GENERAL ARCHITECTURE AND INSTRUCTION SET ENHANCEMENTS FOR MULTIMEDIA APPLICATIONS

Mansour ASSAF
Information Communications Technology (ICT)
University of Trinidad and Tobago
Arima, Trinidad and Tobago W. I.

and

Aparna RAJESH
Information Communications Technology (ICT)
University of Trinidad and Tobago
Arima, Trinidad and Tobago W. I.

ABSTRACT

The present day multimedia applications (MMAs) are driving the computing industry as every application being developed is using multimedia in one or the other way. Computer architects are building computer systems with powerful processors to handle the MMAs. There have been tremendous changes in the design of the processors to handle different types of MMAs. We see a lot of such application specific processors today in the industry; different architectures have been proposed for processing MMAs such as VLIW, superscalar (general-purpose processor enhanced with a multimedia extension such as MMX), vector architecture, SIMD architectures, and reconfigurable computing devices. Many of the General Purpose Processors (GPPs) require coprocessors to handle graphics and sound and usually those processors are either expensive or incompatible. Keeping these and the demands MMAs in mind designers have made changes to GPPs; many GPP Vendors have added instructions to their Instruction Set Architecture (ISA). All these processors use similar techniques to execute multimedia instructions. This survey paper investigates the enhancements made to the GPPS in their general Architecture as well as the ISA. We will present the many different techniques used by GPP designers to handle MMAs, the present day GPP available architectures, compare different techniques, and concludes this survey.

Keywords: Multimedia Applications (MMAs), General-purpose Processor (GPPs), Instruction Set Architecture (ISA), VLIW, Superscalar Processor, Vector Processor, SIMD Architectures.

1. INTRODUCTION

The mismatch between wide data paths and the relatively short data types found in multimedia applications has led the industry to embrace SIMD (single instruction, multiple data) style processing. Unlike traditional forms of SIMD computing in which multiple individual processors execute the same instruction, multimedia instructions are executed by a single processor and pack multiple short data elements into a single wide (64 or 128-bit) register, called Subword Level Parallelism(SLP) or MicroSIMD [1]. Recently there have been tremendous changes to the system architecture itself to speedup the processor. Superscalar processors with dynamic out-of-order scheduling provide higher performance than VLIW processors and superscalar processors with in-order scheduling. Because superscalar architectures include complicated control logic for out-of-order execution, and because VLIW processors have to decode every instruction slot in parallel and need a register file with multiple read and write ports, they are more complex than single-issue vector architectures. This paper investigates the enhancements made to the GPPS in their general Architecture as well as the ISA.

2. CHARACTERISTICS AND FUNCTIONS OF MULTIMEDIA APPLICATIONS THAT ARE CONSIDERED IN GPPS

MMAs have many characteristics that make them unique from General-Purpose Applications (GPAs). The most important ones are the following:

- Real-time response: MMAs such as video conferencing and electronic commerce often require real-time response. In addition, they require a certain quality of service.
- Processing of streaming data: MMAs can keep their instruction code on-chip and commonly stream data in from off-chip.
- Significant fine and coarse grained data parallelism: Typically, MMAs perform the same operations on different data item (e.g., pixels). In addition, many functions need to be performed on these data values. Since these operations and functions are largely independent, it is possible to exploit SIMD and Thread-Level Parallelism (TLP) [1].
- Considerable data reorganization: In addition to the SIMD nature of multimedia processing, most applications also need to be able to reorganize the individual data components efficiently to adjust for various data stream layouts. Therefore, MMAs are not well suited for traditional SIMD architectures where data reorganization can be expensive.
- Small loops: MMAs spend nearly 95% of their execution time over the two innermost loops. These loops tend to have a large number of iterations, typically 10 or more, with some loops having hundreds or thousands of iterations.
- High memory bandwidth requirement: The applications process large data sets, putting a severe burden on memory system.
- Small data types: MMAs typically use small integer data types of 16 bits or less. Additionally, MMAs perform significantly more arithmetic operations than GPAs.

