Collaborative Multithreading: An Open Scalable Processor Architecture for Embedded Multimedia Applications

Wei-Chun Ku, Shu-Hsuan Chou, Jui-Chin Chu, Chih-Heng Kang, Tien-Fu Chen, and Jiun-In Guo

Computer Science and Information Engineering,
National Chung Cheng University,
Chiayi, Taiwan, R.O.C.,
{kuwc, csh93, cjc, kch91, chen, jiguo}@cs.ccu.edu.tw

Abstract

Numerous approaches can be employed in exploiting computation power in processors such as superscalar, VLIW, SMT and multi-core on chip. In this paper, a UniCore VisoMT processor is proposed, which unifies VLIW and multithreading by providing an efficient control and data communication model, while offering explicit parallelisms for embedded applications. The architecture concurrently executes a main thread and several accelerative threads, coordinated by the main thread. A switch-based register-file is provided for fast data exchange between these accelerative threads. Moreover, a SMT helper function unit is employed for controlling and resource-sharing between accelerative threads, and an event-driven mechanism is introduced for synchronization between the main thread and these accelerative threads. Our results show that the proposed architecture provides area and performance advantages for embedded multimedia applications.

1. Introduction

Contemporary portable and home AV electronics with increasing embedded multimedia functionality demand large-volume data processing and computation power on SOC engines. The complication is exaggerated by requirements of supporting multi-modes on a single system, such as MPEG 1/2/4 and H.264. Due to a high NRE cost (Non-Recurring Engineering cost) and limited time-to-market, a powerful programmable processor-based solution is preferable to those using dedicated hardware accelerators. Moreover, a configurable architecture that enables an embedded processor paying less cost to be suitable for various embedded applications without significantly changing its basic architecture is also a key design issue. Numerous approaches can be employed in exploiting computation power in processors such as superscalar, VLIW, simultaneously multithreading (SMT) [1][4], vector machine [3][5][6], and multi-core on chip (CMP) [2].

In this paper, we propose a UniCore VisoMT programming paradigm and its implementation architecture (VIrtual VLIW, SMT with Open configurable architecture), which can provide a multithreading programming model to ease programming efforts without relying on compiler to exploit instruction level parallelism as well as a very fast data exchange mechanism for high internal data bandwidth. The main theme of the proposed architecture is made to minimize integration costs at design time and to reduce the time to market as does in the multicore architecture (by independent threads), and but also provides a flexible and efficient interface for integrating required high-performance functionalities. The processor can be divided into two parts as shown in Figure 1. The left part is a normal RISC-like processor (MPU) and the right part is a cluster of several data-intensive datapaths, providing vast computing power. The collaborative multithreading part adopts a multithreading mechanism where a main thread running on the RISC processor is responsible for controls and synchronization of data communication among threads, and several helper threads, managed by the main thread, provide large and various computation capabilities for multimedia accelerations. Moreover, fast data communication and data reuse are achieved by a "bank-switchable" register file. The adding and removing of different FU's can be configurable at the design time without re-compiling the code of other threads.

We compare several multithreading architecture in the Table 1, The new architectures change the thread in every cycle in general, and the data sharing of our architecture is dual mode that is RF bank mode and memory mode. The thread type of Vector architecture and UniCore VisoMT is heterogeneous, and it more suit to write program for designer.

Table 1: The comparison of multithreading architecture

Multithread architecture	Coarse grained	Fine grained	SMT	Vector MT	UniCore VisoMT
Thread Switch	Costly stall	Every cycle	intermix	Every cycle	Every cycle
Thread parallelism	No	No	Yes	Yes	Yes
Data Sharing	Memory	Memory	Memory	Memory	Hierarchical (RF Bank、Memory)
Thread Type	homogeneous			heterogeneous	heterogeneous

2. UniCore VisoMT Processor Architecture

The UniCore VisoMT model unifies VLIW with simultaneously multithreading to achieve high performance.

