

# Critical Review of Programmable Mediaprocessor Architectures

Stefan G. Berg, Weiyun Sun, Donglok Kim and Yongmin Kim

Image Computing Systems Laboratory  
University of Washington  
Box 352500  
Seattle, WA 98195-2500

## ABSTRACT

In the past several years, there has been a surge of new programmable mediaprocessors introduced to provide an alternative solution to ASICs and dedicated hardware circuitries in the multimedia PC and embedded consumer electronics markets. These processors attempt to combine the programmability of multimedia-enhanced general purpose processors with the performance and low cost of dedicated hardware. We have reviewed five current multimedia architectures and evaluated their strengths and weaknesses.

**Keywords:** multimedia, processors, mediaprocessors, programmable, memory system

## 1. INTRODUCTION

Programmable mediaprocessors are regarded as multimedia DSPs that are targeted to simultaneously handle multiple data types including digital video, image, audio, computer graphics and text.<sup>1</sup> While modern general purpose processors have adopted a number of DSP feature (e.g., SIMD and multimedia instructions), mediaprocessors have incorporated some architectural features that have been available in general purpose processors (e.g., register based, highly pipelined, load/store architecture). Parallelism in most multimedia algorithms is easily visible to programmers and compilers since usually the same computation is performed on every incoming data item (e.g., pixel of an image) and in many algorithms very little data dependence exists between the computation of different output values. For this reason, most mediaprocessors support vector processing in the form of Single Instruction Multiple Data (SIMD) instructions and they aggressively try to exploit this with data paths as wide as 128 bits that can process, for example, eight 16-bit data in a single instruction. While modern general purpose processors offer this too, mediaprocessors contain a large number of functional units dedicated to SIMD instructions and other specialized tasks including input and output.

In achieving this instruction-level parallelism (ILP) efficiently, most mediaprocessors are organized as Very Long Instruction Word (VLIW) architectures.<sup>2</sup> This allows for greater flexibility in instruction scheduling than would be possible with a SIMD architecture alone. Therefore, register files are not only wide, but, are also highly multi-ported as in superscalar design to be able to feed multiple functional units in the same cycle. To avoid the overhead of excessive multi-porting in register file design, some mediaprocessors split the register file into multiple smaller ones, each to be shared by a smaller number of functional units.<sup>3</sup>

Another difference between general purpose and mediaprocessors is that the former are typically superscalar and the latter are VLIW. While both styles have the same goal of exploiting ILP, VLIW architectures scale better because they do not have the complexity of dynamically scheduling sequential instructions onto the on-chip functional units.<sup>4</sup> The downside is that mediaprocessors must often be programmed in assembly and do not offer backward compatibility to older processors. Compilers are available for most architectures, the difficulty of efficiently scheduling for a VLIW, exploiting SIMD instructions and controlling DMA engines and fixed-function units has been the main challenge.

In addition to or instead of caches, mediaprocessors also contain addressable memories and DMA engines in their memory hierarchy due to several reasons. First, there is a higher level of parallelism available in mediaprocessors executing multimedia algorithms resulting in more computations per unit time compared to general purpose processors. Second, the streaming data nature of multimedia algorithms generally leads to a larger bandwidth requirement compared to general purpose algorithms that tend to use loaded data for a longer period of time. Combined, these points add up to high bandwidth requirements for mediaprocessors. However, the predictable nature of many multimedia streams (e.g., sequential, 2D, strided) opens up new possibilities for dealing with them.