A variety of multimedia processing algorithms are used in media processing environments for capturing, manipulating, storing, and transmitting multimedia objects such as text, handwritten data, 2D/3D graphics, and audio objects. Multimedia standards such as MPEG-1, MPEG-2, MPEG-4, MPEG-7, JPEG2000, and H.263 put challenges on both hardware architectures and software algorithms for executing different multimedia processing jobs in real-time, because each media in a multimedia environment needs different algorithms, processes, and techniques.

3. EARLIER GENERAL-PURPOSE PROCESSOR (GPPS) WITH MULTIMEDIA SUPPORT

The real-time multimedia processing on PCs and workstations is still handled by dedicated multimedia processors. However, the advanced GPPs provide an efficient support for certain multimedia applications. These processors can provide software-only solutions for many multimedia functions, which may significantly reduce the cost of the system. Many microprocessor instruction sets include instructions for accelerating MMAs such as DVD playback,speech recognition and 3D graphics. The main differences among these processors are in the way they reconfigure the internal register file structure to accommodate SIMD operations, and the multimedia instructions they choose to add.

Subword Level Parallelism:

The goal in designing SIMD media ISA extensions for GPP has usually been to utilize Subword Level Parallelism (SLP) with existing hardware and without sacrificing the general-purpose nature of the processor [1]. SLP a very low-cost form of small-scale SIMD parallelism in a word-oriented processor. A word-wide integer functional unit can be partitioned into parallel subword units, with small hardware overhead. As illustrated in Figure 1, a 64-bit adder may be partitioned into four 16-bit adders. Such a partitionable adder allows four 16-bit additions, or a single 64-bit addition, to be performed in a single cycle.

The overhead cost is very small, since the same datapaths are used in either case: two 64-bit registers read and one register write. A processor with two 64-bit partitionable ALUs could support eight parallel 16-bit operations with just a 6-ported (4 read and 2 write ports) register file, while a processor with eight independent 16-bit functional units requires a 24-ported register file.

Multimedia kernels process small data types and the registers of GPPs satisfying these requirements. In particular, the double-precision FP registers can hold several of such elements [2]. The same operation is applied to the subwords at the same time. The SLP is a cost-effective solution to exploit the DLP present in MMAs. There is no need to replicate the functional units and the memory port can supply multiple elements at no cost. Earlier extensions supported only integer data types and were introduced in the mid-1990's.3DNow was the first to support floating-point media instructions. It was followed by Streaming SIMD Extension (SSE) and SSE2-4 from Intel. Motorola's AltiVec supports integer as well as floating-point media instructions.

The main differences between these processors are in the way that they reconfigure the internal register file structure to accommodate SIMD operations, and the multimedia instructions they choose to add. Multimedia instruction sets can be broadly categorized according to the location and geometry of the register file upon which SIMD instructions operate. The alternatives are reusing the existing integer or floating point register files, or implementing an entirely separate one. The type of register file affects the width and therefore the number of packed elements that can be operated on simultaneously (vector length).

Despite the similarities, each approach to subword extensions is unique. Key differences include the amount of additional hardware required, ranging from MAX-2, which reuses the integer registers and execution units and requires virtually no additional execution hardware, to AltiVec, which requires an entirely new execution unit. In Table 1, shown below, common and distinguishing features of available GPPs (SN and UN indicate N-bit signed and unsigned integer packed elements respectively) with multimedia instruction set extensions are summarized.

Creating and using instruction-level parallelism-Superscalar execution

Parallelism is simply the practice of doing multiple things at once. Parallelism can happen on multiple levels, from executing multiple instructions at once, to executing multiple threads at once, to executing multiple programs at once. Instruction-level parallelism, or ILP, is parallelism at the lowest level. Machines that can exploit ILP are typically called multiple-issue, meaning they issue multiple instructions each clock cycle to the various functional units on the chip [1]. The functional units are the things that do the actual number crunching: floating-point functional units crunch fp ops, integer functional units crunch integer ops, etc. The more functional units you have, the more instructions you can execute in parallel each cycle. The number of instructions that can be crunched each clock cycle determines the number of executions slots per cycle, and the number of execution slots per cycle is often called the width of the machine. For example, a 4-wide machine (figure 2) would be able to execute four instructions each cycle, and thus it would have four execution slots per cycle.

