell processor," in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*. Ieee, 2005, pp. 184–592.
- [5] S. Mohanty, "Gpu-cpu multi-core for real-time signal processing," in *Consumer Electronics, 2009. ICCE'09. Digest of Technical Papers International Conference on*. IEEE, 2009, pp. 1–2.
- [6] F. Sun, S. Ravi, A. Raghunathan, and N. Jha, "Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors," 2005.
- [7] H. Javaid, A. Ignjatovic, and S. Parameswaran, "Rapid design space exploration of application specific heterogeneous pipelined multiprocessor systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 11, pp. 1777–1789, 2010.
- [8] R. Gonzalez, "Xtensa: A configurable and extensible processor," *Micro, IEEE*, vol. 20, no. 2, pp. 60–70, 2000.
- [9] F. Ghenassia, *Transaction-level modeling with Systemc: TLM concepts and applications for embedded systems*. Springer Verlag, 2005.
- [10] R. Chandra, *Parallel programming in OpenMP*. Morgan Kaufmann, 2001.
- [11] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "Mparm: Exploring the multi-processor soc design space with systemc," *The Journal of VLSI Signal Processing*, vol. 41, no. 2, pp. 169–182, 2005.

TABLE VI.
COMPARISONS BETWEEN DIFFERENT LEVELS OF SIMULATION

Simulation Level	Simulation Time	Code Size	Speedup Ratio	Inaccuracy
Transaction-Level	0.015s	1113lines	1.17	24.5%
Simulator-Level	2.1s 2.4s	963lines	1.37	11.6%
RTL-Level	2-4h	655+224lines(core1+core2)	1.55	0

TABLE VII.
COMPARISONS BETWEEN DIFFERENT INSTRUCTION SET EXTENSION METHODS.

ISE Architecture	Hardware Logic Overhead	Speedup Ratio	Advantages	Disadvantages
XTensa [8] [6] [7]	30%	6X(JPEG Encoding)	Coarse-grained ISE	Industrial Product
ASIG [25]	2.54%	2.75X	Fine-grained ISE	Speedup Ratio is Limited
Proposed Architecture	22%	5.6X(JPEG Encoding)	Coarse-grained ISE More Efficient Than XTensa	No Integrated Design Platform

- [12] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich, "A systemc-based design methodology for digital signal processing systems," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, pp. 15–15, 2007.
- [13] Accessed at <http://nics.ee.tsinghua.edu.cn/people/qiaofei/Research/TMP.SIM.rar>.
- [14] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2. ACM, 2000, pp. 83–94.
- [15] G. Beltrame, L. Fossati, and D. Sciuto, "Resp: a nonintrusive transaction-level reflective mp soc simulation platform for design space exploration," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 12, pp. 1857–1869, 2009.
- [16] R. Bergamaschi, I. Nair, G. Dittmann, H. Patel, G. Janssen, N. Dhanwada, A. Buyuktosunoglu, E. Acar, G. Nam, G. Han, et al., "Performance modeling for early analysis of multi-core systems," in *Hardware/Software Code-sign and System Synthesis (CODES+ ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*. IEEE, 2007, pp. 209–214.
- [17] P. Dziurzynski, W. Bielecki, K. Trifunovic, and M. Kleszczonok, "A system for transforming an ansi c code with openmp directives into a systemc description," in *Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE*. IEEE, 2006, pp. 151–152.
- [18] N. Ma, N. Manjikian, and S. Sudharsanan, "Modeling and simulation of multi-core multi-threaded processor architectures in systemc," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*. IEEE, 2008, pp. 001 155–001 160.
- [19] S. Stattelmann, O. Bringmann, and W. Rosenstiel, "Fast and accurate resource conflict simulation for performance analysis of multi-core systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.
- [20] R. Zhong, Y. Zhu, W. Chen, M. Lin, and W. Wong, "An inter-core communication enabled multi-core simulator based on simplescalar," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 1. IEEE, 2007, pp. 758–763.
- [21] S. Reinhardt, N. Binkert, A. Saidi, and R. Lim, "Using the m5 simulator," in *ISCA tutorials and workshops, Jun, 2005*.
- [22] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.
- [23] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.
- [24] W. Pennebaker and J. Mitchell, *JPEG still image data compression standard*. Kluwer academic publishers, 1993.
- [25] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific instruction generation for configurable processor architectures," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. ACM, 2004, pp. 183–189.
- Bingbing Xia** was born in Laoling, Shandong Province, China. He is now pursuing his Ph.D degree in Tsinghua University. His research focuses on heterogeneous multi-core system design for video processing. He has published several papers regarding to his project, which have been accepted by ISQED (International Symposium on Qualified Electronic Design), and ICCE (International Conference on Consumer Electronics).
- Fei Qiao** is currently a faculty member of NICS (Nanoscale Integrated Circuits and Systems) Laboratory, Tsinghua University, Beijing, China. He received his B.S. degree from Lanzhou University in July, 2000, then furthered his study in Dept. of Electronic Engineering, Tsinghua University and obtained his Ph.D. degree in July, 2006.
- Huazhong Yang** was born in Ziyang, Sichuan Province, P.R.China on Aug. 18, 1967. He received B.S. degree in microelectronics in 1989, M.S. and Ph.D. degree in electronic engineering in 1993 and 1998, respectively, all from Tsinghua University, Beijing. In 1993, he joined the Department of Electronic Engineering, Tsinghua University, Beijing, where he has been a Full Professor since 1998.
- Hui Wang** was born in Chongqing, China, in 1947. She received the B.S. degree in radio technology from the Radio Department, Tsinghua University, Beijing, in 1970. In 1970, she joined the Department of Electronic Engineering, Tsinghua University, where she is now a professor. She was the vice Chairman of the Department of Electronic Engineering, Tsinghua University, from 1996 to 1998.