MaPU: A Novel Mathematical Computing Architecture

Donglin Wang
CASIA, Beijing, China

Shaolin Xie *
CASIA, Beijing, China

Zhiwei Zhang
CASIA, Beijing, China

Xueliang Du
CASIA, Beijing, China

Lei Wang, Zijun Liu
CASIA, Beijing, China

Xiao Lin, Jie Hao
CASIA, Beijing, China

Leizu Yin
Spreadtrum Comm, Inc.

Tao Wang
Huawei Tech Co, Ltd.

Yongyong Yang
Huawei Tech Co, Ltd.

Chen Lin, Hong Ma
CASIA, Beijing, China

Zhonghua Pu,Guangxin Ding
CASIA, Beijing, China

Wenqin Sun, Fabiao Zhou
CASIA, Beijing, China

Weili Ren, Huijuan Wang
CASIA, Beijing, China

Mengchen Zhu, Lipeng Yang
CASIA, Beijing, China

NuoZhou Xiao, Qian Cui
CASIA, Beijing, China

Xingang Wang
CASIA, Beijing, China

Ruoshan Guo
CASIA Beijing China

Xiaoqin Wang
CASIA Beijing China

ABSTRACT

As the feature size of the semiconductor process is scaling down to 10nm and below, it is possible to assemble systems with high performance processors that can theoretically provide computational power of up to tens of PLOPS. However, the power consumption of these systems is also rocketing up to tens of millions watts, and the actual performance is only around 60% of the theoretical performance. Today, power efficiency and sustained performance have become the main foci of processor designers. Traditional computing architecture such as superscalar and GPGPU are proven to be power inefficient, and there is a big gap between the actual and peak performance. In this paper, we present the MaPU architecture, a novel architecture which is suitable for data-intensive computing with great power efficiency and sustained computation throughput. To achieve this goal, MaPU attempts to optimize the application from a system perspective, including the hardware, algorithm and corresponding program model. It uses an innovative multi-granularity parallel memory system with intrinsic shuffle ability, cascading pipelines with wide SIMD data paths and a state-machine-based program model. When executing typical signal processing algorithms, a single MaPU core implemented with a 40nm process exhibits a sustained performance of 134 GLOPS while consuming only 2.8 W in power, which increases the actual power efficiency by an order of magnitude comparable with the traditional CPU and GPGPU.

*Corresponding author; email: shaolin.xie@ia.ac.cn

978-1-4673-9211-2/16/$31.00 ©2016 IEEE

1. INTRODUCTION

Today, power efficiency is a key factor for mobile computing. Great advances have been made to prolong battery life and provide greater computing abilities[1][2]. However, power efficiency is not only an important factor in mobile computing, but also a key metric in supercomputing.

As Moore's Law is still effective, tremendous transistors can be used for building super processors like Intel's Xeon Phi co-processor and the Nvidia GPU. The Intel Xeon Phi 7120p co-processor was built with 61 cores, providing a theoretical peak performance of 2.4 TFLOPS. The latest Nvidia Kepler GK210 provides a theoretical peak performance of 4.37 TFLOPS. However, the peak power levels of these two chips are 300 W and 150 W, respectively, which means their power efficiency levels are only 8 GFLOPS/W and 29 GFLOPS/W, respectively. Furthermore, these figures represent their theoretical power efficiency; their actual power efficiency is even lower. As reported by Green500, which aims to provide a ranking of the most energy-efficient supercomputers in the world, the most power-efficient supercomputer is only 0.7 GLOPS/W,

Although the power consumption of a single processor

is not necessarily a disadvantage for indoor systems with a sustained power supply, the aggregate power of those supercomputers built with thousands of super processors would eventually limit the scale of the system given the increasing cost of deployment and maintenance involved. For example, the most powerful super computer Tianhe2 consumes 24 MW(with a cooling system) and occupies 720 square meters of space. To build such a system, dedicated computer complex and power station are needed.

Another problem with today's computing is the gap between peak performance and sustained performance[3]. Only a few improvements have been made today[4]. The actual performance of the most two powerful supercomputers (ranked by Top500 in June 2015) is only 62% and 65% of the corresponding peak performance, respectively, even with a very structured algorithm such as LINPACK applied. As their sustained performance is much lower than their theoretical performance, the actual power efficiency of their processors is less than the theoretical power efficiency.

Many factors contribute to the performance gap[3], although the most important one is that the compilers usually implicitly use simplified models of processor architecture and do not take the detailed workings of each processor component into account, such as the memory hierarchy, SIMD instruction extensions, and zero overhead circulations. It has been reported that the mean usage for various GPU benchmarks is only 45%, and that the main sources of underuse are memory stalls, which are caused by memory access latency and inefficient access patterns [4]. As a result, most processors rely heavily on hand-optimized libraries to boost actual performance in many applications, which makes the compiler subsidiary in performance critical program.

Given the current abundance of chip transistors, many new architectures leverage various ASIC-based accelerators to increase the power efficiency and narrow the performance gap. However, these architectures are inflexible and require great effort to design a chip for specific applications. We aimed to construct a programmable accelerator architecture from a system perspective that can provide performance and power efficiency comparable with ASIC implementation and can be tailored by the programmer to specific workloads. The intuitive strategy we adopt is to map the mathematical representation of the computation kernel into massive reconfigurable computing logics, and map the data into highly reconfigurable memory systems. We call this architecture MaPU, which stands for Mathematical Processing Unit.

In this paper, we first discuss the considerations involved in designing MaPU. We then introduce the instruction set architecture of MaPU in Section 2. The highlights of the MaPU architecture are presented in Section 3. To prove the advantages of the MaPU architecture, a chip with four MaPU cores is designed, implemented and taped out with a 40-nm process. The structure, performance and power of this chip are fully analyzed in Section 4.

2. RETROSPECT AND OVERVIEW OF MAPU ARCHITECTURE

It took us a long time to design with a feasible micro-architecture for MaPU. Traditional superscalar has proved to be inherently power inefficient[5], and GPGPU is also power hungry. As such, our work excluded both of those architectures. Strategies such as VLIW and SIMD were taken into consideration. The first proposed MaPU micro-architecture featured customized wide vector instructions in a RISC style and an innovative multi-granularity parallel (MGP) memory. This method was discarded for its low efficiency. We then proposed a micro-architecture that included massive computing units with hardwired state machines, in which each computation kernel was represented by a state machine. This micro-architecture manifested high performance and power efficiency. However, as we tried to support more kernels, the state machine became so complex that the circuit could only run at a much lower frequency. The micro-architecture then evolved into the current one in which the state machines were broken down into microcodes and became programmable. This provided the possibility of supporting various kernels with customized state machines.

As an accelerator framework, MaPU is made up of three main components: the microcode pipeline, MGP memory and scalar pipeline, as shown in Figure 1.

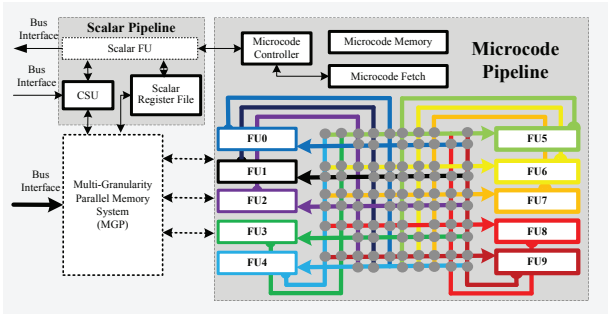


Figure 1: MaPU Architecture Framework

The scalar pipeline is subsidiary and can be a simple RISC core or a VLIW DSP core. It is used to communicate with the system on chip (SoC) and controlling the microcode pipeline. The Communication and Synchronization Unit (CSU) in the scalar pipeline includes a DMA controller used to transport high dimensional data to and from the SoC and some control registers that can be read/written by other SoC masters to control or check the status of the MaPU core.

