An 8.6 mW 25 Mvertices/s 400-MFLOPS 800-MOPS 8.91 mm² Multimedia Stream Processor Core for Mobile Applications

Shao-Yi Chien, *Member, IEEE*, You-Ming Tsao, Chin-Hsiang Chang, and Yu-Cheng Lin

Abstract—For the demands of mobile multimedia applications, a stream processor core is designed with 8.91 mm² area in 0.18 μ m CMOS technology at 50 MHz. Several techniques and architectures are proposed to achieve high performance with low power consumption. First of all, an optimized core pipeline is designed with 2-issue VLIW architecture to achieve the processing capability of 400 MFLOPS or 800 MOPS. In addition, adaptive multi-thread scheme can increase the performance by increasing hardware utilization, and the proposed configurable memory array architecture can reduce off-chip memory accessing frequency by caching both input data and output results. Furthermore, for graphics applications, a geometry-content-aware technique called early-rejection-after-transformation is proposed to remove redundant operations for invisible triangles. As for video applications, the proposed video accelerating instruction set can support motion estimation for video coding. Experimental results show that 86% power reduction and more than ten times speedup of the VLIW architecture can be achieved with the proposed techniques to provide the processing speed of 25 Mvertices/s and power consumption of 8.6 mW. Moreover, CIF (352 $\times$ 288) 30 fps video encoding with the search range of {H[−24,24], V[−16,16]} is also supported by the proposed stream processor. By supporting both video and graphics functions, this highly efficient, high performance, and low power processor core is applicable to multimedia mobile devices.

Index Terms—Adaptive multi-thread, configurable memory array, early-rejection-after-transformation, low-power GPU, stream processor, vertex shader.

I. INTRODUCTION

IN RECENT years, the market for mobile electronics devices has grown rapidly, and multimedia functions, including video/audio recording/playing and image processing, have been integrated in mobile devices, such as handsets, PDAs, and portable media players. In these devices, graphics functions are usually required for gaming and graphics user interface (GUI). There is no doubt that more and more graphics processors will be integrated into mobile devices [1], [2]. On the other hand, hardware video coding accelerators are usually

integrated into mobile devices as well. Integrating hardware accelerators with programmable processors or DSPs, and integrating dedicated video encoder and decoder (codec) [3], [4] are the two possible solutions widely used in every multimedia handset.

Although the data type and the data accessing manner are quite different in video coding and graphics, they are both stream applications [5]. That is, in both graphics and video coding functions, the data is formatted as a stream with elements in uniform data structure, which is vertex or pixel in graphics and is macroblock in video coding. Every element is then processed by the same procedure, which is called the kernel function. In recent years, programmable graphics processors have been developed for more flexible and complicated graphics rendering tasks, where the vertex shader is integrated in the geometry stage, and the pixel shader is integrated in the rendering stage. They can be controlled by a special language called shading language [6]–[8]. The shaders of graphics processors can be viewed as one kind of stream processor and have the potential for supporting more applications than graphics [9], [10]. From the system point of the view, if the video coding function can also be integrated into shaders efficiently, it is possible to accelerate both functions in the same processor, which will reduce the total chip area and improve the hardware utilization since these graphics functions and video coding functions are seldom executed simultaneously. It is quite beneficial for mobile applications and is considered in this paper.

Many research works target mobile graphics processors [11]–[16]. These works all focus on core pipeline architecture and circuit-level power reduction. Arakawa *et al.* integrate a floating-point unit (FPU) into a general-purpose processing core and optimize the FPU pipeline [12]. Sohn *et al.* design the first chip supporting the vertex shader model for mobile devices in the literature [14]; however, only fixed-point datapaths are supported, which cannot satisfy the realistic graphic effects of complicated scenes as desktop GPUs do. Vertex cache is adopted inside the core by Yu *et al.* to reduce the power consumption and memory bandwidth [15]. Nam *et al.* optimize the arithmetic unit by use of logarithmic number system and reduce the power consumption with three power-domain dynamic voltage frequency scaling (DVFS) [16].

In this work [17], our target is to design an efficient stream processor for mobile multimedia applications, including graphics applications and video coding applications. The power, performance, and hardware efficiency of this design are optimized from the top algorithm level to the architecture level,

Manuscript received September 10, 2007; revised March 27, 2008. Current version published September 10, 2008. This work was supported by the National Science Council, Republic of China, under Grant NSC96-2221-E-002-252-. Chip fabrication was supported by National Chip Implementation Center (CIC), <http://www.cic.org.tw>.

The authors are with the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan, R.O.C. (e-mail: sychien@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/JSSC.2008.2001898

Fig. 1. Stream processing model.