For a multiple-issue machine to be running at maximum efficiency, all of these execution slots must be kept full on every cycle. Empty slots are the enemy of performance--they mean wasted resources and wasted time. In order to keep those slots full, you've got to schedule the instructions so that they don't conflict with each other. It's not cool if one instruction has a resource tied up and you try to execute another instruction that needs that resource. That second instruction will just have to wait until the resource comes free, and that waiting wastes time.

Dynamic Scheduling of Instructions

Where and how that scheduling gets done is a major issue in CPU design. Earlier GPPs with MM extensions are the dynamically scheduled superscalar machines. A dynamically scheduled superscalar CPU (figure 3) takes a sequential list of program instructions, decides which instructions can be executed on the same clock cycle, and sends them out to its functional units to be executed. It requires the CPU to do a lot of work. The instruction scheduler takes up a lot of transistors, transistors that could possibly be put to use actually crunching numbers.

The most important features of some GPPs with specialized media instruction set extensions are:

- The processors issue and execute two or more multimedia instructions per cycle.

- These processors issue and execute instructions out-of-order. To do so, they require a substantial amount of hardware.
- These hardware components occupy a large portion of the silicon area and contribute significantly to the power dissipation.
- These processors employ a dynamic branch prediction technique.
- The cache mechanism is designed to exploit 1D locality of consecutive addresses, but media applications require multidimensional locality of accesses.
- The word lengths of these processors are 32 or 64 bits. But the word lengths needed for multimedia applications are typically 8 or 16 bits.
- They implement in excess of 15 million transistors on a chip.

Multimedia extensions have proven to provide significant performance benefits by exploiting the DLP present in multimedia codes. However, these GPPs equipped with multimedia extensions have the following limitations:

- Memory misalignment problems: The nature of subword data introduces memory misalignment problems. Accessing data that is not aligned requires extra instructions.
- Mismatch between storage and computational formats: The computational format is usually larger than the storage format.
- Limitation on the amount of parallelism: The fixed size of the multimedia registers limits the amount of parallelism that can be exploited by a single instruction to at most 8 (VIS, MMX) or 16 (SSE, AltiVec) parallel operations, while more parallelism is present in MMAs.
- Overhead instructions: Implementations of multimedia kernels with short-vector SIMD extensions require a significant amount of overhead for converting between different packed data types and for data alignment, increasing the instruction count. For VIS up to 41% of the total instruction count constitutes overhead.
- No strided memory accesses: Most GPPs can only perform stride-1 memory accesses. It is therefore, inefficient to access, for example, a column of a matrix.
- Scalability: The scalability of subword parallel processors cannot be achieved by simply increasing the machine word size.
- Suffer from a lack of compiler support. This limits developers to using in-line assembly macros and low-level library calls.

4. NEW MEDIA PROCESSING ARCHITECTURES

We will present some of the new GPP architectures that overcome many of the limitations of the earlier GPPs. Motivations for the new architectures are:

- VLSI technology is increasing the number of transistors available on a single die.
- Compiler technology is very advanced now, however, it still has some limitations
- Multithreading is becoming more pervasive.
- "Media-rich" means parallelism.
- Modularity and scalability will become increasingly important.
- JIT compilation will eventually predominate, and binary compatibility will be a thing of the past.
- Convergence will involve integrating multiple IP blocks onto one chip.

IA-64 ISA –Intel Architecture [8] and MAJC – Microprocessor Architecture for Java Computing:

Both MAJC and IA-64 are both multiple-issue VLIW processors (figure 4). A VLIW processor [3] lets the compiler do all the work of instruction reordering; all it worries about is executing whatever the compiler gives it as fast as possible. More specifically, a VLIW compiler, groups instructions into fixed-length packets and those packets are what's fed to the CPU.

One of the problems with this approach is that it can lead to significant code bloat. In a traditional VLIW architecture, the instruction packets are a fixed length, usually between 112 and 168 bits. This means that if you can't find enough instructions that you can execute in parallel to fill up the packet, you have to insert NOPS (no ops) into the packet to fill up the empty spots.