Although MaPU is an accelerator architecture, we include this scalar pipeline to facilitate the interaction between the MaPU cores and SoC. For example, the scalar pipeline includes exclusive load/store operation pairs to support multi-thread primitives such as atomic addition, spin lock and fork/join. Therefore, it would be easier to develop a MaPU runtime library to support a multi-core program framework such as OpenMPI, OpenMP and OpenCL.

2.1 Microcode pipeline

The microcode pipeline is the key component of the MaPU architecture. The functional unit (FU) can be an ALU, MAC or any other module with special functions. Superscalar components such as register rename logics and instruction issue windows are power inefficient[5]. To eliminate the power consumed by control logics, the FUs in MaPU are controlled by microcodes in VLIW style. There is a highly

structured forwarding matrix between FUs, and its routing is controlled dynamically by microcodes. In this way, the FUs can cascade into a complete data path which resembles the data flow of the algorithm. Data dependence and routing are handled by the program to further simplify the control logic.

This micro-architecture, which consists of massive FUs with a structured forwarding matrix, manifests high performance and power efficiency but leaves all of the complexity to the programmers. This is plausible because computation kernels are usually simple and structured, such as the FFT and matrix multiply algorithms. A library that includes common routines would decrease the complexity for programmers.

The microcode pipeline has many features in common with coarse grain reconfigurable architecture(CGRA), but with two enhancements. First, all of the FUs and forwarding matrix in MaPU operate in the SIMD manner and have the same bit width, which is supposed to be wide. Wider SIMD can amortize the power overhead of instruction fetch and dispatch, bringing more benefits in terms of energy efficiency. Second, the microcode pipeline has a highly coupled forwarding matrix instead of dedicated routing units. With this forwarding matrix, FUs can cascade into a compact data path that resembles the data flow of the algorithm. Therefore, it can provide performance and power efficiency comparable with that of ASIC. Furthermore, as each FU has a dedicated path to and from other specific FUs, programmers do not have to consider the data routing congestion problem, making instruction scheduling much simpler than CGRA. These benefits comes at the cost of extra wires, which occupy a considerable chip area. In fact, some implementation has to divide the forwarding matrix into two stages to decrease the connecting wires between FUs, and only part of the FUs maybe connected depending on the characteristics of the applications .

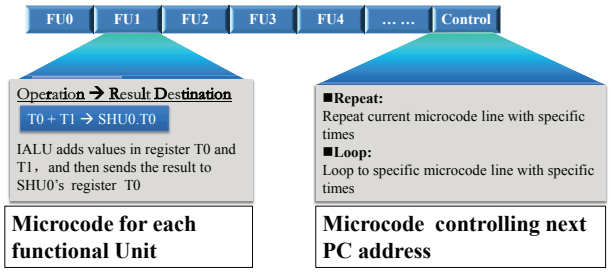


Figure 2: Microcode line Format

There are N microcodes issued in a clock period for an implementation with N FUs. These microcodes issued at the same time are called microcode lines, which are stored in the microcode memory and can be updated at runtime. This coding schema is simple and effective. However, for most kernels, only a few FUs are working simultaneously most of the time; thus, there are NOPs in most microcode lines. A compression strategy can be used to increase the code density in the future.

The microcode line format is illustrated in Figure 2. Each microcode for FU has two parts: "Operation" and "Result Destination." "Operation" tells the FU what should be done, and "Result Destination" tells the forwarding logic how to route the result. At the end of the line is the microcode for the controller, which takes charge of the microcode fetch and dispatch. There are two types of microcodes for the controller: repeat and loop. These two types of microcodes instruct the controller to repeat or loop to a specific microcode line at certain times. The MaPU architecture incorporates these two special microcodes to support the widespread nested loop structure in kernel applications.

2.2 Multi-granularity parallel (MGP) memory

As mentioned in previous work [4], memory access patterns are an important source for processor's underuse. However, computation kernels are always structured, and their memory access patterns can be classified into a few categories. Therefore, it is highly possible to normalize these patterns with a strict data model and special designed memory system.

The MGP memory system serves as a soft managed local memory system and is designed with an intrinsic data shuffle ability, supporting various access patterns. Providing a row- and column-major layout simultaneously for matrices with common data types, this memory architecture makes the time-consuming matrix transposition unnecessary. With the MaPU data model, matrices in the MGP memory system can be treated as normal and transposed forms at the same time. We explore this feature in more details in Section 3.1.

3. ARCHITECTURE HIGHLIGHTS

3.1 MGP memory system

The MGP memory system supports various access patterns, especially the simultaneous row- and column-major layout for matrices with common data types. Before describing the structure of the MGP memory system, some basic concepts should be explained.

- Physical bank: On-chip SRAM that can be accessed with multiple bytes in parallel,which can be generated from a memory compiler or customized.
- Logic bank: a group of physical banks, on which the address resolution is based.
- Granularity parameter: the parameter controls which physical banks should be grouped into a logic bank and how the inputted address is resolved.

3.1.1 Basic structure of the MGP memory system

The MGP memory system provides **W** bytes of parallel access and N bytes of capacity and has three interfaces: one read/write address, a granularity parameter(G) and data for reading or writing. The MGP memory system has the following constraints.

- W should be an integer to the power of 2, and $N=2^k W$, where k must be a natural number.
- G should be an integers to the power of 2, ranging from 1 to W. That is, $G=2^k$, where $0 \leq k \leq log_2 W$.

- W **physical banks**, labeled from 0 to W-1, are required. Each bank can read/write W bytes in parallel and has N/W-byte capacity.

When accessed, this memory system operates according to the following procedures **to decode the address**.

1. **Logic bank formation:** Physical banks cascade and group into logic banks according to parameter G. G consecutive physical banks labeled with i*G to (i+1)*G-1 cascade into a logic bank i, where $0 \leq i < \frac{W}{G}$.
2. **Address mapping:** Physical banks in a logic bank are addressed in sequence, starting from zero. All of the logic banks have the same address space. As the size of each physical memory bank is N/W and there are G physical banks in a logic bank, the address of the logic bank ranges from 0 to G*N/W-1.
3. **Data access:** When reading/writing, each logic bank accesses only G bytes of the whole W bytes. The access address is the address inputted into the MGP memory system. Each physical memory bank uses a mask to control this partial access.