Fig. 2. System architecture and stream processor architecture.

and down to the circuit level with five key techniques. Adaptive multi-thread (AMT) architecture is designed to increase the performance of the stream processor, and configurable memory array (CMA) architecture is designed to increase the flexibility and utilization of on-chip memory, which can be configured as cache memory, register file, or tightly coupled memory buffer. Then the early-rejection-after-transformation (ERAT) technique, which is a geometry-content-aware technique, is used to remove redundant computation in graphics pipelines, and video accelerating instruction set (VAIS) is proposed for accelerating video coding function. Finally, an optimized core

pipeline (OCP) is designed to reduce power consumption with very-long-instruction-width (VLIW) power-optimized instructions and reconfigurable datapath, which can support integer operations for video coding and floating-point operations for graphics. The organization of this paper is as follows. In Section II, the system architecture of the proposed stream processor is described, and the five proposed architectures and techniques are then shown in Section III. Next, the chip implementation result is shown in Section IV and is compared with previous works. Finally, Section V concludes this paper.

II. SYSTEM ARCHITECTURE

A. Stream Processing Model

Stream processor architecture is efficient for media-processing applications [5], [9]. Since the behavior of media processing can be usually modeled as several loops, where data is loaded sequentially and processed with the same procedure, the concept of stream processing is to separate data accessing unit from the execution unit. The data of media-processing applications is viewed as a “stream” containing a set of elements of the same type and is loaded and stored by some specific hardware units, and each stream element is processed with the same procedure, which is called a “kernel” and is executed in the “kernel execution unit.”

In this paper, we model stream processing as shown in Fig. 1. The media data is viewed as stream data stored in a stream buffer. After input stream data is loaded from the stream buffer and processed by the stream processor core with referring the reference data, the output stream data is generated and stored back to the stream buffer, and the output stream data can be used as the input stream data for some iterative operations. Only a part of the stream data is processed by the stream processor core at one time instance. In this paper, the procedure to process each stream element loaded into the stream processor core is also called a “thread.” Since multiple stream elements are loaded into the stream processor and processed with a time-slicing fashion, it is a multi-thread system. In the stream processor core, there are four kinds of register files around the programmable kernel execution unit: stream input registers, stream output registers, temporary registers, and constant registers. The stream input/output registers are also used as input/output cache memories or input/output buffers for some applications. The temporary registers are used to store some partial results during computation, and the constant registers are used to store the data loaded from the reference data buffer.

Most multimedia data can be modeled as streams. A stream consists of many stream elements, and each stream element has a label number called stream index. The stream processor can identify the stream data by use of the stream index in the index buffer. For example, a graphic polygon object can be treated as a stream data, where each vertex is a stream element. Similarly, for video coding applications, a macroblock, which is usually a 16×16 block, is treated as a stream element and fed into the kernel execution unit. The reference data can be the geometry parameters such as transform matrices or lighting coefficients for graphics applications, and search range data in reference frames for video coding applications.

B. Stream Processor Architecture

Fig. 2 shows the proposed hardware architecture of the stream processor core. It can be integrated into an SoC for mobile multimedia applications with a host processor, peripherals, and a memory controller connected to an off-chip memory, which is used as a stream buffer for the input/output stream data reference data buffer, and a secondary storage for instruction codes.

As shown in Fig. 2, the stream processor core connects to the system bus via a host interface. It is a master and slave interface and can transfer instruction codes into the instruction memory from the off-chip memory. It can also transfer media data from the off-chip memory to the stream input registers and constant registers and transfer media data from the stream output registers to the off-chip memory. Note that, the stream input registers, constant registers, and stream output registers in Fig. 1 are integrated together as the configurable memory array (CMA) shown in Fig. 2, which is a flexible memory unit implemented with on-chip SRAM and can be reconfigured to meet the requirements of different applications. With the cache tag memory, CMA can also be used as the input/output cache to reuse the stream data and decrease the memory bandwidth. The details of CMA are described in Section III-C.

In the stream processor core, the core processing module is the kernel execution unit. It is based on 2-issue VLIW architecture with single-instruction-multiple-data (SIMD) instruction in each slot, where four 32-bit data channels can be processed in parallel. When operated in 50 MHz, it achieves the performance of 400 MFLOPS, where two 4-channel floating-point operations are executed simultaneously. As for video coding applications, video encoding acceleration instruction set [18] is also proposed, where the floating point datapath is reconfigured to support 16-channel 8-bit input and 16-bit output fixed-point operation to achieve 800 MOPS. The details of the kernel execution unit are described in Section III-A.

For graphics applications, a dedicated hardware accelerator, the ERAT module, inspects the geometry contents to prevent redundant operations to save power and increase performance. The details of the ERAT module are described in Section III-D.

III. PROPOSED HARDWARE ARCHITECTURES AND TECHNIQUES

In this section, five new architectures and techniques, Optimized Core Pipeline (OCP), Adaptive Multi-Thread Architecture (AMT), Configurable Memory Array (CMA), Early-Rejection-After-Transformation (ERAT), and Video Accelerating Instruction Set (VAIS), are proposed to optimize the performance, power, and hardware efficiency from the algorithm level, architecture level, and circuit level. OCP integrates both the architecture- and circuit-level optimization techniques. AMT, CMA, and VAIS are the architecture-level techniques, and ERAT is the algorithm-level technique.

A. Optimized Core Pipeline

Several architecture- and circuit-level techniques are adopted in the kernel execution unit to reduce the power consumption. Fig. 3 shows the detailed architecture of the kernel execution unit. The 2-issue VLIW instruction format is shown in

Fig. 3. (a) VLIW instruction format of the proposed stream processor. (b) Optimized core pipeline architecture.