The MAJC architecture [6] specifies that the number of functional units in a processor unit is exactly 4. All MAJC processor units will be 4-wide, with four execution slots per cycle. IA-64, on the other hand, allows an arbitrary number of functional units in an implementation. Allowing up to n-way ILP is Intel's idea of scalability. Sun decided against allowing n-wide implementations because they didn't feel it made the best use of transistor resources. As mentioned earlier, compiler constraints make ILP beyond four-way a matter of diminishing returns. One difference between MAJC and a more traditional VLIW machine is that MAJC allows for variable-length instruction packets. This decreases code bloat, and thus saves on memory. In the MAJC Architecture The hardware level at which instruction-level parallelism operates is the level of the processor unit (figure 5).

The MAJC processor unit consists of four functional units, each of which has its own set of registers. These functional units are what do the actual number crunching. Unlike a traditional architecture, and unlike IA-64, each MAJC functional unit is data-type agnostic. That means that there are no floating-point units, no integer units, no address generation units, etc.; any functional unit can operate on any type of data. MAJC also has data-type agnostic registers. There are no *in*t, *fp*, or *SIMD registers*. Any MAJC register can hold any type of data. Once again, this makes the most efficient use of space, because fp- or integer-intensive apps can use all of the registers on the chip, instead of limiting themselves to some subset of available registers. By way of contrast, Intel's IA-64 has dedicated *fp* and *int* units and registers.

MAJC and IA-64 each take a different approach to handling pipeline interlocks. A good VLIW machine handles all scheduling in software (via the compiler), including interlocks. MAJC is just such a machine, and it requires that the compiler know how many cycles every instruction will take to execute so that it can schedule them optimally. The compiler will be tied to a specific MAJC implementation. So a binary compiled on one MAJC implementation wouldn't run on another, because the instruction latencies might differ.

This is where JIT comes in. Just-in-time compilation would eliminate such binary compatibility problems by compiling the code right before it's run. It is in dealing with interlocks that IA-64 slips a little from the VLIW schedule-it-all-in-software ideal. IA-64 requires the specific implementation to take care of interlocks, using some dynamic scheduling technique like score boarding. This adds to the complexity of the hardware implementation, but it allows for binary compatibility across implementations.

It is in explicit architectural support for thread-level parallelism that IA-64 and MAJC really part ways. IA-64's idea of scalability is to keep adding functional units to a single core. MAJC, on the other hand, limits the number of FUs to four and instead adds multiple cores to a single die. Multiple processor units are organized on the same die in what Sun calls a processor cluster (figure 6).

The processor cluster can contain any number of processor units; the above picture shows a cluster with four units, but you can have as many as you like. In addition, you could mix processor units with other units like a GPU or some DSP processor. In addition to grouping the processor units physically, the processor cluster also provides a nice conceptual grouping for thinking about thread-level parallelism. MAJC also has a thread-level version of pipelining called Vertical multithreading. Whenever a thread is executing and there's a cache miss, a MAJC processor unit switches to another thread and executes it while the original thread is waiting for the data to load from memory. Sun claims that vertical multithreading gives some outrageous performance improvements over more traditional VLIW machines.

5. OTHER ARCHITECTURES

New architectures have specially been proposed for processing MMAs. In has been proposed an ISA extension called Complex Streamed Instructions (CSI) for increasing parallelism by processing of two dimensional data streams.

Matrix registers with accumulators are introduced in the Matrix Oriented Multimedia (MOM) ISA. The MOM architecture investigates combining traditional pipelined vector processing with subword processing. The MOM architecture relies on having a vector register file where every element contains subwords that are processed in parallel.

Another related architecture for processing MMAs is the Imagine processor, which has load/store architecture for one-dimensional streams of data records. Imagine is a stand-alone multimedia coprocessor. The focus of the Imagine project is to develop a programmable architecture that achieves the performance of special purpose hardware on graphics and image/signal processing. This is accomplished by exploiting stream-based computation at the application, compiler, and architectural level. The Imagine stream architecture is a novel architecture that executes stream-based programs. It provides high performance with 48 floating-point arithmetic units and power-efficient register organization [4].

6. RESULTS & DISCUSSIONS

Multimedia processing is the technology for a wide variety of applications. It poses very high demands on devices for transmission, storage, and computation. GPPs equipped with multimedia extensions are suitable for GPAs, have overhead instructions but cannot exploit all DLP present in MMAs.

