The Design of a Hardware Thread Manager for a Polymorphic Multimedia Processor

Bowen Qian[1] , Tao Li[1] and Ting Yang[2]

Abstract This paper proposes a hardware thread manager for the polymorphic parallel processor. The thread manager supports the MIMD mode with 8 threads and SIMD mode with multiple threads using the SIMD controllers in a unified approach to manage two operating modes to achieve a mixture of three types of parallel computation. Thread manager monitors the progress of each thread, the activity of near neighbor shared memory and the status of the router. It schedules the execution slots for the threads. It can start and stop a thread, put a thread on wait, resume the execution of a thread. Thread manager can also record the working status of each thread, while avoiding the waiting problem caused by data availability. This manager is able to maximize the efficiency of a processing element in a polymorphic array processor.

Keywords Many core • Multithreading • Array processor • Parallel processing • Graphics and Multimedia

1 Introduction

In the past few decades, with the rapid development of integrated circuit technology, the design flow has all been improved. During the same period, computer architecture has evolved from simple uni-processor, to superscalar and VLIW, and to multi-core/many-core parallel computers. During the uni-processor

[1] B. Qian (✉), T. Li
School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an, China

e-mail: bymoney@126.com

[2] T. Yang
School of Computer, Xi'an University of Posts and Telecommunications, Xi'an, China

and superscalar era, the main drive of technology was to increase the CPU frequency in order to improve CPU performance. This in turn drives up the power consumption, which then becomes the bottleneck of the architecture. To overcome the problem of the power wall, multi-core and many-core processors [1, 2] enter the market. Parallel processing is used instead of increased CPU operating frequency. However, the power consumption problem does not go away, especially when the number of cores is large and processor utilization/efficiency is low. One approach to improve the efficiency and utilization of the multi-core computers is to use hardware accelerators to manage multi-threaded applications and to speed up context switching [3]. This is the focus of this paper.

This paper proposes a hardware thread manager for the polymorphic multimedia processor PAAG [4]. PAAG is able to seamlessly integrate three forms of parallel computation, the data level parallelism (DLP), the thread level parallelism (TLP), and the instruction level parallelism (ILP). The proposed hardware thread manager helps to improve the computational efficiency of PAAG.

2 The Polymorphic Multimedia Processor

PAAG is a polymorphous array processor designed for graphics and multimedia applications. This architecture supports both SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Stream Multiple Data Stream) parallel computational modes.

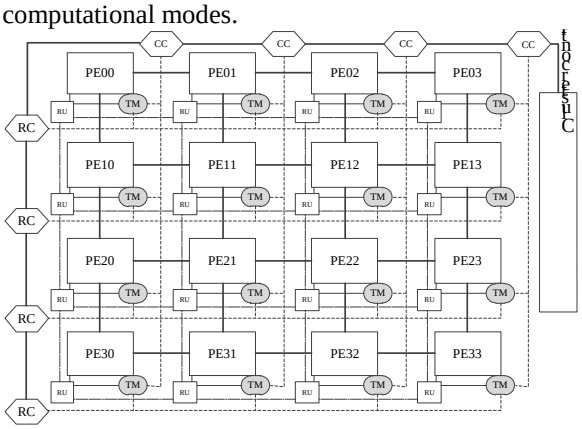

Fig.1 Polymorphic parallel processor

PAAG consists of many clusters of processing elements (PEs). A 4x4 cluster is shown in Fig. 1. The thread manager is a key component of PAAG. Each PE contains up, down, left and right four shared memories for inter-PE communications [4, 5]. A router (RU) is attached to each PE for remote messaging and remote calls. A cluster has 4 row controllers, 4 column controllers and a

cluster controller. These controllers allow SIMD operations on a row or on the entire cluster.

Each PE in the polymorphic multimedia processor contains an ALU, four contiguous shared memories, a data memory and an instruction memory as shown in Fig. 2. It is able to launch two instructions in a cycle. There are two kinds of modes SC load instruction and data: SIMD mode and MIMD mode.

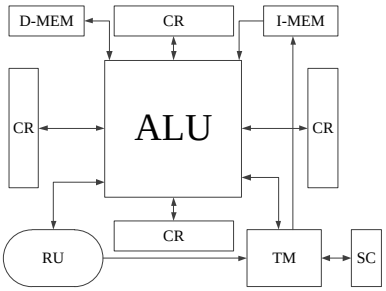

Fig.2 Thread manager & single PE

TM (the thread manager) in Fig.2 is a dedicated hardware accelerator, and is designed to improve the efficiency of the PE, especially when operating in instruction level parallel mode.

3 Thread Manager Workflow

In our polymorphous array processor, an instruction may execute in either blocking mode or non-blocking mode. While executing in blocking mode, an instruction cannot proceed to the pipeline if its operands are not available. In such a case, the program has to wait until operands become available. When multiple threads are in execution, a blocked instruction causes a thread to be put on wait so that other threads may proceed. Fast context switching is needed for efficient execution of multi-threaded programs. Furthermore, each thread is also assigned a quantum of execution. When its running time exceeds the quantum, it is also put on wait. There are some other circumstances that may cause context switching.