Fig. 3(a), where Modify, Swizzle, and Write Mask features are supported to be fully compatible with the vertex shader model [6], [7]. The associated pipeline architecture is shown in Fig. 3(b). In Fig. 3(a), the operation code (OP) is decoded to active the relative processing elements (PEs) in the execution stage (EXE), where 128-bit datapaths are included, and for most of the SIMD instructions, the 128-bit data is viewed as four 32-bit floating-point channels. Although the four-channel parallel processing datapaths are provided, not all the four channels should be always activated. The active vector (AV) is designed to provide the ability to turn on/off each channel, which is called as instruction-level clock gating. The modify (Mdy) field provides the input source or output result modification datapath. For example, the operation “Destination = (1 – Source0) + (Source1)” only requires one floating-point adder (FPADD) instruction with Mdy code (1 – x). Source 0 (Src0) and Source 1 (Src1) fields specify the source addresses to the relative register files or the data forwarding path as shown in Fig. 3(b). Destination (Dst) field specifies the destination address in register

Fig. 4. Execution stage of the core pipeline.

files. Write mask (WM) field provides the ability if only a part of the execution result needs to be written back to the destination location. Swizzle (Sw) field supports the ability of channel-path swizzling. For the example, the following operation “(Position.x, Position.y, Position.z, Position.w) = (ObjectPosition.x + Offset.w, ObjectPosition.y + Offset.z, ObjectPosition.z + Offset.y, ObjectPosition.w + Offset.x)” only takes one FPADD instruction with Sw code, where (x, y, z, w) denote the four channels of one 128-bit data.

As shown in Fig. 3(b), a four-stage pipeline is employed in the proposed stream processor. 3 KB single port SRAM is used as the instruction memory (IM). The decode (DEC) stage decodes the control signals for each PE and relative pipeline registers, the execution (EXE) stage provides several kinds of PEs to execute different instructions, and the write-back stage (WB) generates the address and control signals for the CMA, temporary registers (TMPREG), and general purpose registers (GPREG) to store back the output results. With the data forwarding path, which can be selected by the fields Src0 and Src1, the software data forwarding mechanism gives compilers the flexibility of forwarding the intermediate data to the EXE stage for the next instruction in the pipeline and reduces the complex hardware forwarding logic. Furthermore, with the selective write back (SWB) mechanism controlled by use of WM, the compiler can control which channels of the result need to be written back into registers. It can reduce the data accessing frequency of register files to reduce the power consumption when co-operating with software data forwarding. To further reduce the power consumption, the clock gating cells are also inserted in the pipeline as shown on the top of Fig. 3(b). The OP, AV, Dst and WM fields are used to generate the relative clock enable signals to activate the clock gating cells to provide the instruction level clock gating, where the clock signals of the unused registers can be gated.

Fig. 5. Detailed circuits of (a) floating-point adder (FPADD), (b) SIMD floating-point adder datapath, and (c) adder tree datapath.

The PEs of the execution stage (EXE) are shown in Fig. 4, and the detailed circuits of some PE datapaths are shown in Fig. 5. As shown in Fig. 4, four multiplexors are used to select data from CMA, TMPREG, GPREG, and Data Forwarding Path. The bitwidth of the datapath is 128-bit, and four-channel SIMD floating-point adder (FPADD), four-channel SIMD floating-point multiplier (FPMUL), unified special operations (SOP), adder tree, summation (SUM), and fixed-point ALU

Fig. 6. Illustrations of (a) single-thread, (b) conventional multi-thread, and (c) the proposed adaptive multi-thread architectures.

PEs are included. Note that, in order to clarify the presentation, the interconnections in the datapath are neglected in Fig. 4. The detailed circuits of FPADD are shown in Fig. 5(a) [19], the circuits of SIMD FPADD are shown in Fig. 5(b), and SIMD FPMUL is designed in a similar way. Note that, as shown in Fig. 5(a), the significant adder of the one-channel floating-point adder, which is marked with gray color, can be reconfigured to four 8-bit or two 16-bit fixed-point adders to share the hardware resource. “Type” is used to switch between floating-point and fixed-point modes, and “Mode” is used to switch 8-bit and 16-bit modes. Therefore, when working at 50 MHz, the capacity of 400 MFOPS or 800 MOPS can be achieved by the proposed architecture. SUM datapath supports the floating-point summation (FPSUM) instruction to sum up all the four 32-bit floating-point channels. With the FPMUL and FPSUM, the macro instruction FPD4 (floating-point dot product for two 4-tuple vectors) can be executed in two cycles, and the vertex transformation operation can be pipelined in four cycles. The unified special operation (SOP) PE handles the reciprocal (RCP), square root (RSQ), exponential (EXP), and logarithm (LOG) operations with a shared unified datapath. The scalar arithmetic logic unit (ALU) PE handles the scalar flow control and logic operation instructions. Moreover, the modify (Mdy) PE provides the input/output modification datapath as

described above. Finally, the adder tree datapath, which is shown Fig. 5(c), is used to sum up 16 8-bit fixed-point data and store into an accumulation register to support video coding. The details are described in Section III-E.