With its data-type agnostic functional units, vertical multithreading, and Space-Time Computing, MAJC goes one level up from EPIC(IA-64) in that it not only includes techniques for creating and using instruction-level parallelism, but it expands its bag of tricks to include techniques for creating and using thread-level. However both MAJC and IA_64 seem to be the optimistic approaches for the new GPPs to handle MMAs.

Superscalar processors with dynamic out-of-order scheduling provide higher performance than VLIW and superscalar processors with in-order scheduling.

7. CONCLUSIONS

This paper investigates multimedia processors and the enhancements for improving the general-purpose processor architecture with multimedia extension.

Based on the investigation of media processors, we find that the complexity and variety of techniques, the high computation, storage, multi-formats, and multi-standards associated with multimedia processing pose challenges, particularly from the points of scalability, high flexibility, high performance, resource utilization, dynamic adaptation capabilities, and real-time implementation. However, parallel media processors are expected to be best candidate for processing the next generation of multimedia applications, because they can provide both the performance and flexibility through specialized high-speed processing, and highly parallel programmable architectures.

8. REFERENCES

[1] D. Talla, L. K. John, and D. Burger, 2003, "Bottlenecks in Multimedia Processing with SIMD Style Extensions and Architectural Enhancements", **IEEE Transactions on Computers**, VOL. 52, NO. 8.
[2] Y. Chia-Lin, B. Sano, and A. R. Lebeck, 2000, "Exploiting Parallelism in Geometry Processing with General Purpose Processors and Floating-point SIMD Instructions", **IEEE Transactions on Computers**, VOL. 49, No. 9.
[3] M. F. Jacome, and G. De Veciana, 2000, "Design Challenges for New Application-Specific Processors", **IEEE Design & Test of Computers**.
[4] A. Krikelis, **Multimedia Processing Architectures**, Aspex Microsystems Ltd. Brunel University Uxbridge, Middlesex, UK.
[5] N. Slingerland, and A. J. Smith, 2002, "Measuring the Performance of Multimedia Instruction Sets & Performance Analysis of Instruction Set Architecture Extensions for Multimedia", **IEEE Computer Society**.
[6] www.sun.com
[7] M. Schlansker, and B. Rau, 2000, "EPIC: Explicitly Parallel Instruction Computing", **IEEE Computer**.
[8] www.intel.com

