Linköping University Post Print

ePUMA: a novel embedded parallel DSP platform for predictable computing

Jian Wang, Joar Sohl, Olof Kraigher and Dake Liu

N.B.: When citing this work, cite the original article.

©2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Jian Wang, Joar Sohl, Olof Kraigher and Dake Liu, ePUMA: a novel embedded parallel DSP platform for predictable computing, 2010, International Conference on Information and Electronics Engineering, (5), 32-35.

<http://dx.doi.org/10.1109/ICETC.2010.5529952>

Postprint available at: Linköping University Electronic Press

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-58573>

ePUMA: a Novel Embedded Parallel DSP Platform for Predictable Computing

Jian Wang, Joar Sohl, Olof Kraigher, Dake Liu

Department of Electrical Engineering

Linköping University, Sweden

{jianw, joar, kraigher, dake}@isy.liu.se

Abstract—In this paper, a novel parallel DSP platform based on master-multi-SIMD architecture is introduced. The platform is named ePUMA¹ [1]. The essential technology is to use separated data access kernels and algorithm kernels to minimize the communication overhead of parallel processing by running the two types of kernels in parallel. ePUMA platform is optimized for predictable computing. The memory subsystem design that relies on regular and predictable memory accesses can dramatically improve the performance according to benchmarking results. As a scalable parallel platform, the chip area is estimated for different number of co-processors. The aim of ePUMA parallel platform is to achieve low power high performance embedded parallel computing with low silicon cost for communications and similar signal processing applications.

Keywords—chip multi-processor; SIMD architecture; multi-bank memory; conflict-free memory access; data permutation

I. INTRODUCTION

Parallel processing has emerged as an area to meet the ever-increasing computing demands in real-time signal processing. Existing parallel solutions are mostly based on general multi-core platform with a cache coherent programming model [2] or custom implementations with an application specific hardware design [3]. General solutions are not cost and energy efficient for embedded systems. Custom solutions are excellent only for a selection of applications. Parallel architectures with on-chip scratch pad memory and very large register file were developed [4]. However, large register files consume much power. Recently, the master-multi-SIMD architecture has emerged in high performance parallel computing, for example the CELL processor from STI [5]. The Cell architecture provides up to hundreds GOPS computing power for a wide range of applications. However, the power consumption of CELL is not satisfactory for some applications.

From the computational perspective, the major part of high performance embedded algorithms and applications are based on predictable computing kernels. Most of the computations use regular and repetitive data accesses. For example, the memory access patterns for video coding are analyzed in [6]. The data accesses tend to be data independent, meaning that the locations of the data in memory do not depend on the data values and can be

determined beforehand. Therefore, when designing a parallel platform for such applications, it is important to have both the hardware architecture and the software programming model rely on regular and predictable memory access. The ePUMA project at Linköping University aims at providing such a parallel platform optimized for predictable computing. Its essential technology is to separate data access from arithmetic computing and run them in parallel to save execution time.

In a recent publication [7] we presented a white paper of the ePUMA project. The programming flow was described in detail in the white paper. The originally proposed multi-core architecture is with one master controller, eight SIMD processors, two ring buses, and two DMA controllers. In this paper, a new interconnection network is designed to replace the previous two buses. The new architecture is presented in section II. Section III is about the memory subsystem design essentials. This paper also brings the chip area estimation in section IV and some more benchmarking results in section V. The paper is ended with a conclusion and a discussion of future work in Section VI.

II. MASTER-MULTI-SIMD ARCHITECTURE

The ePUMA master-multi-SIMD architecture is illustrated in Figure 1. It consists of one master controller, eight SIMD coprocessors, and a memory subsystem for the on-chip communication. The master processor executes the sequential task in an application algorithm, while the SIMD cores run the parallelizable portion of the algorithm. Usually for embedded streaming applications, the MIPS cost of the

Figure 1. ePUMA master-multi-SIMD architecture

¹ ePUMA, embedded Parallel computing architecture with Unique Memory Access, is a project at Linköping University and supported by the Swedish Foundation for Strategic Research (SSF).

Figure 3. Memory hierarchy

sequential task in the algorithm is about 10%.

Each SIMD has a local program memory (PM) and data memory (DM). DM is a vector memory which can exchange data with main memory through the central DMA controller. The vector data from one SIMD could also be sent to any other SIMD(s) by the packet based interconnection network with eight switching nodes.

III. MEMORY SUBSYSTEM

The goal of the memory subsystem design is to minimize the data access cost for the SIMD cores. It should hide the communication overhead as much as possible so that the execution time approaches the computing time for the arithmetic instructions. This is possible for regular and predictable memory accessing patterns.

A. The memory hierarchy

The ePUMA memory hierarchy is compatible with the OpenCL specification. OpenCL (Open Computing Language) is a programming framework for heterogeneous parallel platforms and will be used in the software tool-chain of ePUMA. As illustrated in Figure 2, the memory hierarchy consists of three layers. The highest layer is the off-chip main memory, which is with a low clock rate and has the longest access latency from the processing cores. The local memory as the second level computing buffer includes both data memory and program memory. The master controller uses two data memories and a cache as the local program memory. The SIMD processors use eight-bank vector memory as the local data buffer and a simple scratchpad memory for program. The lowest layer in the memory hierarchy is the register files in the master and SIMD processors.