B. Adaptive Multi-Thread Architecture

As mentioned in Section II, a thread is the procedure to process one stream element and is composed of many instructions. Due to the stream processing nature, many homogenous threads can be attended. This characteristic makes the multi-threaded stream architecture much more efficient and require less hardware overhead than general purpose RISC. In this section, cooperating with data forwarding mentioned in the previous subsection, the adaptive multi-thread (AMT) architecture is proposed to increase the performance by increasing the utilization of the PEs. Fig. 6(a) shows an example of the conventional single-thread scheme with data forwarding. In this example, “Vn” denotes a input thread with a thread number “n.” Each thread has three instructions: Instruction 0 (Inst0), Inst1, and Inst2, which have data dependency. That is, Inst1 needs the result from executing Inst0, and Inst2 needs the result from executing Inst1. Inst0 and Inst2 are normal instructions with one cycle latency. Inst1 represents a special instruction with long latency, whose latency is assumed as six cycle here.

Note that, each thread is for one stream element, which can be a vertex or a macroblock, and the instructions for different threads are identical. As shown Fig. 6(a), for normal instructions, the pipeline can operate fluently using data forwarding. For the long-latency instructions, such as the texture loading (TxLoad) instruction [6], [7], single-thread with data forwarding introduces pipeline stalls (NOP in the figure) because of data hazard, and the system performance is degraded. Fig. 6(b) shows the other example with the conventional multi-thread scheme. We assume that there are four on-chip threads. In this scheme, different threads are executed in interleaving manner to hide execution latency to mitigate performance degradation. However, the pipeline still suffers from stall penalty when the number of latency cycles is larger than the thread number. For an optimized multi-thread pipeline, it should satisfy the following equation: $\#(\text{Latency Cycles}) \leq \#(\text{On-Chip Threads}) - 1$. In this example, at most three latency cycles can be hidden. The developed AMT technique efficiently uses the minimum threads to provide the maximum hidden latency cycles. AMT combines both the data forwarding and multi-thread schemes. The thread controller inspects the current instruction state to adaptively determine if it should switch the thread or just forward the partial data. As shown in Fig. 6(c), with considering data forwarding, when the current instruction can be executed without introducing any data hazard condition, the controller would not change the thread, such as the Inst0 for each thread; otherwise, the controller would change the thread to hide the execution latency. In the example shown in Fig. 6(c), AMT executes fluently without any pipeline stall. The optimized AMT pipeline satisfies the following equation, $\#(\text{Latency Cycles}) \leq [\#(\text{On-Chip Threads}) - 1] \times \#(\text{Consecutive Instructions})$, where “consecutive instructions” are the instructions that can be executed consecutively without introducing any pipeline stalls before the long-latency instructions. In this example, $\#(\text{Consecutive Instructions}) = 2$, and at most six latency cycles can be hidden, which is much longer than that of conventional multi-thread architecture. Because the AMT can efficiently hide the instruction latency and pipeline the threads, the processing time can be reduced, and thus the power consumption can be saved.

Several circuits are designed to support AMT in the proposed stream processor. The data forwarding path is designed as shown in Fig. 3 and Fig. 4, which can be controlled by instructions. An adaptive multi-thread controller is integrated in the control module in Fig. 3. The adaptive multi-thread technique employs double thread context FIFOs to maintain the thread processing order. In thread fetching stage, the thread in the foreground context FIFO is fetched and fed into the pipeline. When the AMT decoder determines that it is required to switch to the other thread, the current thread context is popped out from the foreground context FIFO and pushed into the background context FIFO. Once the foreground context FIFO is empty, the thread fetching logic resumes the threads in the background context FIFO again by swapping the context FIFOs.

C. Configurable Memory Array

Memory bandwidth is an important design factor in power-limited mobile devices, and two buffering mechanisms

Fig. 7. Architecture of CMA.

are usually employed to reduce the memory bandwidth. For a dynamic data accessing pattern, cache memory can be used to reduce the off-chip memory bandwidth and the accessing latency. The vertex stream in graphics pipeline inherently introduces high cache hit rate because the adjacent triangles usually share the same vertices. On the other hand, for a static data accessing pattern, tightly coupled memory could outperform cache memory in reducing the bandwidth since the optimal data updating mechanism could be pre-determined, where fixed cache updating rule may lead to data miss-replacing. The search range data in reference frames of motion estimation (ME), which is the most computationally intensive operation in video coding [20], is the best candidate to take advantage of the tightly coupled memory.