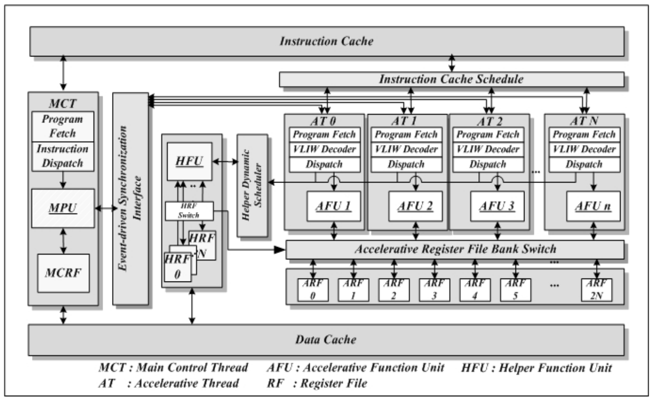

Figure 1: UniCore VisoMT Processor Architecture

The abstract architecture allows concurrent execution of a main thread with several assistant threads, which are called accelerative threads (AT). These accelerative threads, accelerating specific complex computations, are concurrently executed in a collaborative way and are coordinated by the main thread by a pre-determined execution flow. The accelerating part can be extracted by software profiling for embedded applications, such as sum of absolute difference and matrix-vector multiplication, etc. Moreover, the architectural framework provides interface for plugging functionalities, and they are scalable and configurable based on the applications required.

As shown in Figure 1, a main control thread (MCT) maintains the flow control of the whole program code, and it starts an accelerative thread by a predetermined execution flow of software analysis. The starting interface, which passes the start signal to the front-end process, is completed by an event-driven synchronization module of an AT by a package fashion. The activated AT will be started and by its program counter it executes its own firmware program code, which is coded in a 2-way VLIW format. The VLIW instruction contains two micro-operations, and the possible combination of the two may be one accelerative function unit (AFU) instruction with one helper function unit (HFU) instruction or two accelerative function unit instructions. Therefore, a firmware program code is achieved by the help of one AFU and the HFU, where the AFU executes data, and the HFU is in charge of program control.

The work load of the MPU increases significantly while maintaining the program control of each AT. We provide a small simultaneously multithreading SMT-HFU that can execute simple RISC instructions to assist in the flow control and obtainment of data for each AT. Therefore, the MPU can be freed to execute the main program control of the program code or to start an AT. Multiple parallel AT's impose on the burden of ports of instruction cache (I-cache). We simplify the number of ATs that can access the I-cache, and only one AT can access it at a time. Therefore, we need an instruction cache schedule (ICS) logic, and only the AT that has the highest priority can access it. Moreover, each VLIW instruction may have one HFU instruction. If *N* AT's concurrently executes, there will be at most *N* HFU instructions sent to the HFU each cycle. For the reason, we need a helper dynamic schedule (HDYS) logic to schedule each HFU instruction according to the priority.

The instruction format of our thread is VLIW format, each AFU focus on its work, the load/store and branch work does by HFU, each FU is work balance.

2.1 Efficient Architectural Multithreading Supports

We provide two mechanisms to enhance the efficiency for control and data communications among AT's.

- **Event-driven Communication Mechanism**

While the MCT issues start thread signals, a main control wrapper will pack the signals, and then sent the packed package to an internal event communication controller (IECC). While each AT finishes execution, each AT will also inform the MCT that it has been halted through the event-driven notification mechanism.

While adding new AFUs, we must spent time on adding control signals between the MPU and the added AFUs before. However, it is helpful to reduce the time for changing the interface while coding through the mechanism. Moreover, it reduces routing area and unpredictable bugs possibly occur on adding signals.

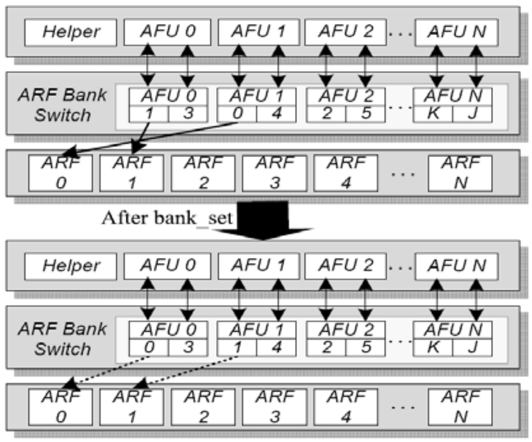

Figure 2: Accelerative register file bank switch can be reset by a start thread instruction.

- **Data Communication Mechanism**

Multimedia applications need bulk of data for computations. However, the volume of register file that contains control and data information is not sufficient for computations. It leads to numerous data movements between memory and registers, resulting in the consumption of power, and impacts the difficulty on synchronization between ATs. We provide tightly-coupled data communication between two ATs as shown in Figure 2. The ARFBS records two ARF banks used by each AT. While an AT needs data stored in an ARF bank used by another AT, the bank_set field of the start thread instruction will be used for exchanging the two ARF banks. We only have to specify the bank ID, which is now occupied by a halted AT, as one of the bank ID indicated by the bank_set field of the start thread instruction. Therefore, from Figure 2 we can see that the bank with bank ID 1 used by AT 0 is exchanged with the bank with ID 0 used by AT 1.