	TM-1000	MAP1000	Mpact 2	TMS320C6x	TMS320C80
Base Architecture	VLIW	VLIW	VLIW	VLIW	MIMD
Function Units	27	12	6	24	33
Issues per Cycle	5	4	6	8	5
Data Path Width (bits)	32	64 (128)	72	32-40	32
Clock Rate (MHz)	100	200	125	200	60
Release Date	1Q1996	1999	3Q1997	3Q1997	1994
Technology [ $\mu\text{m}$ ]	0.5	0.35	0.35	0.25	0.5
Cost [US\$/10k]	< 50	n.a. <sup>1</sup>	30-36	90-96	300-400
Peak 8/9-bit BOPS <sup>2</sup>	est. 2-4	est. 20.4	est. 6	est. 3.2	est. 2.4
Instr. Fetch Width (bits)	16-244	16-142	n.a.	32-256	4×64
Pipeline Depth	est. 20	18	n.a.	16	n.a.
Floating Point Unit	Y	Y	N	Y	Y
Register File					
Width (bits)	32	32	72	32	32
Number	128	2×64	4 kbytes	2×16	4×8 <sup>3</sup>
Ports	15R 5W	5R 5W	6R 6W	10R 6W	8
Predicate Registers	uses regs	16	none	none	none
On-Chip Memory			8K Total		
Instruction Memory	32K Cache	16K Cache		64K Cache/Mem	12K Cache
Data Memory	16K Cache	16K Cache		64K Data	34K Data
Fixed Function Units	FIR, VLx	VLx, 2D/3D, FIR	Motion, 3D	N	N
Interfaces					
Host	PCI	PCI	PCI/AGP	Host Port	Bus
Off-Chip Memory	SDRAM	SDRAM	2×Rambus	various	any via bus
Audio	I/O	I/O	I/O	N	N
Video	I/O	I/O	I/O	N	N
Serial	I/O	I/O	I/O	Y	N

**Table 1.** Summary of Mediaprocessor Features

Current memory designs generally approach the data transfer side by using high-bandwidth off-chip memories. In addition, mediaprocessors typically contain dedicated I/O ports for serial, audio, video and host transfers. On-chip memories tend to be large (as in general purpose processors), but not always organized as caches. Important is a dedicated DMA engine that in some cases is a small processor of its own, whose main task is to ensure that data are available (usually close to the processor) when they are needed for processing.

In the remainder of this paper, we will first summarize the features of five modern mediaprocessors: Philips TM-1000, Chromatic Research Mpact 2, MAP1000 from Equator Technologies and Hitachi, Texas Instruments TMS320C6x and TMS320C80. Then, we will discuss how these processors compare in the key areas of data transfer, data storage and programming.

## 2. ARCHITECTURE OVERVIEW

So far, we have roughly defined our mediaprocessors as a VLIW processor with wide data path supporting SIMD instructions and a memory hierarchy able to perform long non-sequential data transfers with little processor intervention. Let us briefly look at each processor in turn now to summarize their main features (see Table 1).

<sup>1</sup>This information was not available to us.

<sup>2</sup>Billion Operations Per Second

<sup>3</sup>Eight data registers, a status register and a special multiple flags register.

## 2.1. Philips TM-1000

The TM-1000 is the first processor from Philips Trimedia family.<sup>5</sup> Philips targets it at the video, audio, graphics and communications domain of multimedia applications. The TM-1000 comes with a range of interfaces to allow low-cost system integration. These include a PCI interface, audio input and output, video input and output, a synchronous serial interface, and the main memory interface supporting SDRAM memory.

The processing core, the I/O interfaces, and the memory interface are connected to a central on-chip 32-bit bus. A central bus arbiter is used to control the prioritized request services. For each I/O interface, a simple local DMA engine is employed, which must get permission from the central arbiter before being able to transfer data from/to off-chip memory.

The processing core is a 5-issue VLIW with 27 functional units, including several integer ALUs, shifters, branch units, multipliers, a number of floating-point units (includes square root/divide) and four DSP ALU units dedicated to partitioned operations. The data path is 32 bits wide, but the DSP units can treat it as partitioned data consisting of four 8-bit operands or two 16-bit operands. All operations can be performed conditionally based on the least significant bit of a register. Consequently, the register file (a total of 128 registers) must support up to three reads and one write for each issue slot per cycle, resulting in a total of 20 ports.

The instructions are stored in a compressed format. A 10-bit header is associated with each 5-issue VLIW instruction, with every two bits indicating one out of four instruction types, i.e., no-op, 26-bit operation, 34-bit operation, or 44-bit operation. Therefore, the unused issue slots (no-ops) are encoded in only 2 bits. The variable length instruction coding scheme overcomes the traditional large code size problem of VLIW architectures. However, the instruction fetch pipeline becomes more complicated due to the decompression operations.

Trimedia employs a 32-kbyte 8-way associative instruction cache and a 16-kbyte 8-way associative data cache. Parts of the data cache can be locked to allow it to be used as a general purpose addressable memory. Two fixed-function units are embedded in the TM-1000, one containing 32 5-tap FIR filters and another for variable length decoding. The former can be used, for example, for image scaling while the latter is a key component for MPEG-1 and MPEG-2 decoding.