Configurable memory array (CMA) architecture is proposed to support both the two mechanisms to decrease the memory bandwidth for different applications. CMA with 4-channel and 8-bank accessing ability serves as a physical on-chip memory pool. The data organization of the on-chip memory pool can be configured for different applications to achieve high memory utilization, and the software driver can reconfigure the CMA in frame base to achieve the optimized configuration. The architecture of CMA is shown in Fig. 7. Among the four logic channels, three channels can be accessed by the kernel execution unit as the stream input, stream output, and constant registers

Fig. 8. Different configurations of CMA.

in the stream processing model shown in Fig. 1. The other one channel is reserved for loading input stream and storing output stream from/to the stream buffer. With the separated design of the internal stream channels and external stream channels, high utilization of PEs and memory bandwidth can be achieved. The address decoders decode the logical addresses into physical SRAM addresses with the CMA configuration set by the programmer. The read data is dispatched and assembled before being sent to the channels, and the write data is dispatched and assembled after being received from the channels. With the multi-bank architecture, it provides the ability to be reconfigured into different modes, and only the banks accessed by channels are needed to be activated to reduce power consumption. Each SRAM bank provides 128-bit bandwidth, which satisfies the four floating-point channels for graphic processing and sixteen 8-bit fixed-point channels for video processing. With CMA, the size of stream input, stream output, and constant registers can be adjusted dynamically by the programmers as shown in Fig. 8. When there is no constant data to be loaded, the programmer can configure the whole CMA into stream input and output registers.

With CMA bank configuration, the address generation unit generates the physical addresses from the source and destination addresses in the instructions. There are two memory address mapping modes in CMA. The width of the physical address (PA) is 12-bit for addressing the whole eight SRAM banks, where each bank has 32 words, and each word has 128 bits (16 bytes). In the linear mode, the physical bank is encoded in bit11-bit9 in PA, the word address is encoded in bit8-bit4 in PA, and the byte address is not used in this mode. In the interleaving mode, the physical bank is encoded in bit6-bit4 in PA, the word address is encoded in bit11-bit7 in PA, and the byte address is fed into the byte alignment assembler shown in Fig. 7 to enable the

Fig. 9. Byte alignment assembler.

byte accessing ability as shown in Fig. 9. With the interleaving mode and byte alignment assembler, it gives the accessing flexibility for stream data or constant reference data in the CMA. It is very useful for video coding, and the details are described in Section III-E.

D. Early Rejection After Transformation (ERAT)

Due to the limited computing power of the host CPU in mobile devices, for real-time graphic applications, visibility testing is difficult to be performed in software application stage since some testing methods can only be employed after transformation stage, but the invisible triangles passing through the pipeline usually lead to redundant computation and power consumption for lighting and other following operations. In average, there are about 35% redundant triangles in a dynamic 3-D scene [18]. In this paper, the geometry-content-aware technique called ERAT [18] is developed to reduce power consumption and increase the performance by rejecting redundant triangles in the frontmost stage, the transformation stage, of a graphic pipeline. A dedicated module, ERAT, is designed to detect and reject the redundant triangles as shown in Fig. 2 and Fig. 10. The cache tag records the state information for each thread (vertex) in CMA, and the detailed data structure of the cache tag is shown in Fig. 10, where the tag “Trans” denotes if the transformation operation of the vertex is done by the kernel executing unit, the tag “Lighted” denotes if the lighting operation of the vertex is done, and the tag “valid” denotes if the vertex is valid for further processing. After the three vertices of a triangle are all transformed, the ERAT module inspects the triangle to see if it belongs to the invisible triangles of the cases shown in Fig. 11. If so, those invisible triangles are removed from the pipeline by resetting the valid tags of the associated vertices to avoid redundant computation. With ERAT technique, the redundant triangles can be rejected before lighting, which cannot be supported in other vertex shader architectures.

Note that, with the cache tag, CMA can act as a cache memory. The Index tags are compared with the current input

Fig. 10. Cache tag is modified by ERAT module to avoid redundant computation.

Fig. 11. Early rejected triangle types.

thread index to decide the cache hit/miss. Once the cache is hit, the memory accessing of loading the stream element from the external stream data buffer can be avoided, which can be called as pre-transformation-and-lighting (Pre-TnL) cache. Furthermore, with Lighted tag, the lighting result of the hit vertex can be reused in the CMA, like the Post-TnL cache. For the case of a triangle strip object, the 66% cache hit rate can be achieved, and the vertex throughput can be at least doubled.

E. Video Accelerating Instruction Set

There are two main challenges for general purpose processors in video applications. The first part is the huge data flow operations due to the massive data loading/storing between the processing kernel and external memory. The second one is the huge arithmetic operations where parallel processing is usually required. In the proposed stream processor, the first part can be solved by use of stream processing model and CMA architecture as described in the previous subsections. As for the second part, Video Accelerating Instruction Set (VAIS) is proposed to reuse the existing hardware resource to achieve highly parallel processing. Since motion estimation (ME) is the most computationally intensive operation in video coding [20], we take it as an example, and full search block matching algorithm (FSBMA)

Fig. 12. Configuration of CMA for motion estimation.

is considered. For each block of the current frame, the corresponding block, or the best matched block, in the previous frame can be found by the following equations:

$$\text{SAD}(m, n) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |c(i, j) - p(i + m, j + n)| \quad (1)$$

$$\text{MV} = \{(u, v) | \text{SAD}(u, v) \leq \text{SAD}(m, n) \quad \forall (m, n) \in \text{Search Range}\} \quad (2)$$

where $c(\cdot)$ is a block of the current frame, $p(\cdot)$ is the search range of the previous frame, SAD is sum of absolute difference, which is the most popular matching criterion, N is the block size, whose typical value is 16, and MV is motion vector, which is searched in a pre-defined search range to find the best matched one with smallest SAD. For efficient data accessing, CMA is reconfigured as shown in Fig. 12, where the current macroblock is pre-loaded into TMPREG, and the search range data is loaded into CMA as constant registers. As described in Section III-C, interleaving mode is employed to match the current block to any positions in the search range, as shown in Fig. 12. It can support at least $\{H[-24, 24], V[-16, 16]\}$ search range, and level C data reuse scheme is performed to reduce the off-chip memory bandwidth [21].