3. Collaborative VisoMT Programming Model

In this section, we describe how to rewrite a program code to achieve real-time constraint under the architectural model. We take a function as the basic unit for rewriting, for example, motion estimation in MPEG-4 codec. Each original program code of the complex computations is rewritten to firmware program code. While developing each firmware program code, we must realize that each AT is developed individually, which incorporates an AFU for computation and the HFU for program control. Therefore, the firmware program code must contain HFU instructions for flow controls and obtainment of data. For the reason, each firmware program code must be written by hand.

The basic concept is illustrated in Figure 3. The whole program code must be embraced by an outer while loop while programming, and it is controlled by the MT. Moreover, several complex computations have been extracted from the internal program code of the while loop. These extracted program codes are rewritten to firmware program codes. We analyze iterations of the outer loop, and make these firmware program codes that are not data dependent to be executed in parallel for performance improvement. Therefore, several ATs can be executed simultaneously to parallelize the iterations. We take video coding as an example, and use a macro block as the basic processing unit, such that each iteration loop can process a macro block at a time. Therefore, several macro blocks can be processed simultaneously. However, these processed macro blocks are not started to process at the same time because each firmware program code must be executed by its own accelerative function unit. Figure 4 depicts an example of three iterations which are processed simultaneously. Original program code is listed on the left side, and we parallelize each firmware program code with the main program code, which is depicted on the right side. Each firmware program code is executed by its corresponding AT, and each AT is started at different time.

Next we modify the whole program code. First, we must insert start thread subroutine calls to start these ATs, and it will pass parameters for each AT. The start thread subroutine call is a function call, which utilizes inline assembly code inserted in the original program code. Users must specify four parameters in the subroutine call, including the thread_id, the start address of a firmware code, bank pair number. Second, we introduce interrupt mechanism to synchronize between the main thread and the ATs. Therefore, the architects must develop interrupt service routine (ISR) for the interrupt, and the ISR will update a flag in a control register, which indicates that an AT has been halted.

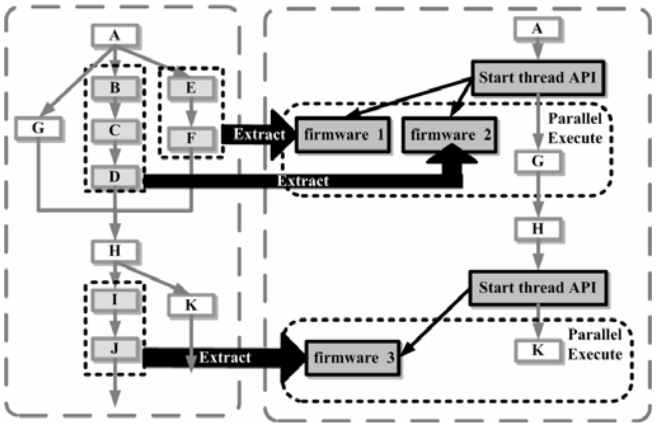

Figure 3: Extracted complex computations are rewritten

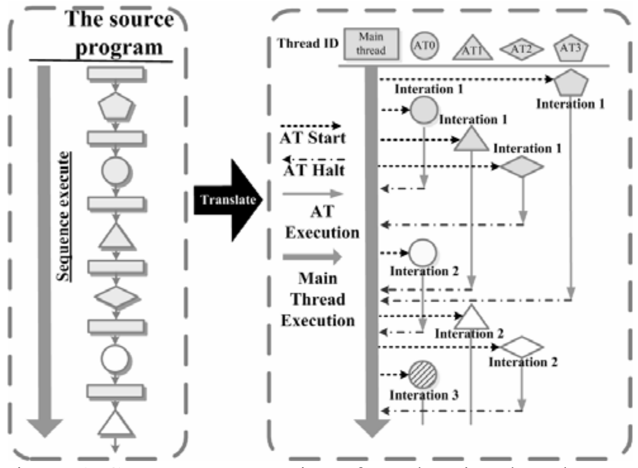

Figure 4: Concurrent execution of accelerative threads

4. Evaluation Results

For evaluating the performance of the UniCore VisoMT model, we use XviD, a public MPEG-4 codec. Based on the programming model in Section 3, we modify the encoder of the XviD program code, and build a firmware library for the extracted computations. Each firmware program code is evaluated, and the simulation results are compared with TI C64 DSP processor. Moreover, the MPU is also subsumed to compare with these processors. We use O2-level optimization option to compile these C codes for TI C64. Moreover, the firmware program codes are written in assembly language for our UniCore VisoMT. The overall system parameters for our chip are listed in Table 2. Three AFUs are integrated for the encoder, which are vector 0, vector1, and butterfly. We choose round robin as the scheduling policy for ICS and HDYS modules.