## 2.2. Chromatic Research Mpact 2

Mpact 2<sup>6</sup> is unusual in a sense that it is a mediaprocessor programmed in-house only and therefore little technical detail is publicly available. It can interface with PCI or AGP, has audio and video input and output ports, a serial port interface, and a RAMDAC. Mpact 2 has been targeted as a coprocessor for multimedia processing in a PC. It uses two separate Rambus interfaces for high off-chip bandwidth, but is lacking a large on-chip data memory. Instead, it has 2 kbytes of instruction memory, 2 kbytes of texture memory (for 3D rendering), and 4 kbytes of data memory that is directly accessed by the functional units. This on-chip data memory has six read and six write ports to be able to keep up with the six functional units, i.e., four integer units, one 3D unit and one motion estimation unit. The four integer units are modified to be able to support the floating point operations.

The external interfaces have their own DMA engines that connect to the processing core. The processing core looks like a DSP with VLIW. The six functional units get their operands from and write their results to the data memory, which can be bypassed to feed data directly between functional units. All the data are fed across a crossbar that is 11 words wide. Each word consists of 72 bits and is partitioned into 9-bit or 18-bit quantities.

## 2.3. MAP1000

The MAP1000 is a new mediaprocessor<sup>7</sup> introduced by Equator Technologies, Inc. and Hitachi Ltd. It offers very solid performance with a programmable core without relying so much on fixed-function units. It includes a similar range of external interfaces as Mpact 2, but uses a single SDRAM or SGRAM interface for its off-chip memory.

Its VLIW processing core consists of two separate clusters with their own register file. This gives a flexible option to scale their processor to more clusters without major architecture modifications. Each cluster contains six functional units. One integer unit that is often used as a load/store and branch unit and is exclusively controlled by the first instruction slot (IALU). The second instruction slot (IFGALU) controls the remaining five functional units, i.e., the integer, floating-point, graphics, divider and media units. The graphics unit supports partitioned instructions and the media unit is 128 bits wide and capable of fully-pipelined inner-product operations on partitioned data.

The  $64 \times 32$ -bit register file in each cluster supports 64-bit data access. In addition, a 16 one-bit predicate register file is used for conditional execution. MAP1000 support a rich instruction set, which allows 8-bit, 16-bit, and 32-bit fixed-point partitioned operations and 32-bit floating point partitioned operations. Various rounding modes and shifting modes can be incorporated in the fixed-point instructions, which results in concise instruction code and more accurate output. Communication across clusters is supported by allowing writes into the other register file. Each cluster has special purpose 128-bit registers for 128-bit wide partitioned operations. Data from a 64-bit register can be used to shift new values into one 128-bit register. Many algorithms (e.g., FIR filter, motion estimation, convolution, DCT) can take advantage of these wide registers for a significant performance boost.

The MAP1000's DMA engine can be programmed to perform many useful data transfer operations without intervention from the processing core. By specifying the cache allocation and cache coherence modes, the programmer can directly control the data transfer between off-chip memory and the data cache. The 16-kbyte 4-way set-associative data cache allows way-masking, so that the data in the masked portion of the cache will not be replaced by the newly-loaded cache line. This provides a convenient mechanism to implement data prefetching, a technique that reduces the memory latency and consequently reduces the overall execution time.

#### **2.4. Texas Instruments TMS320C6x (VelociTI)**

TI's TMS320C6x processor is of the VelociTI family.<sup>8</sup> It is not really targeted at video and graphics processing due to its lack of appropriate interfaces. However, its internal processing core is quite versatile and powerful enough to be used in many multimedia processing tasks. TI has targeted this processor at the communications and multichannel audio domain. The C6x has two separate 100 MHz serial ports and a 16-bit, 50 MHz host port. The memory interface can connect to several memories simultaneously, including SDRAM, SBSRAM, SRAM and ROM.

Like the MAP1000, it contains a DMA engine that can transfer data from the interfaces to on-chip memories, but it is less sophisticated and can only perform fixed address, 1D and 2D address striding movement, and data interleaving. The C6x offers more on-chip memory than any of the other processors. A total of 64 kbyte of instruction memory or cache (configurable) and 64 kbytes of data memory are available. The data memory is addressable and cannot be used as a cache.