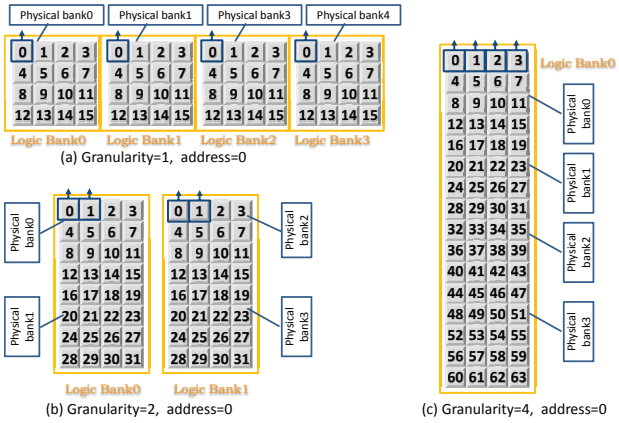


Figure 3: MGP memory example that provides W=4 bytes of parallel access and with a total capacity of N=64 bytes. (a)G=1, address=0, one physical memory bank is grouped into a logic bank and each logic bank accesses 1 byte. (b)G=2, address=0, two physical banks are grouped into a logic bank and each logic bank accesses 2 bytes. (c)G=4, address=0, four physical banks are grouped into a logic bank and each logic bank accesses 4 bytes

Figure 3 shows an MGP memory system in which W=4, N=64. With a granularity parameter G, the memory system can support log_2W+1 types of access patterns. The layout of physical memories is controlled dynamically by granularity parameter G. When G=W, the MGP memory system has only one logic bank and all of the physical banks are addressed in sequence. In this case, the memory system falls into an ordinary memory system with W bytes accessing the interface. When combined with carefully designed data layouts, the MGP memory system can provide interesting features such as simultaneous parallel access for matrix rows and columns. In fact, when writing data into memory with one G value and then reading them with a different G value, we shuffle the data implicitly. Different pairs of G represent different shuffle patterns, which makes this MGP memory system versatile in handling high dimensional data.

In fact, the MGP memory system is the most distinguishing feature of MaPU and other designs may certainly benefit from it. Other reconfigurable architectures focus mainly on computing fabric, ignoring or compensating for the performance and power penalties caused by an inefficient memory system. MGP memory tries to address the root cause of the problem. It is not like the scatter/gather-enabled memory in conventional vector processors. In such a scatter/gather operation, multiple addresses are sent to memory banks, and access conflicts are unavoidable and can lead to pipeline stall, affecting the performance and increasing the complexity of pipeline control in turn. MGP memory in MaPU is a mathematically structured, conflict-free and vectorized system. Most vector accessing patterns can be mapped into the MGP memory system, such as accessing a matrix row or column with the same data layout, and reading and writing FFT data for any stage of butterfly diagram. More of the applications that may benefit from MPG memory are explored later.

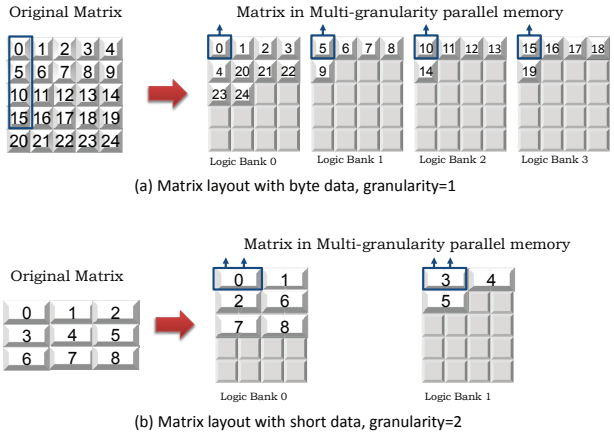


Figure 4: Matrix initial layout when (a): the element's bit width is the same as that of the addressable units and (b) the element's bit width is twice as that of the addressable units. The bit width W in this example is 4 bytes. Both (a) and (b) show only the initial layout of the matrix. The actual layout is controlled by granularity parameter G when this memory is read/written.

3.1.2 Matrix layout in the MGP memory system

An matrix can be accessed in parallel in row or column order simultaneously only if it is initialized in a specific layout in the MGP memory system. Figure 4 shows two matrices in with the initial layout can be accessed in either row or column order. Keep in mind that figure 4 shows only the layout in which the matrices should be initialized. The data layout will change according to the provided G parameter when accessed.

For matrices with elements whose bit widths are the same

as those of the addressable units, the following procedures would produce the initial layout.

- Set granularity parameter G=1.
- Put the *i*th row in logic bank i%W. Rows in the same logic bank should be consecutive.

These procedures generate the initial layout for a 5x5 matrix in an MPG memory system with W=4, as shown in Figure 4(a). Providing this data layout, the matrix can be accessed in row order with G=W and accessed in column order with G=1. When G=1 and address=0, column elements (0, 5, 10, 15) are accessed in parallel. When G=W and address=0, row elements (0, 1, 2, 3) are accessed in parallel.

In fact, a formal expression for the address offset of each row during the initialization procedure and the address offset of each element for the read/write process after initialization can be derived for general cases. Matrices with elements whose bit width is M times those of the addressable units, where M is an integer to the power of 2, should be initialized as follows (providing the capacity of the memory system is N, and the dimension of the matrix is PxQ).

- Set the granularity parameter G=W. (In this case, there is only one logic bank.)
- The address offset of the ith row is A(i) = $[i\%(\frac{W}{M})](\frac{NM}{W})+(\frac{iM}{W})QM$.

Table 1: Address offset for matrix elements (i, j), which occupy M memory units. The matrix size is PxQ. The memory width is W and the total capacity is N.

Access Mode	G	Address Offset
Row major	W	$(i\%\frac{W}{M})\frac{NM}{W}+(\frac{iM}{W})QM+W(\frac{jM}{W})$
Column major	M	$Mj+(\frac{i}{W})QM$

When masters access the matrix in the MGP memory system, they access consecutive W bytes as a whole by row or by column. The address offset for accessing elements (i, j) can be calculated as shown in Table 1. Here, "%"represents the modulo operation, and all the of divisions operate with no remainder. As masters can only access W bytes as whole, Table 1 shows only the accessing address for these W bytes. The address computation is complicated, but the stride of the address is regular. Thus, it is plausible to transverse the whole matrix with simple address generation hardware.

3.2 High dimension data model

As mentioned previously, the MGP memory system is versatile in handling high dimensional data, but its address computation is complicated and addresses are always not consecutive. To describe and access high dimensional data in the MGP memory system, a much more expressive data model other than vectors is required.

Figure 5 shows the basic parameters needed for each dimension: the base address(KB); the address stride(KS), which is the address difference between two consecutive elements; and the total number of elements (KI). Figure 6 shows the two-dimensional data described by these parameters.

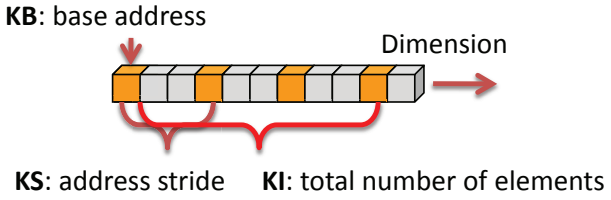


Figure 5: Parameters to describe the dimensions of data

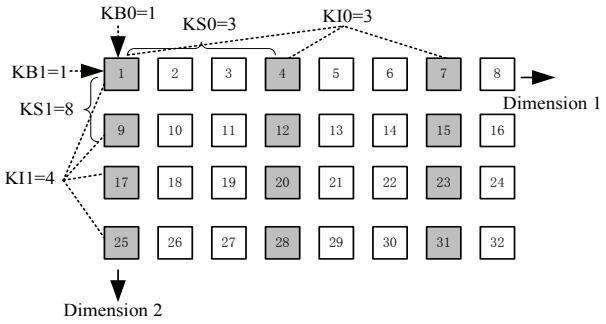


Figure 6: Example of high dimensional data description: A4x3 matrix distributed in a 4x8 space.

With this data model, the matrix described in Table 1 can be configured according to the access modes. All of the base addresses of the dimensions are the same as the start address of the matrix.