On the other hand, as described above, the massive loop, sum and absolute difference take a huge amount of instructions for a RISC-type machine. VAIS includes application specific instructions such as SAD, LOOP, etc. [18] to accelerate motion estimation operation. The SAD instruction can calculate the SAD of a row of pixels (16 pixels) in the macroblock. It is a macro instruction and is implemented with FPADD and Adder Tree datapaths described in Section III-A in two cycles. As shown in Fig. 13, the full adders inside the four FPADD PEs are reconfigured to support 16-parallel 8-bit absolute difference, which is shown as D_0 to D_{15} in Fig. 13. With data forwarding path, the results of absolute difference are summed up by the adder tree. The LOOP instruction provides a specific loop register with auto-increment and end-condition checking. With the 2-issuing VLIW pipeline, the LOOP and SAD instructions can be issued simultaneously, a whole 16×16 macroblock SAD only takes 16 cycles in average, which is much faster than traditional general purpose processors. Moreover, for speeding up ME, several fast

Fig. 13. Datapath for motion estimation operation.

algorithms have been proposed. Among them, partial distortion elimination (PDE) [22] algorithm is also supported in the proposed processor. It requires a little overhead of hardware cost but gains good performance without any quality lost. The PDE PE inspects the intermediate SAD value. If it is larger than the current minimum SAD, the current candidate search position is impossible to the best one and can be skipped.

IV. IMPLEMENTATION RESULTS AND COMPARISON

A prototype chip for the proposed stream processor is fabricated by TSMC with 0.18 μm 1P6M process. The chip specifications are shown in Table I, and the chip micrograph is shown in Fig. 14. With the area of 8.91 mm² at working frequency of 50 MHz, the performance of 400 MFLOPS/800 MOPS can be achieved with only 8.6 mW in power consumption. The processing speed of 25 Mvertices/s is achieved for graphics applications, and it can support motion estimation for 30 CIF (352 $\times$ 288) frames per second with search range $\{H[-24,24], V[-16,16]\}$ and the power consumption of 21.6 mW.

Fig. 15 shows the effects with the proposed techniques. A real graphic application including transformation and specular lighting is used as a test pattern. The power measurement results are shown in Fig. 15(a). With all the proposed CMA, AMT, and ERAT techniques described above, 86% power saving can be achieved. When CMA is employed with cache tag as a cache memory, it can provide high hit rate for duplicate vertices, and the external memory bandwidth and power consumption can be greatly reduced. AMT uses data forwarding to reduce SRAM accessing frequency to reduce power consumption. Finally, when ERAT technique is applied, averagely 35% redundant triangles can be removed in the early stage of the whole

TABLE I
CHIP SPECIFICATIONS

Technology	TSMC 0.18 μm CMOS 1P6M
Die Size	8.91mm ²
Supply Voltage	1.8V
Clock Frequency	50MHz
Power Consumption	8.6mW for graphics* ¹ 21.6mW for video coding* ²
On-chip SRAM	IM: 3KB CMA: 4KB TMP REG: 512B
Performance	400MFLOPS* ³ /800MOPS* ⁴ —Graphics 25Mvertices/s* ⁵ —Video Coding 30 CIF (352 $\times$ 288) frames per second with search range $\{H[-24,24], V[-16,16]\}$

*1) Measured in the case of specular light effect with 20 instructions

*2) 30 CIF (352 $\times$ 288) frames per second

*3) Peak floating-point performance

*4) Measured in the case of integer motion estimation

*5) Peak vertex throughput with post-TnL cache hit rate of 50%

Fig. 14. Chip micrograph of the proposed stream processor.

graphics system, which can reduce the power consumption down to 8.6 mW. In Fig. 15(b), the performance increasing with the proposed techniques is shown as well, where more than ten times improvement can be achieved than that of OCP. When CMA is applied, it not only reduces the bandwidth in redundant vertices but also reuse the results generated by kernel execution unit, which saves the extra processing time and improves the throughput. With AMT technique, multi-threading is employed for long-latency instructions to reduce the pipeline stall. The performance is also improved when the pipeline stall hazard was removed. Finally, ERAT technique provides the dynamic performance improvement as shown in the gray part in Fig. 15(b), and the optimal performance of 25 Mvertices/s can be achieved when the rejection rate is very high, where most of the lighting operations are removed, and only transformation operations are executed.

Finally, Table II shows the comparison between the prior arts and our work. Under the optimal situation, the power efficiency of the proposed stream processor is the best in terms of Mvertices/(s $\cdot$ mW). Moreover, the proposed stream processor can support both graphics applications and video coding

Fig. 15. (a) Power reduction performance of the proposed techniques. (b) Performance increasing with the proposed techniques.