The processing core is split into two units as in the MAP1000, but each unit contains a total of 12 functional units with the ability to issue 4 operations per cycle. The data path is 32 bits wide, but supports 40-bit long integers and 64-bit floating-point operations. Floating-point operations are only supported in the C67x, which is clocked a little slower than the C62x. The C6x supports only few partitioned instructions, mostly dual 16-bit arithmetic by inhibiting the carry from the 15th into 16th-bit position. A single read port exists for reading from the other register file. This is just opposite to the MAP1000, where communication across register files is done via a write port.

#### **2.5. Texas Instruments TMS320C80**

The TMS320C80<sup>9</sup> (MVP) is the oldest processor in our review, yet its performance is very respectable particularly in image processing, which is its primary application domain. Externally, the C80 interfaces via a 64-bit bus that requires external glue logic to interface to other components (frame buffers, host processor, memories). Ultimately, this increases the cost of C80-based systems. The bus connects to a sophisticated DMA engine that is called the Transfer Controller (TC). The TC can perform flexible and sophisticated memory transfers from off-chip to any of the on-chip memories, and vice versa.

The MVP contains a total of five independent processors and is therefore a true single-chip MIMD processor. The Master Processor (MP) is a general purpose RISC engine with its own instruction cache and data cache. In addition, there are four Advanced DSPs (ADSP) that are connected via a crossbar to many shared data memory blocks. Each ADSP has its own instruction cache.

The ADSPs contain eight data registers and numerous other special purpose registers. Each contains a number of simple functional units, e.g., barrel rotator, mask generator, 32-bit ALU, multiplier, two addressing units and loop controller. The ALU and multiplier support 16-bit and 8-bit partitioned operations.

### 3. DATA TRANSFER AND STORAGE

Most programs access their data with significant temporal locality. In other words, data that have been accessed recently will be accessed again soon. Computers exploit this with a hierarchy of memories. Smaller and therefore faster memories are placed closer to the functional units.<sup>10</sup> This allows the frequently-accessed data to be placed close to the processor where it can be referenced again quickly. The memory hierarchy encompasses registers, on-chip memories, off-chip memories, disk, and archival storage.

In the first half of this section, we will define a number of characteristics of memory references. We will use these in the second half to discuss the memory hierarchy of the processors we reviewed.

#### 3.1. Data Reference Characteristics

Different multimedia algorithms access data in different ways. In order to evaluate the memory architectures of the reviewed mediaprocessors, we will categorize the different access patterns according to predictability, spatial locality, temporal locality, and data set size.

Our first category is predictability. An access pattern that is *predictable* is known in advance of the computation (the address generation can be done statically). These are found less often in general purpose algorithms, but are common in multimedia algorithms. For example, in the finite impulse response (FIR) filter algorithm, three data sets are used, the input data, output data and FIR coefficients. Assuming a reasonably large register file, the input data are read sequentially from memory and the output data are written back to memory sequentially. The values stored in those streams or in the coefficients do not affect their access order and therefore we call this reference characteristic *predictable*. In contrast, data stored in lookup tables are often accessed unpredictably (e.g., as transcendental operations via a lookup table). The exact ordering of table lookups depends on the input data and therefore cannot be determined at the time the application is compiled. We have already defined temporal locality. High spatial locality exists when data are accessed that are stored closely together. Low locality means that successive references are not likely to come from nearby memory locations.<sup>10</sup>

Our final dimension is an estimate of data size. With *very small size*, we refer to data that can be stored in a register file. *Medium size* data can roughly fit as a whole in on-chip memory. Everything else is classified as being of *large size*, in particular most media data are of large size (e.g., video and audio streams).

Our boundaries are not intended to be fixed rigid boundaries. In some cases, gray areas exist. For example, we do not distinguish between temporal or spatial locality for very small data sizes as we do not believe those to be of importance (not when stored in the register file). Furthermore, algorithms can often be modified to move from one category to another. Matrix multiply is an example where locality can be improved by partitioning the input matrices into blocks and performing matrix multiply on these sub-matrices to compute partial results that can be accumulated to obtain the final result. However improving the access pattern of one stream could adversely affect another and/or increase the computational requirement.