When accessed by row, the matrix in the MGP memory system can be seen as three-dimensional data. The first dimension is the elements in a row. As the memory system accesses W bytes for a time, the address stride of the dimension is W and the number of elements is QM/W. The second dimension is the rows between the logic banks, whose address stride is the size of the logic banks, i.e., NM/W. The number of elements is the number of logic banks, i.e., W/M. The third dimension is the row in the same logic bank, whose address stride is the memory units occupied by a row, i.e., QM, and the number of elements is PM/W.

When accessed by column, the first dimension is the elements in a column, whose address stride is the memory units occupied by a row, i.e., QM. The number of elements is PM/W. The second dimension is the column of the matrix, whose address stride is M, and the total number of elements is Q.

Table 2 shows the KS and KI configurations of these two access modes.

Table 2: Parameters for a matrix whose size is PxQ. Each element occupies M memory units. The memory width is W and the total capacity is N.

Access Mode	G	KS0	KI0	KS1	KI1	KS2	KI2
Row major	W	W	$\frac{QM}{W}$	$\frac{NM}{W}$	$\frac{W}{M}$	QM	$\frac{PM}{W}$
Column major	M	QM	$\frac{PM}{W}$	M	Q		

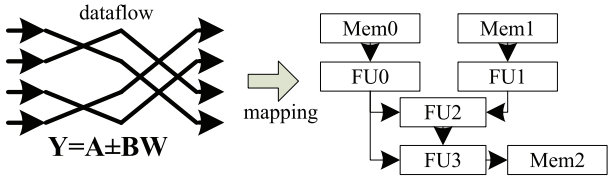
In the MaPU architecture, the number of dimensions that the chip can support depends on implementations. These

parameters are set by the scalar pipeline. Their values are stored in registers of FUs that take charge of the load/store operation. During operation, data are accessed in sequential groups where low dimensions are accessed first, followed by the high dimensions. The addresses of these groups are calculated automatically, like those in DMA operations, except that the time required to access the data is controlled by microcode.

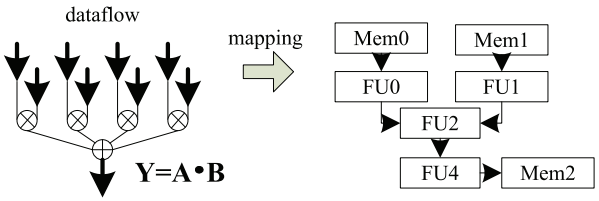
3.3 Cascading pipeline with state-machine-based program model

FUs in the microcode pipeline can cascade into the data path that suites the algorithms, like the data path in the ASIC controlled by the state machine. Different storage elements such as pipeline registers and caches have different access time and energy consumptions for each operation. This cascading structure can dramatically decrease data movement between register files and memory, thus decreasing the overall consumed energy. Moreover, keeping the dataflow centralized in FUs and forwarding paths increases the overall computation efficiency, as the FUs and forwarding logics are always running at a higher frequency then caches and memories.

Figure 7 shows this concept for the FFT and matrix multiply algorithms.



(a) Data path mapping for FFT



(b) Data path mapping for Matrix Multiply

Figure 7: Dataflow mapping in MaPU architecture, which can change dynamically through microcode sequences.

The mapping between the dataflow and data path is represented by a microcode sequence that defines the operations of the FUs at every clock cycle. As these microcodes are stored in memory and can be updated at runtime, this mapping can be also updated dynamically.

As hardware does not handle dependency between microcodes, programmer should write the microcodes carefully to make sure that the result will not come too early or too late. This involves a great number of effort. MaPU implements a-state-machine based program model to simplify this task. Programmers only need to describe the state machine of each FU and the time when these state machines should start. The compiler can then transform these state machines into microcode lines that can be emitted simultaneously.

First, the programmer establishes a state-machine-based program description. Sub-state machines for each node are constructed using the MaPU instruction set, and the top state machine is then constructed according to the time delay, i.e., data dependence. Next, according to the micro-architecture feature of MaPU, the compiler transforms each state machine into an intermediate expression that conveys the sequential, cyclic or repeated structure of the basic block. The compiler then merges the different state machines by abstracting their same attributes to generate combinational state machines and the microcode lines. Finally, the compiler implements grammatical structure, resource conflict and data dependence detection to ensure that the microcode lines satisfy the MaPU instruction set constraints.

Figure 8 shows the concept of this program model.

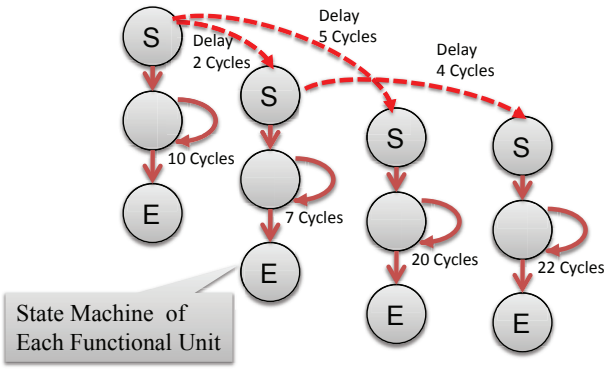


Figure 8: Illustration of state-machine-based program model.

The state machine of each FU is described by microcodes that the FU supports and microcodes for loop control. The following code snippet shows an example state machine of a load/store unit.

```
.hmacro FU1SM
//loop to label 1, loop count is stored in KI12 register
LPTO(1f)@(KI12);
//load data to Register file and calculate next load address
BIU0.DM(A++, K++)->M[0];
NOP; //idle for one cycle
BIU0.DM(A++,K++)->M[0];
NOP; //idle for one cycle
1:
.endhmicro
```

The following code snippet shows the top state machine of the algorithm.

```
.hmacro MainSM
FU1SM; //start FU1 state machine at cycle 0
REPEAT@(6); // wait six cycles
FU2SM || FU3SM;  //start FU2 and FU3 state machine at cycle 7
REPEAT@(6);  // wait six cycles
FU4SM; //start FU4 state machine at cycle 13
.endhmicro
```

4. THE FIRST MAPU CHIP

To prove the advantages of the MaPU architecture, we design, implement and tape out a chip with four cores that implement MaPU the instruction set architecture.

4.1 SoC architecture

Figure 9 shows a simplified diagram of this chip. The

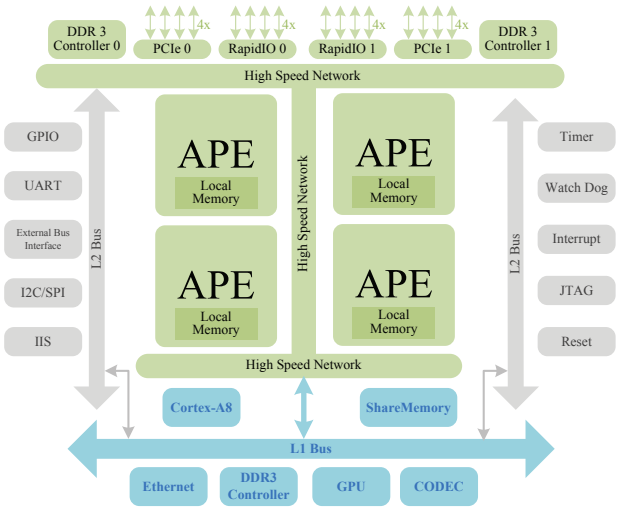


Figure 9: Simplified SoC structure

MaPU cores in this chip are called **APE**, which stands for **A**lgebraic **P**rocessing **E**ngine. In addition to the for MaPU cores, there are other subsidiary components like the Cortex-A8 core and IPs. This chip also includes some high-speed IOs like DDR3 Controller, PCIe and RapidIO, and some other low-speed interfaces. All of these components are connected by a three-level bus matrix. This chip is implemented with a TSMC 40-nm low-power process. Figure 10 shows the final layout. The total area is 363.468mm^2.