TABLE II
COMPARISON TO EXISTING WORKS

	ISSCC'04 [12]	ISSCC'05 [14]	ISSCC'06 [15]	ISSCC'07 [16]	This Work
Process (nm)	130	180	180	180	180
Clock Frequency (MHz)	400	200	100	200	50
Polygon Rate* ¹ (Mvertices/s)	36	50	120	141	25* ⁵ /12.5* ⁶
Power Consumption (mW)	250	155* ³	157	86	8.6
Power Efficiency (Mvertices/(s·mW))	0.14	0.32* ³	0.76	1.64	2.91* ⁵
Area (mm ²)	-	36* ³	25	9.7* ⁴	8.9
Feature	Graphics, DSP	Graphics* ²	Graphics	Graphics	Graphics, DSP

*1) Vertex transformation throughput

*2) Fixed-point arithmetic pipeline

*3) Include rendering engine

*4) Core size

*5) With post-TnL cache hit rate of 50%

*6) With cache hit rate of 0%

applications with a unified architecture, which can dramatically reduce the cost of the SoC for mobile multimedia applications.

V. CONCLUSION

A stream processor core is proposed in this paper for mobile multimedia applications. Five architectures and techniques, optimized core pipeline (OCP), adaptive multi-thread (AMT), configurable memory array (CMA), early rejection after transformation (ERAT), and video acceleration instruction set (VAIS), are proposed to increase performance and reduce power consumption in algorithm, architecture, and circuit levels. From the experimental results, these techniques are successfully proved. It shows that 86% power consumption of the OCP architecture can be reduced with AMT, CMA, and ERAT techniques to achieve the power consumption of 8.6 mW. On the other hand, more than ten times speedup of the OCP architecture can be also achieved with AMT, CMA, and ERAT techniques to provide the optimal processing capability of 25 Mvertices/s for graphics applications. Moreover, with VAIS, the proposed

stream not only supports graphics applications but also supports full-search motion estimation (ME) for CIF (352×288) 30 fps video coding. The fabricated prototype chip shows that the area is only 8.91 mm² to satisfy both the demands of the graphic and video processing for mobile multimedia applications.

Many commercial vertex shaders have the feature of texture accessing to support some special graphics effects, such as bump mapping. It is also considered in our stream processor core but is not implemented and integrated in this work. If the texture feature is integrated, the proposed stream processor core can also support the functions of pixel shaders or unified shaders to cover most of the functions of a graphics processing unit (GPU), which will be included in our future works.

REFERENCES

- [1] M. Kameyama, Y. Kato, H. Fujimoto, H. Negishi, Y. Kodama, Y. Inoue, and H. Kawai, "3D graphics LSI core for mobile phone Z3D," in *Proc. Graphics Hardware '03*, 2003, pp. 60–67.
- [2] T. Akenine-Möller and J. Ström, "Graphics for the masses: A hardware rasterization architecture for mobile phones," *ACM Trans. Graphics (SIGGRAPH'03)*, vol. 22, no. 3, pp. 801–808, Aug. 2003.