It takes two steps to move a vector data from main memory to the SIMD data-path across the memory hierarchy. The first step is to load data from main memory to SIMD local vector memory. This is done by a DMA transaction issued by master controller. In order to hide this communication time, ePUMA implements ping-pong buffer in local vector memory. By this way the DMA transaction is overlapped with the SIMD computing. The second step is to load data from the local vector memory to the SIMD register.

Figure 2. SIMD local vector memory with data permutation

It is also beneficial if the SIMD load instruction can run in parallel with its arithmetic instructions. The ePUMA platform implements SIMT (Single Instruction Multiple Tasks) [8] instructions to achieve this. The SIMT instruction is a task level instruction, which is usually an iterative loop function and is handled by a FSM in the data-path. By SIMT instructions, the run time cost of loading data to register file can be negligible.

The data is stored back to the main memory in a reversed process. The ways of overlapping data access with SIMD computing in the two steps are the same as loading data.

B. SIMD local memory and data permutation

SIMD local memory uses a group of scratch pad memory blocks instead of cache. Three vector memories are prepared for each SIMD; each one is composed of eight single-port SRAMs. At run time, two vector memories are accessible from SIMD data-path, and the third one can do data exchange with either the main memory or other SIMD(s). A software controlled switch circuit is implemented to achieve the ping-pong swapping.

As the vector memory is used in the SIMD local buffer, it is essential to achieve conflict free parallel memory access to provide vector data to SIMD data-path at a minimum latency. Conflict free memory access is important when the designers want to save SIMD register size for cost and power reasons, especially when doing transform and matrix operations. The solution to conflict free memory access is to use data permutation. In hardware design, two permutation tables are implemented in the SIMD local memory. One is used for DMA access and another is for SIMD access. In the software side, prolog code of a kernel running in master processor selects the permutation function and configures the permutation tables in SIMD. Since the data access kernels are separated from and orthogonal to its computing kernel in ePUMA, designers for the data access kernels can therefore focus on the design for conflict free data access.

A well known data permutation example is shown in Figure 4. It shows two ways to store a 4*4 matrix in the local vector memory. In Figure 4(a), when there is no permutation, it can be observed that only row vectors can be accessed without bank conflict. In Figure 4(b), a permutation is applied while loading data to the memory. In this way both row and column vectors can be accessed in parallel.

Figure 4. A data permutation example

C. Main memory and DMA

One off-chip main memory is used to store both data and program codes. A DMA controller is designed to transfer data between main memory and on-chip buffers. The DMA transactions are initialized by the master processor. When one transaction finishes, it interrupts the master. The DMA controller separates task configuration from data transaction. Multiple tasks could be configured and stored in a task queue. The master controller can configure the next DMA task during the current DMA transaction. Thus as soon as the current one finishes, the next transaction can start immediately. Another feature of this DMA controller is the broadcasting mode. The same data can be sent to different SIMDs within one DMA transaction. One use of DMA broadcasting is to support cache coherence between SIMDs. The dirty data from one SIMD is sent to all the other SIMDs from the central DMA controller.

D. On-chip network

As discussed in our previous publication [7], the first proposed interconnection network is to use normal buses, one shared bus for program code and one cross-bar for data. These two buses are implemented in parallel connecting two off-chip memories and on-chip processor cores. Each bus has its own DMA controller and a bridge is designed to support communication between two shared buses.

In the new interconnection architecture, we use a star connector for DMA transactions, a ring connector for streaming computing, and mixed star and ring for emulating cache coherence. The DMA controller is designed with multi I/O ports; it has direct connection to every processing core's

Figure 5. SIMD streaming mode

local memory. We change the interconnection between SIMDs to a packet based network-on-chip in a ring structure (Figure 1) for the following reasons. First, in ePUMA programming model, the software can explicitly control the task assignment to SIMD and schedule the data transactions on-chip. Therefore the on-chip traffic can be quite regular

when implementing an algorithm on ePUMA. Actually the most common on-chip communication model of ePUMA is the streaming model, as illustrated in Figure 5. Second, the implementation complexity of a cross-bar is larger than a ring network, especially when we scale up the number of SIMDs. The third reason is for low power. Both shared bus and cross-bar are not as power efficient as the network-on-chip interconnection.

IV. CHIP AREA ESTIMATION

The chip area is estimated in this section. Table I lists the local memories for master and SIMD, both program memory and data memory are specified. In Table II, the logic gates estimation is listed. We use a single MAC DSP controller [8] as the master processor, which has already been done in our group. The total gate count of the master processor is about 50k gates. The SIMD logic estimation is divided into three parts, the register file, the data-path, and the control-path plus AGU. The register file includes both SIMD vector registers and permutation table implementation.