Figure 11 shows the APE structure. The microcode pipeline in APE runs at 1 GHz an the other components run at 500MHz. There are 10 FUs in each APE, as listed below. Each FU can handle 512 bits of data in the SIMD manner.

- **IALU:** for integer computation, with an SIMD ability for 8-, 16- and 32-bit data types.
- **FALU:** for IEEE 754 single and double precision floating point computation.
- **IMAC:** for integer multiply accumulation,with an SIMD ability for 8-, 16- and 32-bit data types.
- **FMAC:** for IEEE 754 single and double precision floating point multiply accumulation.
- **BIU0,BIU1,BIU2:** for load/store operation. Calculates next data address automatically and supports data access with four dimensions.
- **MReg:** 128x512bit matrix register file with slide window and auto index features.

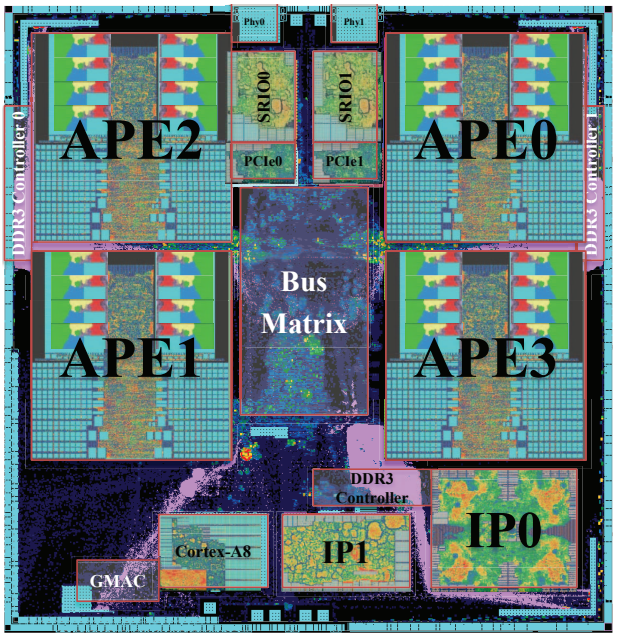


Figure 10: Final layout of the chip

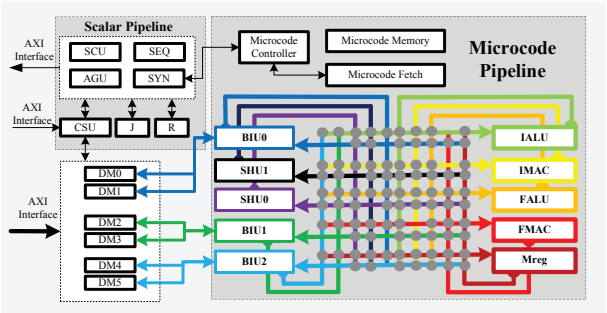


Figure 11: APE structure

- **SHU0, SHU1:** shuffle unit that can extract specific bytes in the source register and write them into the destination register in any order.

Some special function units are designed to explore data locality. Therefore, data flow can be concentrated inside the microcode pipeline and decrease the load/store operations. For example, SHU can perform cascading shift operations, in which two 512-bit registers are connected and shifted for 1, 2 or 4 bytes circularly, just like the sliding window in the finite impulse response (FIR) algorithm. Combined with large matrix register files, the coefficients and input datum need to load only once in the whole FIR process.

The bus, forwarding matrix and memory system are also 512 bits wide. To decrease the scale of the forwarding matrix, not all of the FUs are connected. For example, the FMAC result can not be forwarded to the IALU and IMAC. The microcode syntax embodies these constraints. There are six data memories in total, each of which is an MGP memory system, with a 2M bit capacity. The microcode line includes

14 microcodes, each of which is assigned to an FU except for MReg, which requires 4 microcodes. The microcode line is 328 bits wide, and the microcode memory (MIM) can hold 2,000 microcode lines. To accelerate the turbo decoding process, we add a dedicated turbo co-processor in APE.

The scalar pipeline is a 32-bit VLIW DSP core that contains four FUs :

- **SCU:** for 32-bit integers and IEEE 754 single precision floating point computation.
- **AGU:** for load/store and register file transfer.
- **SYN:** for microcode pipeline configuration and control.
- **SEQ:** for the jump, loop and function call.

These four FUs can run in parallel and the scalar pipeline can issue four instructions at one cycle. Figure 12 shows the final layout of APE. The total area is 36 mm^2.

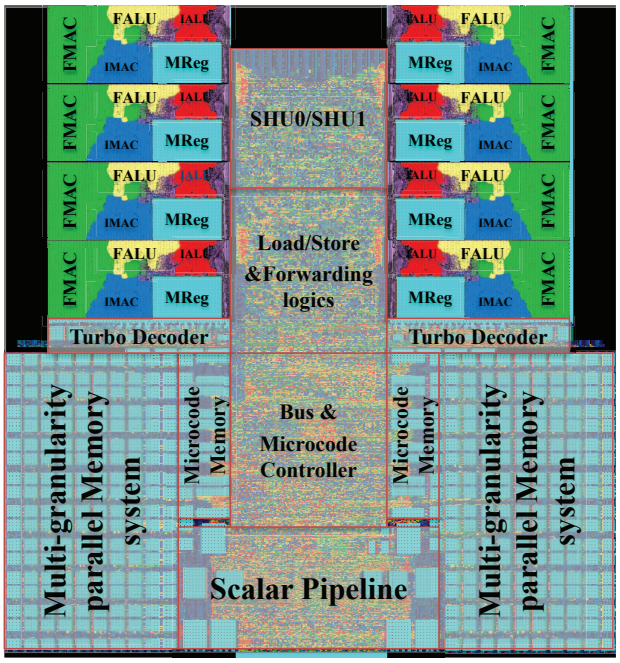


Figure 12: Final layout of APE. FMAC, IMAC, FALU, IALU and MReg are distributed in eight blocks, each of which can handle 64 bits of data. The microcode pipeline runs at 1 GHz an the other components run at 500MHz in typical cases.

The tool chain of APE is based mainly on an open source framework, as shown in Table 3.

4.2 Performance

Before taping out the chip, we simulate many typical signal processing algorithms in APE with the final RTL and compare the performance with that of the TI C66x core, a commercial DSP with a similar process node and computation resource. We suppose that APE runs at 1 GHz and that the C66x core runs at 1.25Hz. We obtain the performance of

Table 3: MaPU tool chains

Tool Name	Open Source Framework
Compiler for microcode pipeline	Ragel & Bison & LLVM
Compiler for scalar pipeline	Clang & LLVM
Assembler/Disassembler	Ragel & Bison & LLVM
Linker	Binutils Gold
Debugger for scalar pipeline	GDB
Simulator	Gem5
Emulator	OpenOCD

the C66x core by running an algorithm in Code Composer Studio (CCSv5) with the official optimized DSP library and image processing library (DSPLIB and IMGLIB).