#### 3.2. Architectural Choices

We will now explain the major architectural features of our reviewed mediaprocessors related to the memory hierarchy and describe how they deal with the different data reference categories that we defined previously. Not all combinations of categories are meaningful, so we only discuss the ones that are most relevant to the spectrum of existing multimedia applications.

##### 3.2.1. Register File

In most architectures, the register file is where the functional units receive their operands from<sup>4</sup>. As such, it is generally the smallest and fastest memory on the processor. There are three main uses for the register file. First, it can be used for storing data of the *very small size* category. This can be coefficient data as in 8-point discrete cosine transform (DCT) and FIR filter (although these algorithm can of course also have a large number of coefficients). These data usually form an integral part of the algorithm and are often accessed on every cycle of the algorithm tight loop. Storing the data in the register file has the following advantages: (1) no address computation needs to be

---

<sup>4</sup>In some cases, bypassing can be used to directly feed result of one functional unit to the input of another.

done to retrieve the data, (2) the data are available with the lowest possible latency, and (3) valuable on-chip and off-chip memory bandwidth is saved for other tasks.

Second, it can store short-lived data, e.g., an intermediate result of a computation. This is different from the previous group that contains data that have a longer life-time. An example is a single element of a FIR filter input stream. Each input stream element must be multiplied with each FIR coefficient in turn. For a few cycles, the element is accessed frequently, then discarded. The register file is a good storage for holding such data.

Third, extra register are needed when software pipelining is applied. The functional units of most mediaprocessors are deeply pipelined to achieve the lowest possible cycle time. In pipelining, an operation takes many cycles before it is completed even though a new instruction can be issued every cycle.<sup>10</sup> An instruction requiring the result from a previous instruction may need to be delayed for several cycles to wait until the result has been computed. Software pipelining is a powerful compiler optimization technique that restructures the instructions to eliminate these idle cycles. It does so by unrolling a tight loop several times and overlapping multiple iterations of a loop,<sup>11</sup> thus creating the need for more registers.

The first two groups suggest the need for a large register file. In particular, when one data stream is arithmetically combined with a set of coefficients to produce an output stream (e.g., FIR filter, DCT), the on-chip bandwidth may already be pushed to its limit. In particular, most mediaprocessors can only perform one or two memory accesses per cycle. The MAP1000, for example, can perform two loads or stores per cycle. The ability to keep even somewhat larger coefficient sets in the register file can keep an algorithm from becoming I/O-bound.

The third group (software pipelining) does not add to, but multiplies the current register requirement. When software pipelining is performed, we may have unrolled a loop, for example, four times. That means, we now have four times the number of intermediate values to keep in the register file. Software pipelining is a key for fully utilizing the available resources on heavily-pipelined mediaprocessors, yet its contribution to the overall performance improvement in a particular application could be limited by an insufficiently large register file.

Not surprisingly, therefore, all mediaprocessors contain a large number of registers. It ranges from 32 to 128. However, register file size is not going to scale arbitrarily because it increases die area, cycle time and instruction fetch bandwidth due to more bits needed in encoding the register address.

Mpact 2 takes a slightly different approach by using a multi-ported 4-kbyte on-chip memory instead of a register file. Due to its multi-porting, this memory does in fact behave very much like a register file. Its larger size is an advantage, but from the available documentation, we could not infer whether this memory is organized like a cache or addressable like a SRAM module. In Mpact 1, the on-chip SRAM is addressable. This makes programming Mpact 1 significantly more difficult than a register-based processor, because the programmer must handle data layout carefully to efficiently use the limited on-chip storage.

### 3.2.2. On-Chip Cache

Generally, the processor performance doubles every 18 months. In comparison, off-chip memories have been growing in speed much more slowly. Their sustainable bandwidth on sequential accesses has been improved considerably with new memory designs like SDRAM and Rambus, but their access latency has changed very little over the past years. This gap in functional unit latency and off-chip memory latency has grown significantly and is now at a point where a processor may execute a hundred cycles between requesting and receiving data from off-chip memory. Caches play a key role in keeping the functional units busy, thus achieving good performance. Let us first review the desirable and undesirable properties of caches and then discuss why they are also a frequently-used on-chip memory form in mediaprocessors.