In this paper, the hardware thread manager solves the problem of computational efficiency for the polymorphic multimedia processor. Each PE may have up to 8 SIMD threads and 8 MIMD threads.

- The cluster controller and the column controllers help to load instruction and data to the PEs. The initial state of the threads is also loaded.
- Then, the thread manager schedules thread execution according to thread status. The chosen thread will start executing its program.
- If the operands of a blocking instruction are not all available, the thread manager will put the thread on wait. Other conditions may also block an instruction and the thread needs be put on wait.

- The thread manager monitors the shared memories for data availability, the router status and its own thread status table for changes that may trigger status change. Should that change occur, the thread manager reschedules the threads for execution.

4 Thread Manager Functional Description

The thread manager needs to monitor the execution status of the eight threads in a PE. The TM needs to perform context switching in response to the status changes. If a thread is blocked, the thread enters a wait state. If a thread's required data arrives, the thread is ready for execution again.

4.1 Thread register

Thread registers are used to store the working status of the threads and configuration information. The thread manager schedules the threads based on their ranks. There are two groups of registers, the thread configuration table register (*thread_configure_reg*) and the thread status table register (*thread_state_reg*).

Each entry in *thread_configure_reg* contains a total of 58 bits, including five fields as shown in Table 1. The meaning of these five fields as follows.

- quant: The maximum execution time of a thread in each scheduled slot
- I-base: A thread instruction memory base address;
- I-size: The instruction memory size assigned to threads;
- M-base: A thread data memory base address;
- M-size: The data memory size assigned to threads;

Table 1 Thread configuration table register

fields	quant	I-base	I-size	M-base	M-size
bits	10	14	10	14	10

Each entry in *thread_state_reg* contains a total of 38 bits, including six fields, as shown in Table 2. The meanings of these six fields are as follows.

- PC: A thread's program counter value
- state: Current state of a thread;
- avail: Two source operands and a destination operand exists or not;
- mask: Source operand and the destination operand is being used currently executing instruction;
- rank: Scheduling priority, 0 means highest priority;

- stamp: Quant within the thread running time;

Table 2 Thread status table register

fields	PC	state	avail	mask	rank	stamp
bits	10	2	6	6	4	10

4.2 Data exchange instruction processing

Data exchange instructions can be issued by the PE or the RU. A PE may issue the following data exchange instructions.

- MOVEF: This is a request for data from a remote PE. A request message is sent to the remote PE through the router;
- MOVET: This sends data to remote PE. The data is sent out by the router. After the data is sent, it continues to the next instruction;
- CALLR: This is a remote function call. A call message is sent through the router and the current thread suspends execution until a RETR message arrives;
- RETR: This is a return from a remote function call. A message is sent by the remote PE to the calling PE;
- MVT: This is a write operation to cluster memory. The data is sent to the memory bank in a controller. The current thread is suspended until an ACK signal comes back. It may then resume execution;
- MVF: This is a read operation to request data from cluster memory. A request message is sent via the router to a controller and the thread suspends until the requested data come back.
- CALLC: This is a SIMD operation. 8 SIMD threads is started after this instruction. These threads is unified manage by SC.

4.3 Thread scheduling

This design of thread scheduling algorithms in a single module design (*rank_ctrl* module). It is easy to modify. Scheduling algorithm currently used as follows [6].

- Each execution thread only runs the top rank in the sequence;
- Stamp value is equal to quant value, the thread stops executing, and switch to rank last one in the sequence;
- Encounter neighboring communication is blocked, the thread stops executing, and switch to rank last one in the sequence;
- Encounter routing communication is blocked, the thread stops executing;

- When blocking data arrives, the executing thread stops executing, rank value plus 1. Wake thread rank value becomes 0, means that rank in the top of sequence.

Example of 0, 1, 2 and 6 four threads as follows:

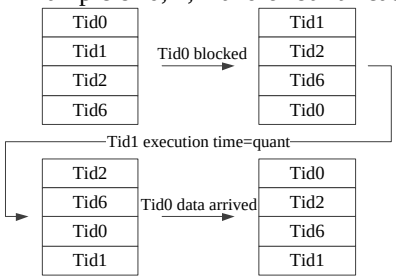

Fig.3 Thread scheduling

5 Thread Manager Design

According to the thread management functions, the TM can be divided into the following five modules:

- *tm_ctrl*: Thread manager control module for controlling SIMD mode, RU operation, PE control and arbitration;
- *regfile*: Register modules, including *thread_configure_reg* and *thread_state_reg* two registers;
- *stamp_ctrl*: Controlling the stamp and the timeout signal;
- *rank_ctrl*: Control the rank value in *thread_state_reg*;
- *state_ctrl*: Control the state value in *thread_state_reg*.