- [3] T. Nishikawa, M. Takahashi, M. Hamada, T. Takayanagi, H. Arakida, N. Machida, H. Yamamoto, T. Fujiyoshi, Y. Maisumoto, O. Yamagishi, T. Samata, A. Asano, T. Terazawa, K. Ohmori, J. Shirakura, Y. Watanabe, H. Nakamura, S. Minami, T. Kuroda, and T. Furuyama, "A 60 MHz 240 mW MPEG-4 video-phone LSI with 16 Mb embedded DRAM," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2000, pp. 230–231.
- [4] H. Arakida, M. Takahashi, Y. Tsuboi, T. Nishikawa, H. Yamamoto, T. Fujiyoshi, Y. Kitasho, Y. Ueda, M. Watanabe, T. Fujita, T. Terazawa, K. Ohmori, M. Koana, H. Nakamura, E. Watanabe, H. Ando, T. Aikawa, and T. Furuyama, "A 160 mW, 80 nA standby, MPEG-4 audiovisual LSI with 16 Mb embedded DRAM and a 5 GOPS adaptive post filter," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2003, pp. 1–11.
- [5] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors," *Computer*, vol. 36, no. 8, pp. 54–62, Aug. 2003.
- [6] "DirectX 9.0 SDK," Microsoft, 2003.
- [7] A. Munshi, OpenGL ES Common Profile Specification 2.0. Mar. 2007 [Online]. Available: http://www.khronos.org/opengles/2_X/opengles_spec_2_0.pdf
- [8] R. J. Rost, *OpenGL Shading Language*. Reading, MA: Addison-Wesley, 2004.
- [9] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. Graphics (SIGGRAPH'04)*, vol. 23, no. 3, pp. 777–786, Aug. 2004.
- [10] C. J. Thompson, S. Hahn, and M. Oskin, "Using modern graphics architectures for general-purpose computing: A framework and analysis," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO-35)*, 2002, pp. 306–317.
- [11] R. Woo, S. Choi, J.-H. Sohn, S.-J. Song, Y.-D. Bae, C.-W. Yoon, B.-G. Nam, J.-H. Woo, S.-E. Kim, I.-C. Park, S. Shin, K.-D. Yoo, J.-Y. Chung, and H.-J. Yoo, "A 210 mW graphics LSI implementing full 3-D pipeline with 264 Mtexels/s texturing for mobile multimedia applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2003, pp. 44, 476.
- [12] F. Arakawa, T. Yoshinaga, T. Hayashi, Y. Kiyoshige, T. Okada, M. Nishibori, T. Hiraoka, M. Ozawa, T. Kodama, T. Irita, T. Kamei, M. Ishikawa, Y. Nitta, O. Nishii, and T. Hattori, "An embedded processor core for consumer applications with 2.8 GFLOPS and 36 M Polygons/s FPU," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2004, pp. 334, 531.
- [13] J.-H. Sohn, R. Woo, and H.-J. Yoo, "A programmable vertex shader with fixed-point SIMD datapath for low power wireless applications," in *Proc. Graphics Hardware '04*, 2004, pp. 107–114.
- [14] J.-H. Sohn, J.-H. Woo, M.-W. Lee, H.-J. Kim, R. Woo, and H.-J. Yoo, "A 50 Mvertices/s graphics processor with fixed-point programmable vertex shader for mobile applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2005, pp. 192, 592.
- [15] C.-H. Yu, K. Chung, D. Kim, and L.-S. Kim, "A 120 Mvertices/sec multi-threaded VLIW vertex processor for mobile multimedia applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2006, pp. 1606–1615.
- [16] B.-G. Nam, J. Lee, K. Kim, S. J. Lee, and H.-J. Yoo, "A 52.4 mW 3-D graphics processor with 141 Mvertices/s vertex shader and 3 power domains of dynamic voltage and frequency scaling," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2007, pp. 278, 603.
- [17] Y.-M. Tsao, C.-H. Chang, Y.-C. Lin, S.-Y. Chien, and L.-G. Chen, "An 8.6 mW 12.5 Mvertices/s 800 MOPS 8.91 mm² stream processor core for mobile graphics and video applications," in *Symp. VLSI Circuits Dig. Tech. Papers*, 2007, pp. 218–219.
- [18] Y.-M. Tsao, S.-Y. Chien, C.-H. Chang, C.-J. Lian, and L.-G. Chen, "Low power programmable shader with efficient graphics and video acceleration capabilities for mobile multimedia applications," in *Int. Conf. Consumer Electronics Dig. Tech. Papers (ICCE'06)*, 2006, pp. 395–396.
- [19] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford, U.K.: Oxford Univ. Press, Inc., 2000.

- [20] S.-Y. Chien, Y.-W. Huang, C.-Y. Chen, H. H. Chen, and L.-G. Chen, "Hardware architecture design of video compression for multimedia communication systems," *IEEE Commun. Mag.*, vol. 43, no. 8, pp. 122–131, Aug. 2005.
- [21] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 1, pp. 61–72, Jan. 2002.
- [22] ITU-T Recommendation H.263 Software Implementation. Digital Video Coding Group, Telenor R&D, 1995.

Shao-Yi Chien (S'99–M'04) received the B.S. and Ph.D. degrees from the Department of Electrical Engineering, National Taiwan University (NTU), Taipei, Taiwan, R.O.C., in 1999 and 2003, respectively.

During 2003–2004, he was a research staff member with Quanta Research Institute, Tao Yuan Shien, Taiwan. In 2004, he joined the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, as an Assistant Professor. His research interests

include video segmentation algorithm, intelligent video coding technology, image processing, computer graphics, and associated VLSI architectures.

You-Ming Tsao received the B.S. degree in electrical engineering from National Central University, Taiwan, in 1999 and the M.S. degree in electrical engineering from National Taiwan University in 2001.

He joined SiS Corp. in 2002, where he developed graphic processor unit. He currently is pursuing the Ph.D. degree at the Graduate Institute of Electronics Engineering, National Taiwan University. His research interests include video coding algorithm, computer graphics system, and associated VLSI

architectures.

Chin-Hsiang Chang received the B.S. degree from the Department of Electronics Engineering and the M.S. degree from the Graduate Institute of Electronics Engineering (GIEE), National Taiwan University (NTU), Taipei, Taiwan, R.O.C., in 2004 and 2006, respectively.

During 2004–2006, he was with Media IC and System Lab, GIEE, NTU, where his research interests included 3-D computer graphics and associated VLSI architecture and implementation. In 2007, he joined MediaTek Inc., HsinChu, Taiwan, as an

engineer.

Yu-Cheng Lin received the B.S. degree from the Department of Electrical Engineering, National Taiwan University (NTU), Taipei, Taiwan, R.O.C., and the M.S. degree from the Graduate Institute of Electronics Engineering (GIEE) from National Taiwan University (NTU), Taipei, in 2005 and 2007, respectively.

During 2005–2007, he was with Media IC and System Lab, GIEE, NTU, where his research interests included computer graphics, video coding technology, and VLSI implementation.