A cache does mainly two things when asked to retrieve data from off-chip memory. First, it fetches not just the requested data, but all the data within a small block size around the requested word. If the data reference pattern has high spatial locality, such prefetching will prevent future cache misses<sup>5</sup>. Second, it attempts to keep frequently-accessed data in its storage space, by replacing other less frequently-accessed data. A reference pattern with high temporal locality will benefit from this.

A cache does not reduce access latency relative to off-chip memory when there is little spatial or temporal locality and when the data set is significantly larger than the cache memory size. This is because the data that have been

---

<sup>5</sup>A cache miss means that data referenced do not exist in the cache and must first be loaded from off-chip.

stored in the cache are not likely to be accessed again in the near future (lack of temporal locality) and neither will nearby data (lack of spatial locality). When such data are accessed again, chances are high that they have already been replaced by other data. Furthermore, caches do not take advantage of the predictability of a memory reference pattern. Since a cache does not have information about what the future reference pattern is, it cannot take advantage of it.

The TM-1000, MAP1000 and MVP processors all contain some amount of on-chip data cache. In the MAP1000 and TM-1000, the data caches are the only memories directly supplying data to the functional units. In MVP, only the RISC core has its own data cache while each of the four ADSPs has several SRAM memory modules for data storage. Caches work well with medium data sizes and large data sizes that have a lot of spatial or temporal locality. Table lookups and larger coefficient data that cannot be stored in the register file often fall in this category. Very common in multimedia algorithms is streaming data (e.g., FIR filter, FFT) where data are referenced mostly sequentially. A cache's prefetching ability can handle streams well, especially when there is some amount of temporal locality as in convolution with larger kernel sizes. However, this temporal locality is generally of a very short duration, the data are rarely accessed for a long period of time. What this means is that a stream will cause large amount of data to be brought into the cache that will replace other data that may still be useful. This is a common problem with caches and is generally referred to as cache pollution by the streaming data.

If caches do not work all that well with streaming references, why then are being employed on a number of mediaprocessors? High-level programmability is certainly one reason and the other may be that none of the other approaches handle larger unpredictable data sets with high locality much better. There is more to programmability than to ease the programming task. A cache is generally invisible to the programmer, and code does not have to be written to exploit a cache (except to improve locality if possible). This property is especially valuable to compilers that currently do not perform significant optimizations on the memory reference pattern. Nevertheless, all processors do offer alternatives to the basic cache approach. These are addressable on-chip memories and DMA engines. We will discuss those next.

### 3.2.3. Addressable On-Chip Memory and DMA Engines

Streaming data are a dominant memory reference form in multimedia algorithms and as previously discussed are not efficiently dealt with in a pure cache-based system. The primary concern is that streaming data tend to replace a lot of other useful data that may be stored in the cache.

A common answer is to replace the cache with a similar-sized addressable on-chip memory (C6x and MVP) and control the data flow between on-chip and off-chip memory using a DMA engine. Streaming data are often fetched from off-chip memory sequentially (e.g., FIR filter, FFT), as a 2D reference (e.g., blocked matrix multiply) or using some other access patterns. A DMA engine does not need to be very complex to handle these easily. Synchronization between the DMA engine and functional units is typically handled by double buffering the stream. While the functional units process a particular block of the input data, the DMA engine can load the next into a second region of the on-chip memory. The exact dimensions of these blocks are fixed. Therefore, no other data stored in the addressable memory are disturbed compared to the case in a cache-based system.

Table data can also be handled well in an addressable memory assuming that they fit in entirety in the on-chip memory or can somehow be broken into small parts. An example for this is variable length decoding using a table-driven implementation.<sup>12</sup> In variable length decoding, locality is in fact very low as the input data tend to be fairly randomly distributed. Therefore, for both cache and memory-based approaches, it is important to be able to fit the entire data structure in on-chip memory. The only additional step that is needed with the addressable memory approach is to preload the table into the on-chip memory. The remaining part of the implementation is otherwise identical to a cache-based system.

The most significant disadvantage of the on-chip addressable memory is the complexity in managing it. The programmer must specify exactly how data are going to be laid out in the on-chip memory and must initiate all DMA transfers at the correct times. Current compilers are not quite capable of performing this task efficiently for the programmer.