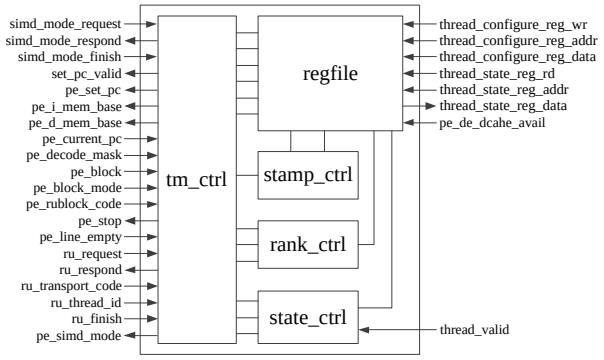

Fig.4 Thread Manager Structure Diagram

The *state_ctrl* module uses the state machine controlling. There are eight state machines. Each state machine individually controls the corresponding thread. These eight state machines are the same. And the states of each thread are

recorded in the thread state table. Each state machine contains four states. As follows:

- idle: Idle state, which means that the thread needs no instruction execution. After the thread loads instruction and data, i.e. *thread_valid* = 1'b1, the thread jumps to ready status;
- ready: Ready state, which means that the thread already has the executable condition. The PE doesn't execute the thread, probably due to PE executing other threads or other reasons. When the thread is selected execution, i.e. *thread_hit* = 1'b1, the thread jump to run state;
- run: Run state, which means that the thread is running. When the thread running time stamp equal to the distribution of the execution time quant or when forced to stop running, the thread jumps to ready state. After the thread finish running, the thread jump to the idle state. When blocking the thread, the thread jumps to wait state;
- wait: Wait state, which means that the thread is waiting for blocking data to arrive. When the router or neighborhood communication blocking data arrives, the thread will jump into the ready state, waiting for the thread to be selected execution.

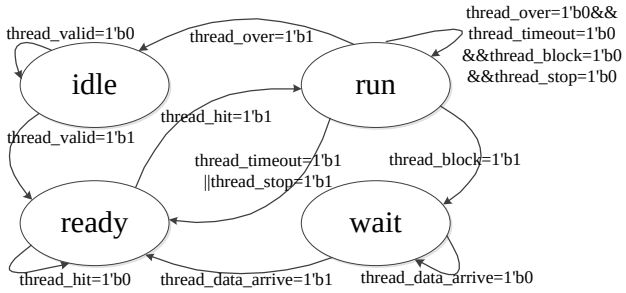

Fig.5 *tm_ctrl* module state machine

6 Simulation and Performance Analysis

This paper completes functional simulation in ModelSim, and edits 4x4 array assembler. Simulation verifies that the thread manager works correctly and produces correct results. The TM can run, switch and stop the thread properly [7].

After the completion of functional simulation, this paper made a simple performance analysis. This article uses the eight different 16-core PE program without TM. The first program load in No. 0 thread, the second program load in No. 1 thread, and so on, eight threads are loaded corresponding program.

The statistics, the total number of program calculating is 3127 clocks with the thread manager; the total number of eight programs calculating is 3762 clocks without the thread manager. According to performance formula (1).

$$\text{Performance} = \frac{\text{Clocks without TM - Clocks with TM}}{\text{Clocks without TM}} \quad (1)$$

The performance value is 16.9%.

7 Summary

This paper proposes a hardware thread manager for the PAAG polymorphic parallel processor. The hardware thread manager has been implemented and verified on a Xilinx V6-550 FPGA board. In MIMD mode, this hardware thread manager can manage up to 8 threads. The thread manager enables the efficient use of the blocking mode execution. It can greatly enhance the utilization of the processing elements and reduce processor power consumption.

Future research work will focus on more extensive testing and analysis of the performance of the thread manager. According to the analysis results improve the thread manager scheduling algorithm to achieve higher performance requirements.

Acknowledgements This study is supported by the National Science Foundation of China (61136002) and the Shaanxi Province Science and Technology Research and Development Program (2011K06-47).

References

1. Keckler S W, Dally W J, Khailany B, et al. GPUS and the future of parallel computing[J]. IEEE Computer, 2011, 44(9): 7-17
2. Marowka A, Gan R. Back to Thin-Core Massively Parallel Processors[J]. IEEE Computer, 2011, 44(12): 49-54
3. Liu Jin-Guang, Liang Man-Gui. The Development and the Software System Architecture of Multi-core Multi-threading Processor[J]. Micro-processors, 2007, 1: 1-7
4. Tao Li, L. Xiao, H. Huang and J. Han, "PAAG: A Polymorphic Array Architecture for Graphics and Image Processing", Proc. 5th Int. Symp. Parallel Architectures, Algorithms and Programming (PAAP2012), Dec. 2013, Taipei, Taiwan, IEEE Computer Society CPS, pp242-249.
5. Dongrui Fan, Hao Zhang, Da Wang, Xiaochun Ye, Fenglong Song, Guojie Li, Ninghui Sun. Godson-T: An Efficient Many-Core Processor Exploring Thread-Level Parallelism[J]. IEEE Computer Society, 2012, 32(10): 38-47
6. Liu Chung-Laung, Layland J W. Scheduling al-gorithms for multiprogramming in a hard-real-time environment[J]. Journal of the ACM, 1973, 20(1): 46-61
7. T. F. Tsuei, W. Yamamoto. Queuing simulation model for multiprocessor systems[J]. Computer, 2003, 36(2): 58–64