Table 2: System Parameters for VisoMT architecture

Register File	Item	Parameter
MRF	Bank * Entry * Size	1 * 32 * 32
	Read/Write Port	4R * 2W
HRF	Bank * Entry * Size	4 * 16 * 32
	Read/Write Port	2R * 1 W
CRF	Bank * Entry * Size	8 * 32 * 64
	Read/Write Port	4R * 2W
Instruction Cache	Size	8K
	Port	2
	Port Width	128 bits
	Policy	Pseudo LRU
Data Cache	Size	8K
	Port	2
	Port Width	32 bits and 64 bits
	Policy	Pseudo LRU
Policy	ICS Policy	Round Robin
	HDYS Policy	Round Robin
	Memory Switch Policy	FCFS
Viewable ARF Banks of Partial Mapping	Vector0	0、1、2、3、4、5
	Vector1	4、5、6、7
	Butterfly	0、1、2、3、4、5

Table 3 illustrates the comparison of the UniCore VisoMT processor and TI's C64 DSP. Moreover, the TI C64 DSP processor averagely reaches 5.51 frames per second, while the UniCore VisoMT can reach 15.93 frames per second.

Table 3: Comparisons of MPU, TI C64, and VisoMT

Frames Per Second@352x288 Resolution in 200MHz			
Testcase	MPU	TIC64	VisoMT
Akiyo	6.16	11.84	17.35
Bus	3.85	5.37	15.18
Foreman	4.15	6.12	15.25
Average	4.72	7.77	15.93

4.2 Chip Implementation

Each module of the UniCore VisoMT is synthesized and reported in Table 4. Three primary fields are listed in the table – main control thread, accelerative thread, and memory system. We also listed the timing constraint, actual arrival time, area, and the number of each module in the table. From the table, we can see that the UniCore VisoMT can run at 200MHz with the area of approximate 0.47 cm^2 in 180mm process. Moreover, the butterfly function unit is designed for DCT and quantization, which consists of 16 multipliers and a quantized table, such that it is the critical path in the architecture. We provide 8k I-cache, and 8k D-cache.

Table 4: The Area and Speed of Each Module in 180 mm.

Areas and Speeds of Modules				
Module Name	**Timing Constraint (ns)**	**Arrival Time(ns)**	**Area(μm^2)**	**Counts**
Main Thread				
MCT Frontend	3	2.08	101588	1
MPU	5	4.38	424485	1
MCRF	1.5	1.5	120960	1
Accelerative Thread				
ICS	1	1.27	29335	1
AT Frontend	3	2.03	242010	1
HFU	5	4.41	403299	1
Vector FU	5	2.74	183058	2
Butterfly FU	5	4.78	782838	1
HDYS	1	0.65	13029	1
HRFS	1	0.38	13425	1
HRF	1.5	1.5	46284	4
ARFBS	1	0.75	111663	1
ARF	1.5	1.5	201600	8
Memory System				
I-Cache Controller	3	2.38	261706	1
D-Cache Controller	3	2.78	688172	1
Cache Data Bank	2	1	1493863	8
Cache Tag Bank	2	1	88146	8
Cache LRU Bank	2	1	35046	2
Total	5	4.38	500*9400	1

5. REFERENCES

[1] M. Chaudhuri and M. Heinrich, "*SMTP: An architecture for next-generation scalable multi-threading*". In Proc. of 31st ISCA, June 2004, 124–135.

[2] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi and K.I. Farkas, "*Single-isa heterogeneous multi-core architectures for multithreaded workload performance*" In Proc. of 31st ISCA, June 2004, 64–75.

[3] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper and K. Asanovic, "*The vector thread architecture*". In Proc. of 31st ISCA, June 2004, 52–63.

[4] Il Park, B. Falsafi, T.N. Vijaykumar, "*Implicitly-multithreaded processors*". In Proc. of 31st ISCA, June 2003, 39–51.

[5] C. Kozyrakis, D. Patterson, "*Overcoming the limitations of conventional vector processors*" In Proc. of 31st ISCA, June 2003, 399–409.

[6] B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, S. Ciricescu, R. Essick, and A. Saidi, "*The reconfigurable streaming vector processor*". In Proc. of 36th MICRO, 2003, 141–150.

[7] G. L. Corinna and G. S. Mark "*Simple vector microprocessors for multimedia applications*". In Proc. of 36th MICRO, 1998, 25–36