The TM-1000 and MAP1000 processors use lockable caches to be able to reconfigure it as an addressable memory. Parts of the cache memory can be locked, so that it does not get replaced by other data. It can be written to by

a DMA engine or the functional units. This feature retains the advantages of both cache and addressable memory, but the programmer still needs to choose between either one of these two.

Mpact 2 has no large on-chip memory. Cost may have been a deciding factor as on-chip memories (especially caches) often occupy significant portions of the processor die area. Mpact 2 has a disadvantage with algorithms that contain moderate amount of locality (e.g., 2048-point FFT), as it is unable to store all the input and output data on-chip. However, it does have two Rambus interfaces giving it a respectable off-chip bandwidth of 1.2 Gbytes/s. This bandwidth in fact comes close to the on-chip bandwidth available on several mediaprocessors and even exceeds the on-chip bandwidth available on the TM-1000 (0.8 Gbytes/s). In spite of this, latency cannot be ignored and any non-sequential access pattern will exhibit considerably less than the peak off-chip bandwidth.

## 4. PROGRAMMING INTERFACE

The current programmable mediaprocessors employ the VLIW architecture to support ILP and they employ SIMD instructions in order to exploit the data-level parallelism in media processing applications. Using a VLIW architecture, the computation power can be improved by a factor equaling to the number of issues in one VLIW instruction. SIMD instructions can further improve the computation power by a factor equaling to the number of partitions in each functional unit. However, this computation power can only be achieved by efficiently implementing the algorithms on the targeted architecture.

Two approaches have been used to implement applications on VLIW processors, assembly programming and high-level programming. In this section, we will compare the two programming strategies and describe the current software tools that provide the programming environment for mediaprocessors. We also discuss DMA programming, which aims to reduce the memory access latency and ultimately reduce the processing time.

### 4.1. Low-level programming

The assembly programming approach is mostly used in the conventional DSP applications, where fast performance is important. To achieve the optimized performance, the programmers must be able to fully understand the processor architecture and the algorithms. Given the multi-issue capability and vectorized instructions, the low-level programmer must first find the parallelism within each algorithm. There are mainly two steps: (1) determine the SIMD instructions that can be applied to the algorithm and (2) schedule the instructions to maximally utilize the issue slots without violating the instruction latency rules. In general, the programmer's productivity is highly restricted by the architectural complexity. It might take several months or years to implement a truly optimized algorithm and it is also difficult to maintain and upgrade the assembly-level source code.

### 4.2. High-level programming

The main goal of the modern mediaprocessors is to obtain high performance by high-level language programming. The modern compilers try to examine the parallelism within a piece of sequential source code and schedule the instructions to fill up the wide issue slots as much as possible. The instruction latency is also taken into account during this optimal scheduling. Many compilers support various compilation options and optimization schemes, such as software pipelining. Also, the programmers can insert some hints in the high-level source code to guide the compiler. For example, a loop unrolling amount inserted before a loop would advise the compiler to unroll the loop, so that the overall performance in the loop body can be maximized. Another example is to tell the compiler explicitly that two specified memory spaces are not overlapped, so that data accesses to the two memory locations can be treated as independent.

While the compilers are efficient in scheduling and optimizing the instruction issues, it is difficult for them to extract data-level parallelism from sequential source code accessing individual data elements. Although the compiler may detect some obvious vectorized computation patterns and find the appropriate SIMD instructions, it cannot generate the data packing and reordering instructions to prepare for the vectorized computation. Therefore, the current compilers include intrinsics in their high-level language, with each intrinsic corresponding to an SIMD instruction. The programmer can invoke the SIMD instructions by using the intrinsics in the high-level source code. The programmer still needs to understand the instruction set architecture to exploit the data-level parallelism and map the algorithm accordingly in a high-level language.

The current compiler technologies provide a better programming environment for modern mediaprocessors in high-performance multimedia applications. The high-level language programming results in more compatible source code among different generations of processors and eases the code maintenance and upgrade tasks. However, the manually-optimized assembly code usually outperforms the compiler-optimized one, with the performance ratio dependent on the complexity of algorithms, programmer's skills, etc.<sup>13</sup> In addition, the compiler usually generates code with a larger size than the manually-optimized assembly code.