The ePUMA architecture is fully scalable and the number of SIMD co-processors can be configured from 1 up to 8. Table III shows the final chip area estimation (65nm) with different number of SIMD co-processors.

TABLE I. LIST OF PROCESSOR LOCAL MEMORIES

	Memory module	Word length	Size	Module number	Total size
Master	PM	32-bit	32k	1	1Mb
	DM	32-bit	16k	2	1Mb
SIMD	PM	128-bit	1k	1	0.1Mb
	LVM	128-bit	5k	3	1.9Mb

TABLE II. LOGIC GATES ESTIMATION

	Sub-module	Total gates
Master	-	50k
SIMD	Registers	128k
	Data-path	80k
	Control-path + AGU	60k
Memory subsystem	-	150k

The ePUMA architecture is fully scalable and the number of SIMD co-processors can be configured from 1 up to 8. Table III shows the final chip area estimation (65nm) with different number of SIMD co-processors.

TABLE III. TOTAL GATE COUNT AND CHIP AREA ESTIMATION WITH DIFFERENT NUMBER OF SIMDs

Number of SIMDs	1	2	4	8
Total memory (Mb)	4	6	10	18
Memory gates (gates)	2M	3M	5M	9M
Logic gates (gates)	468k	736k	1272k	2344k
Total gates (gates)	2.5M	3.8M	5.3M	11.4M
Area (mm ²)	5	8	11	23

V. BENCHMARKING RESULTS

The ePUMA platform is designed to achieve a goal of minimizing the communication overhead in parallel processing. We use the ratio R (Equation 1) as a measurement of efficiency when evaluating the ePUMA platform at the beginning of this project.

$$R = \frac{\text{total_cycles}}{\text{arithmetic_instructions}} \quad (1)$$

The ideal value of R is 1, that is, the total execution time is just the arithmetic computing time and the data access overhead equals to 0. Table IV shows the benchmarking results which are very inspiring. The algorithms include matrix multiplication (MMUL), LU decomposition (LU), and fast Fourier transform (FFT).

TABLE IV. BENCHMARKING RESULTS

Algorithm	MMUL[9]	LU[7]	64k FFT
R (ePUMA)	1.154	1.668	1.75
R (Conv. SIMD)	7.469	4.656	1.67

VI. CONCLUSION AND FUTURE WORK

ePUMA is a parallel DSP platform optimized for predictable computing. It aims at providing a low power high performance embedded parallel computing platform with low silicon cost. In this paper, the master-multi-SIMD architecture has been introduced. The memory subsystem architecture, conflict free vector memory access, and the interconnection network were described. Preliminary chip area estimation is done for ePUMA as a scalable platform. The finished work in the first year includes a white paper as the design guideline, a behavior simulator and some benchmarks as an early proof of ePUMA's design essentials.

The future work is to improve the cycle accuracy of ePUMA behavior simulator, which will be done using our cycle accurate simulation framework specifically developed for the ePUMA project. Models of SIMD processor core and memory subsystem will be further improved and integrated

into the simulator. The existing master processor simulator will be wrapped into one module and compiled into the ePUMA simulator. After updating the behavior model, more algorithms will be mapped to ePUMA platform and be simulated in the cycle true simulation. More benchmarking results are expected to prove the efficiency of ePUMA platform. We also need to find the optimal hardware configuration, the best fit permutation table size, and the addressing patterns for the vector address generation unit from the later benchmarking.

ACKNOWLEDGMENT

The authors would like to thank SSF, Swedish Foundation for Strategic Research, for the support of this project.

REFERENCES

- [1] "ePUMA project at Linköping University Sweden," <http://www.da.isy.liu.se/research/scratchpad/>.
- [2] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 3rd Edition. Morgan Kaufmann, 2002.
- [3] A. Nilsson, E. Tell, and D. Liu, "70mW Fully Programmable Baseband Processor for Mobile WiMAX and DVB-T/H in 0.12m CMOS," ISSCC, 2008.
- [4] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. Dally, "A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing," IEEE Journal of Solid-State Circuits, vol. 43, pp. 202–213, 2008.
- [5] M. Gschwind, "The Cell Broadband Engine: Exploiting Multiple Levels of Parallelism in a Chip Multiprocessor," International Journal of Parallel Programming, vol. 35, no. 3, 2007.
- [6] J. K. Tanskanen and J. T. Niittylahti, "Scalable Parallel Memory Architectures for Video Coding," The Journal of VLSI Signal Processing, vol. 38, no. 2, pp. 173–199, 2004.
- [7] D. Liu, J. Sohl, and J. Wang, "Parallel Computing and its Architecture Based on Data Access Separated Kernels," IJERTCS, International Journals Embedded and Real-Time Communication systems, 2009.
- [8] D. Liu, Embedded DSP Processor Design. Linköping, Sweden: Morgan Kaufmann, 2008.
- [9] J. Sohl, J. Wang, and D. Liu, "Large Matrix Multiplication on a Novel Heterogeneous Parallel DSP Architecture," APPT, International Conference on Advanced Parallel Processing Technologies, 2009.