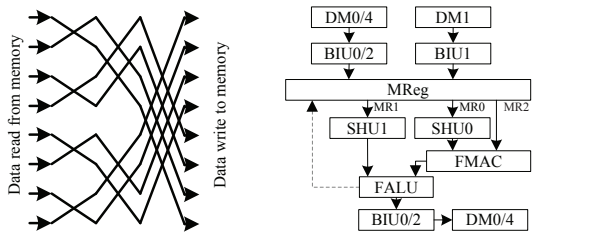
Table 4: Complex SP FFT performance (TimeUnit: us)

length	128	256	512	1024	2048	4096
C66x	0.65	1.18	2.82	5.49	13.09	26.00
APE	0.56	0.88	1.41	2.63	4.75	9.79

Table 4 shows the execution time of a complex single precision floating point FFT algorithm of varying lengths. Table 5 shows the execution time of a complex 16-bit fixed point FFT algorithm. The specific FFT algorithm used here is cached-fft [6]. In this algorithm, butterflies are divided into groups. Each group contains multiple butterflies and stages that can be computed independently without interacting with the data in other groups. Thus, a group of butterflies can be loaded and computed thoroughly without writing back to memory. Figure 13 (a) shows a dataflow diagram of a butterfly group within a complex single floating point FFT. Figure 13 (b) shows the corresponding data path of the FUs. Although the butterfly in a group can be computed independently, the datum must be shuffled between groups in different epochs. This is done naturally using MGP memory. The result of a group is stored back to memory with one G value after computation and then loaded back to the microcode pipeline with a different G value. The loaded data can be computed directly without any shuffle operation. As different pairs of G values are used for the FFT with different data types, the memory access pattern matches the data path perfectly. As result, the overall performance is boosted and the power is reduced. The original work was implemented with a dedicated co-processor. In this paper, the algorithm is re-implemented with only microcodes.

The average speedups of APE vs. the C66x core for the SP FFT algorithm are **2.00x** and **1.89x**, respectively, for a fixed point FFT. In fact, we can further improve the performance of APE through microcode optimization. For example, the execution time for a 4,096-point 16-bit FFT can be reduced from 4.80 us to 4.10 us after further microcode optimization, an improvement of almost 15%. From this example, we can see that MaPU has a huge performance potential with customized state machines.

We implement other typical algorithms with the same FFT strategies, in which the data path resembles the dataflow and MGP memory provides the matched access patterns. Table 6 summarizes the average speedups of APE vs. C66x for these algorithms. These algorithms have different data types and



(a) FFT butterfly diagram, each cross indicates a butterfly

(b) Cascading pipeline for FFT, multiple stages of butterfly are computed before write the result back to memory

Figure 13: FFT dataflow diagram and data path mapping in APE.

Table 5: 16-bit complex fixed point FFT performance (TimeUnit: us)

length	256	512	1,024	2,048	4,096
C66x	0.60	1.33	2.59	5.91	12.03
APE	0.56	0.79	1.50	2.41	4.80

dataflow. However, given the cascading pipeline and intrinsic shuffle abilities of MGP memory, all of them are mapped successfully in APE, and their performance is quite impressive. APE has hundreds of speedups for table lookups due to its SHU units, which can handle 64 parallel queries for a table with 256 records within 5 cycles. Taking the frequency of both processors into consideration, APE has more advantages in terms of architecture as it runs at a lower frequency but performs better.

4.3 Power efficiency

Before taping out the chip, we estimate the power of APE using Prime Time with various algorithms. The switching activity is generated through the final post-simulation with the final netlist and SDF annotation. We also test the power of APE when the chip returns from fab. As in the SoC, there are dedicated power domains and clock gates for AP. As we can turn on the power supply and clock for each APE separately, the power of each APE can be measured precisely by the power increase when it is invoked. Table 7 shows the power data of the typical algorithm. The data types of each algorithm are the same in Table 6.

All of the used micro-benchmarks are held in on-chip memory, but the overall amount of power consumed by memory is small. The number of DMs in APE is designed for scalability. For benchmarks that exceed the size of DM, three

Table 6: APE vs. C66x core: Actual performance comparison

Algorithm	Speedup	Data Types
Cplx SP FFT	**2.00**	Complex single floating point
Cplx FP FFT	**1.89**	Complex 16-bit fixed point
Matrix mul	**4.77**	Real single floating point
2D filter	**6.94**	Real 8-bit fixed point
SP FIR	**6.55**	Real single floating point
Table lookup	**161.00**	table with 8bit address, 256 records
Matrix Trans	**6.29**	16bit matrix

Table 7: Estimated and Tested Power of APE at 1GHz (PowerUnit : Watt)

Algorithm	Est	Tested	Diff	Size
Cplx SP FFT	2.81	2.95	-5%	1,024
Cplx FP FFT	2.63	2.85	-8%	1,024
Matrix Mul	3.05	3.10	-2%	65*66, 66*67 matrix
2D Filter	4.13	4.15	-1%	508*508, 5*5 template
FIR	2.19	2.20	-1%	4,096, 128 coefficients
Table lookup	2.75	2.95	-7%	4,096 queries
Matrix Trans	2.28	2.45	-7%	512*256 matrix
Idle	1.51	1.55	-2%	While APE stand by

of the six DMs can be used for computation buffers, and the other three DMs can be used for DMA transfers. The aggregated power of all of the MGP memories (six DMs in Figure 13) for benchmarks in Table 7 (in order, except for Idle) are 8%, 6%, 3%, 3%, 2%, 3% and 15%, respectively. The 15% is for matrix transpose, which consists entirely of memory read/write operations. Taking DMA transfers into consideration, the energy efficiency will degrade slightly but remain almost the same as the presented result.

We can see clearly from the table that the power consumption of most of the algorithms is below 3 W and that the standby power of APE is as high as 1.55 W. Preliminary analysis indicated that the clock network contributed most of the idle power, which will require improvement in future. It is also can be seen from Table 7 that the estimated and measured power are almost the same. This indicates that our power evaluating method is effective and our module based power analysis discussed later is highly reasonable.

To compute the actual dynamic power efficiency of APE, we collect detailed instructions statistics. Table 8 shows the number of microcodes issued when APE runs different algorithms. The data type and size are the same as in Table 7.

Based on the microcode statistics in Figure 8, we know how many data operations are needed to complete an algorithm. We obtain the *actual* GFLOPS or GOPS of APE for different algorithms by dividing the number of operations with corresponding execution times. The corresponding *actual* GFLOPS/W and GOPS/W are computed by dividing GFLOPS or GOPS with the power, as show in Figures 14 and 15.

The computation operations include IALU, IMAC, FALU, FMAC and SHU0, SHU1(each MAC instruction is considered as two operations) and the total operations includes computation, register file read/write and load/store operations.

Figure 14 shows that the maximum *actual* computation performance of APE for floating point application is 64.33 GFLOPS with the SP FIR algorithm. The maximum *actual* computation performance for the fixed point is 255.21 GOPS with an 8-bit 2D filter application.

When taking power into account, the maximum *actual* total power efficiency of APE for floating point application is 45.69 GFLOPS/W with the SP FFT algorithm. The maximum *actual* total power efficiency across all of the algorithms is 103.49 GOPS/W with a 2D filter application, as shown in Figure 15.