### 4.3. DMA programming

Both the assembly and high-level programming approaches aim to reduce computation cycles. However, the overall performance heavily relies on the data flow efficiency in addition to how well the tight loop has been programmed. Various data prefetching techniques have been used to overlap the data transfer with the computation time, so that the memory stall cycles due to data fetching are minimized.<sup>14</sup> A sophisticated programmable input/output engine is most widely used by the mediaprocessors. The Data Streamer in MAP1000 and the Transfer Controller in MVP are such examples.

Although high-level language APIs are provided for programming the data transfers, the programmer must determine and set up the parameters for each data transfer. In addition, the programmer must handle the synchronization between data transfers and computation. No compiler-directed prefetching techniques have been reported in utilizing the block-based DMA prefetching. However, the data transfer template has been proposed,<sup>15</sup> that relieves the programmer from the data flow control details. The scheme is effective in dealing with multimedia processing applications, where data flow patterns are relatively regular.

## 5. CONCLUSIONS

We have reviewed the main architectural features of five recent mediaprocessors. From the processing side, we see that the large amount of ILP and data parallelism has made a VLIW architecture with SIMD instructions the dominant choice. As a matter of fact, future general purpose processors tend to follow this trend. From the memory system side, most mediaprocessors are constrained by a common goal for low cost. Generally, the well-understood cache-based memory architecture in mediaprocessors is getting more popular, but has been augmented with simple to advanced DMA engines and the ability to directly address parts of the cache memory. While the compilers seem to be the way of the future, the code performance they provide has not yet reached the same performance level that the hand-coded programs can achieve.

## ACKNOWLEDGEMENTS

We would like to thank Inga Stotland for her valuable comments on an early draft of our paper and Yukio Chiba and Yoochang Jung for providing us with valuable information on the reviewed architectures.

## REFERENCES

1. K. Konstantinides, "VLIW architectures for media processing," *IEEE Signal Processing Magazine* **15**(2), pp. 16–19, 1998.
2. J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *10th Annual International Conference on Computer Architecture*, pp. 140–150, June 1983.
3. A. Capitanion, N. Dutt, and A. Nicolau, "Partitioned register files for VLIWs: a preliminary analysis of trade-offs," in *MICRO 25. Proceedings of the 25th annual international symposium on Microarchitecture*, pp. 292–300, December 1992.
4. E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith, "Trace processors," in *MICRO 30. Proceedings of the 30th annual international symposium on Microarchitecture*, pp. 138–148, December 1997.
5. S. Rathnam and G. Slavenburg, "Processing the new world of interactive media," *IEEE Signal Processing Magazine* **15**(2), pp. 108–117, 1998.
6. S. Purcell, "The Mpact 2 VLIW media processor improves multimedia performance in PCs," *IEEE Signal Processing Magazine* **15**(2), pp. 102–107, 1998.

7. C. Basoglu, R. J. Gove, K. Kojima, and J. O'Donnell, "A single-chip processor for media applications: The MAP1000<sup>TM</sup>," *International Journal of Imaging Systems and Technology*, in press, 1999.
8. N. Seshan, "High Velocity processing," *IEEE Signal Processing Magazine* **15**(2), pp. 86–101, 1998.
9. K. Gutttag, R. J. Gove, and J. R. Van-Aken, "A single chip multiprocessor for multimedia: The MVP," *IEEE Computer Graphics Application* **12**(6), pp. 53–64, 1992.
10. D. A. Patterson and J. L. Hennessy, *Computer Organization & Design*, Morgan Kaufmann Publishers, San Mateo, CA, 1994.
11. V. H. Allan, R. B. Jones, R. M. Lee, and S. J. Allan, "Software pipelining," *ACM Computing Journal* **27**(3), pp. 367–432, 1995.
12. C. Fogg, "Survey of software and hardware VLC architectures," in *Image and Video Compression, Proc. SPIE*, vol. 2186, pp. 29–37, 1994.
13. C. Basoglu, D. Kim, R. J. Gove, and Y. Kim, "High-performance image computing with modern microprocessors," *International Journal of Imaging Systems and Technology*, in press, 1999.
14. T. Chen and J. Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE Transactions on Computers* **44**, pp. 318–328, 1995.
15. J. H. Kim and Y. Kim, "UWICL: A multi-layered parallel image computing library for single-chip multiprocessor-based time-critical systems," *Real-Time Imaging* **2**, pp. 187–199, 1996.