Table 9 summarizes the energy efficiency of several pro-

Table 8: Microcode statistics for different algorithms

Algorithm	MR0	MR1	MR2	MR3	SHU0	SHU1	IALU	IMAC	FALU	FMAC	BIU0	BIU1	BIU2
Cplx SP FFT	1,908	1,841	1,888	12	1,878	1,836	0	0	1,771	1,847	807	746	832
Cplx FP FFT	942	930	897	10	894	894	0	885	0	0	255	193	288
Matrix mul	29,478	0	0	1,650	29,478	0	0	0	12,854	29,476	1,650	488	7,451
2D filter	20,336	20,336	0	0	105,740	105,740	0	105,737	0	0	8,703	20,344	7,599
FIR	2,048	2,049	0	0	34,817	34,817	0	0	1,792	34,817	265	2,048	511
Table lookup	258	192	128	0	257	0	320	0	0	0	4	64	64
Matrix tran	0	0	0	4,098	0	0	0	0	0	0	4,099	0	4,096

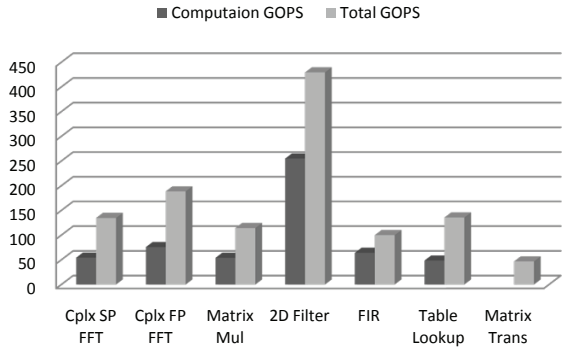


Figure 14: Actual Performance of APE for different algorithms. The units are GFLOPS for floating point application and GOPS for fixed point application.

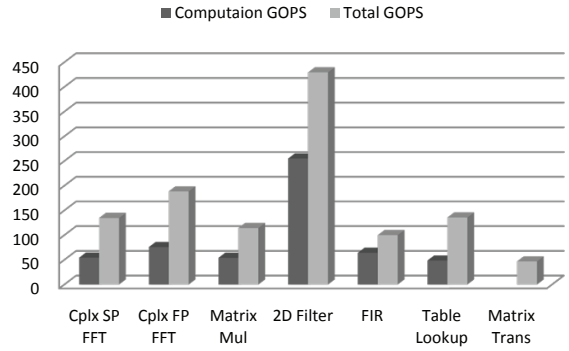


Figure 15: Actual power efficiency tested from real chip. The units are GFLOPS/W for floating point application and GOPS/W for fixed point application.

Table 9: Estimated performance of current processors[7].

Processor	Process	GFLOPS	GLOPS/W
Core i7 960	45nm	96	1.2
Nvidia GTX280	65nm	410	2.6
Cell	65nm SOI	200	5.0
Nvidia GTX480	40nm	940	5.4
Stratix IV FPGA	40nm	200	7.0
TI C66x DSP	40nm	74	7.4
Xeon Phi 7210D	22nm	2,225	8.2
Tesla K40(+CPU)	28nm	3,800	10.0
Tegra K1	28nm	290	26.0
MaPU Core	**40nm**	**134**	**45.7**

cessors, as presented in [7]. From this table and the data presented in [4], we can see that the peak energy efficiency of GPGPU and processors are below 10 GFLOPS/W. The actual maximum energy efficiency of APE is around 40-50 GLFOPS/W, a **4x** to **5x** improvement for the floating point applications and a **10x** improvement for the fixed point applications.

4.4 Discussion

In addition to the instruction statistics in Table 8, we gather the power data for each individual module through detailed simulation. With the microcode count and power, we can estimate the average dynamic energy consumed per microcode, as presented in Table 10. Table 7 shows the estimated and tested power from real chip are very close, thus the power statistics here that based on simulation are highly reliable.

The result is calculated as follows:

$$\frac{(Running\ Power - Idle\ Power) * Time}{Instruction\ Count} \quad (1)$$

The average load/store energy includes the load/store unit, data bus and memory. Table 10 clearly shows that the register file access is mostly energy efficiency. The energy consumed by most of the computation FU is almost half that of the load/store unit, except for IMAC, which requires further improvement.

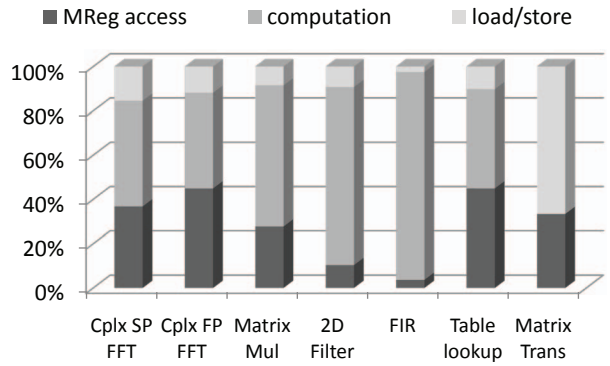


Figure 16: Microcode composition for different algorithms. MReg access includes microcodes for MR0-MR3; computation includes microcodes for SHU, IALU, IMAC, FMAC and FALU; load/store includes microcodes for BIU0-BIU2.

Table 10: Dynamic energy consumed per microcode

Algorithm	Energy Per Microcode (Unit : pJ, 512 bit)
Register R/W	133.25
Load/Store	609.20
FALU	345.65
IALU	335.18
FMAC	387.23
IMAC	788.77
SHU	213.04

As the energy consumed by FUs is much less than that consumed by load/store operations, it is reasonable to keep data moving between FUs through cascading pipelines as much as possible. Figure 16 shows microcode compositions for different algorithms. Using the MGP memory system and cascading pipelines, typical algorithms can be mapped into structured data paths in which the operations mainly comprise register file access and computation operations, which consume much less energy than the load/store operations.

Figure 17 further shows the average usage and average energy consumption for different components when APE runs algorithms in Table 7, except for matrix transpose. A trend can be seen in Figure 17: the register file access operation consumes much less energy than the load/store operations but is used much more, and the computation units are used much more than the file registration and memory units. As such, the energy-efficient FUs are used much more frequently than the energy-inefficient FUs.

This is an important factor that contributes to the remarkable energy efficiency of MaPU. Power efficiency benefits from two aspects. The first aspect is the novel micro architecture. MaPU consists of massive FUs but simple control logic. As in ASIC, most of the energy is consumed by FUs that do the real computations, and energy-hungry operations such as memory access are minimized through MPG memory systems. Therefore, it is possible to achieve high power efficiency in MaPU. In particular, instruction fetching and dispatching logic consume only 0.18% power, and FUs consume up to 53% power in the FFT benchmark.

The second aspect is the elaborately optimized algorithm of MaPU. To achieve outstanding power efficiency, data locality should be explored at the algorithm level and state machines should be constructed in a manner that centralizes data movements in the FUs and forwarding matrix. Figure 17 indicates that benchmarks have been mapped mostly into the computation and register file access operations. As power-consuming load/store operations are only a small part of the overall operations, power consumption is reduced overall.

Great efforts have been made to optimize the micro benchmarks presented in this paper. Although the state-machine-based program model has simplified the optimization process, it would take about one month to implement a kernel on MaPU for those familiar with the micro-architecture. We develop an informal flow to facilitate the optimization process. We would like to explore this flow more thoroughly in the near future and hope to develop a more convenient and high-level program mode based on the one adopted in this paper.

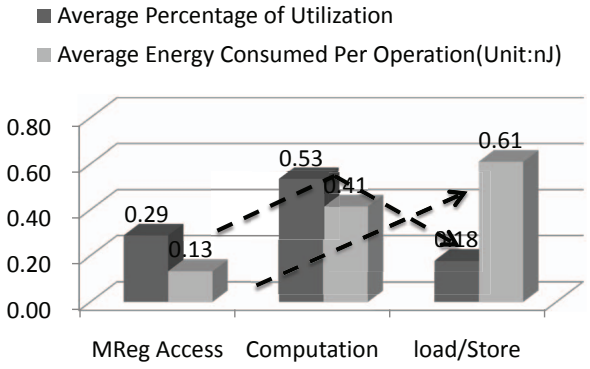


Figure 17: Average usage and energy consumption of different components

5. RELATED WORK

In general, MaPU can be classified into CGRA and vector processors. One principle under MaPU involves mapping the computation graph into a reconfigurable fabric while mapping the data accessing pattern into an MGP memory system. The computation mapping of MaPU is similar to DySER[8]. However, MaPU uses a crossbar-based forwarding logic rather than a switching network. Furthermore, the reconfiguration fabric and computation sub-region in DySER is more like an instruction extension to the scalar pipeline. The reconfiguration fabric and microcode pipeline in MaPU is a standalone processor core that can execute kernel algorithms such as matrix multiply and 2D filter algorithms. MaPU is not like GARP [9]and SGMF[10]. GARP uses fine-grained reconfigurable arrays to construct FUs such as adders and shifters in a way that resembles FPGA. MaPU uses FUs to map high-level algorithms. At the same time, MaPU has no thread concept and thus no data dependence handling logic. This is different from SGMF[10].

The power efficiency of computer architecture has become more and more important in recent years. Voltage and frequency adjustments and clock gating are two main techniques used to decrease the power of previous processors[11]. However, although chips are integrating far more transistors than before, their total power is still strictly constrained. Only a few parts of the chip can be lighted, which leads to the idea of using so-called dark silicon [12]. In this new design regime, heterogeneous architectures have been proposed in which some general-purpose cores are augmented by many other cores and accelerators of different micro-architectures[13]. GreenDroid[14] is such an aggressive dark silicon processor with great power efficiency. Although the dedicated co-processor c-core in this chip can be reconfigured after manufacture, its compatibility with algorithm updates presents a concern for programmers.

Some other customized processors that aim at power efficiency are less aggressive. Most of them focus mainly on improving data path computation, such as by adding a vector processing unit[7][1], adding chained FUs[7] and adding specialized instructions[2][15]. Although improvements have been made through these techniques, they are limited by their memory access efficiency.

Transport triggered architecture possesses many advantages including modularity, flexibility and scalability. Its low-power potential is exploited in [16] but mainly focuses on compiler optimization and the reduction of register file access.

As clock distribution networks consume around 20-50% of the total power in a synchronous circuit[17], an asynchronous circuit is considered an alternative to build low-power processors[18]. However, asynchronous circuits are difficult to design, and related EDA tools are far from mature and may only be successful in special chips such as neuromorphic processors [19].

6. CONCLUSION

The novel MaPU architecture is presented in this paper. With an MGP memory system, cascading pipeline and state-machine-based program model, this architecture possesses great performance and energy potential for computation intensive applications. A chip with four MaPU cores is designed, implemented and taped out following a TSMC 40-nm low-power process. The performance and power of the chip are fully analyzed and compared with other processors. Although the implementation of the first MaPU chip can be further improved, the results indicate that MaPU processors can provide a performance/power efficiency improvement 10 times higher than the traditional CPU and GPGPU.

The first MaPU chip presented here is only an example of the MaPU architecture. Designers can easily implement this architecture in specific domains through customized FUs.

Great efforts are needed in programming MaPU. Although a low-level state-machine-based programming model has been proposed, we hope to develop a more efficient model at a high level. Furthermore, we are also trying to implement a heterogeneous computing framework such as OpenCL on MaPU, which would hide all of the hardware complexity and provide efficient runtimes and libraries designed for specific domains. In addition, we are now working hard to construct relevant wiki pages and make detailed documentation and tools available to the open source community.

7. ACKNOWLEDGEMENT

This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences (under Grant XDA-06010402).

8. REFERENCES

[1] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, no. 2, pp. 56–66, 2015.

[2] L. Codrescu, W. Anderson, S. Venkumanhanti, M. Zeng, E. Plondke, C. Koob, A. Ingle, C. Tabony, and R. Maule, "Hexagon dsp: An architecture optimized for mobile multimedia and communications," *Micro, IEEE*, vol. 34, no. 2, pp. 34–43, 2014.

[3] D. Parello, O. Temam, and J. M. Verdun, "On increasing architecture awareness in program optimizations to bridge the gap between peak and sustained processor performance - matrix-multiply revisited," in *SC Conference*, pp. 31–31, 2002.

[4] A. Sethia, G. Dasika, T. Mudge, and S. Mahlke, "A customized processor for energy efficient scientific computing," *Computers, IEEE Transactions on*, vol. 61, no. 12, pp. 1711–1723, 2012.

[5] V. V. Zyuban and P. M. Kogge, "Inherently lower-power high-performance superscalar architectures," *Computers, IEEE Transactions on*, vol. 50, no. 3, pp. 268–285, 2001.

[6] B. M. Baas, "A low-power, high-performance, 1024-point fft processor," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 3, pp. 380–387, 1999.

[7] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," *Micro, IEEE*, vol. 35, no. 1, pp. 6–14, 2015.

[8] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, "Dynamically specialized datapaths for energy efficient computing," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 503–514, Feb 2011.

[9] J. Hauser and J. Wawrzynek, "Garp: a mips processor with a reconfigurable coprocessor," in *Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, pp. 12–21, Apr 1997.

[10] D. Voitsechov and Y. Etsion, "Single-graph multiple flows: Energy efficient design alternative for gpgpus," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 205–216, June 2014.

[11] S. Kaxiras and M. Martonosi, "Computer architecture techniques for power-efficiency," *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[12] M. B. Taylor, "A landscape of the new dark silicon design regime," *Micro, IEEE*, vol. 33, no. 5, pp. 8–19, 2013.

[13] M. Själander, M. Martonosi, and S. Kaxiras, "Power-efficient computer architectures: Recent advances," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 3, pp. 1–96, 2014.

[14] N. Goulding-Hotta, J. Sampson, S. Swanson, M. B. Taylor, G. Venkatesh, S. Garcia, J. Auricchio, P.-C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. B. Taylor, "The greendroid mobile application processor: An architecture for silicon's dark future," *IEEE Micro*, no. 2, pp. 86–95, 2011.

[15] S. Z. Gilani, N. S. Kim, and M. J. Schulte, "Power-efficient computing for compute-intensive gpgpu applications," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 330–341, IEEE, 2013.

[16] Y. He, D. She, B. Mesman, and H. Corporaal, "Move-pro: a low power and high code density tta architecture," in *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pp. 294–301, IEEE, 2011.

[17] F. H. Asgari and M. Sachdev, "A low-power reduced swing global clocking methodology," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 5, pp. 538–545, 2004.

[18] M. Laurence, "Introduction to octasic asynchronous processor technology," in *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on*, pp. 113–117, IEEE, 2012.

[19] J.-s. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. Tierno, L. Chang, D. S. Modha, and D. J Friedman, "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pp. 1–4, IEEE, 2011.