FIGURES AND TABLES

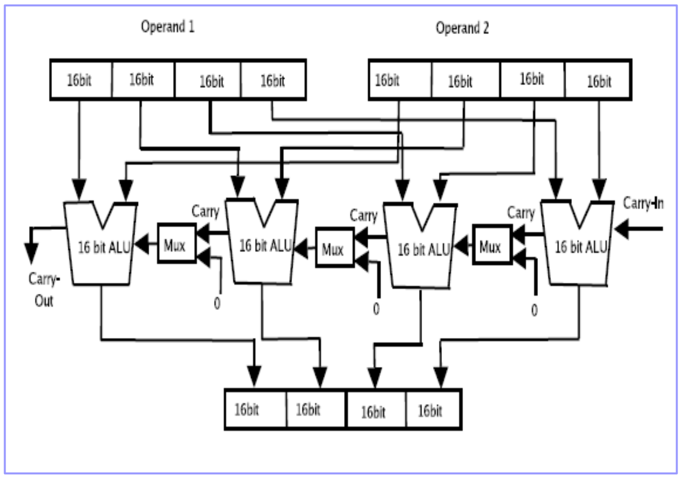


Fig. 1 A 64-bit adder

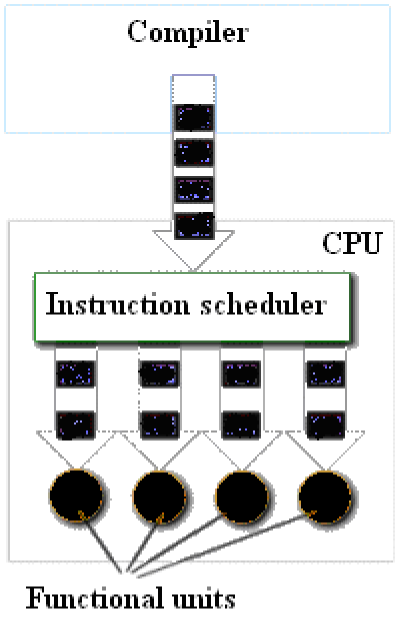


Fig. 3 Dynamic superscalar instruction scheduling

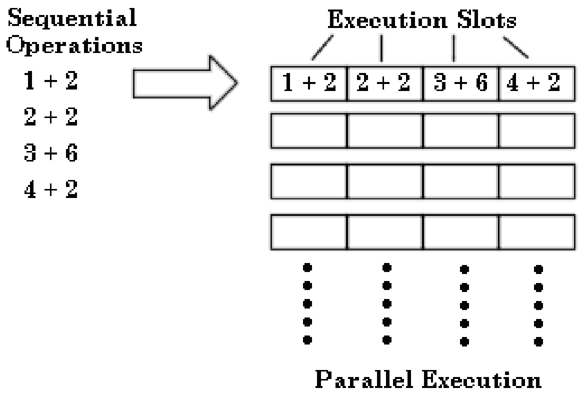


Fig. 2 4-wide machine

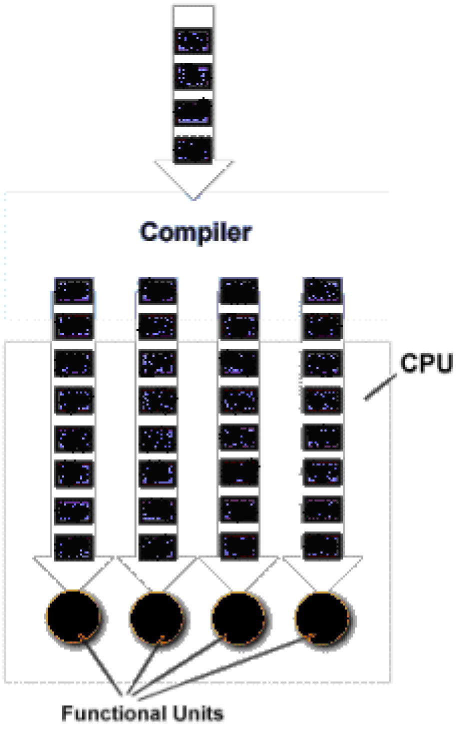


Fig. 4 VLIW instruction scheduling

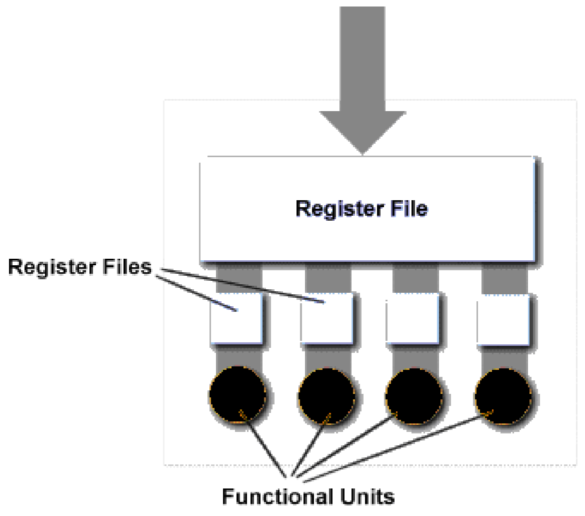


Fig. 5 Processor unit

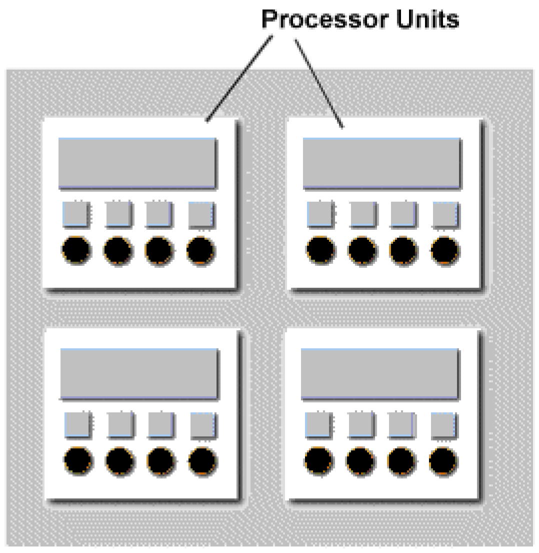
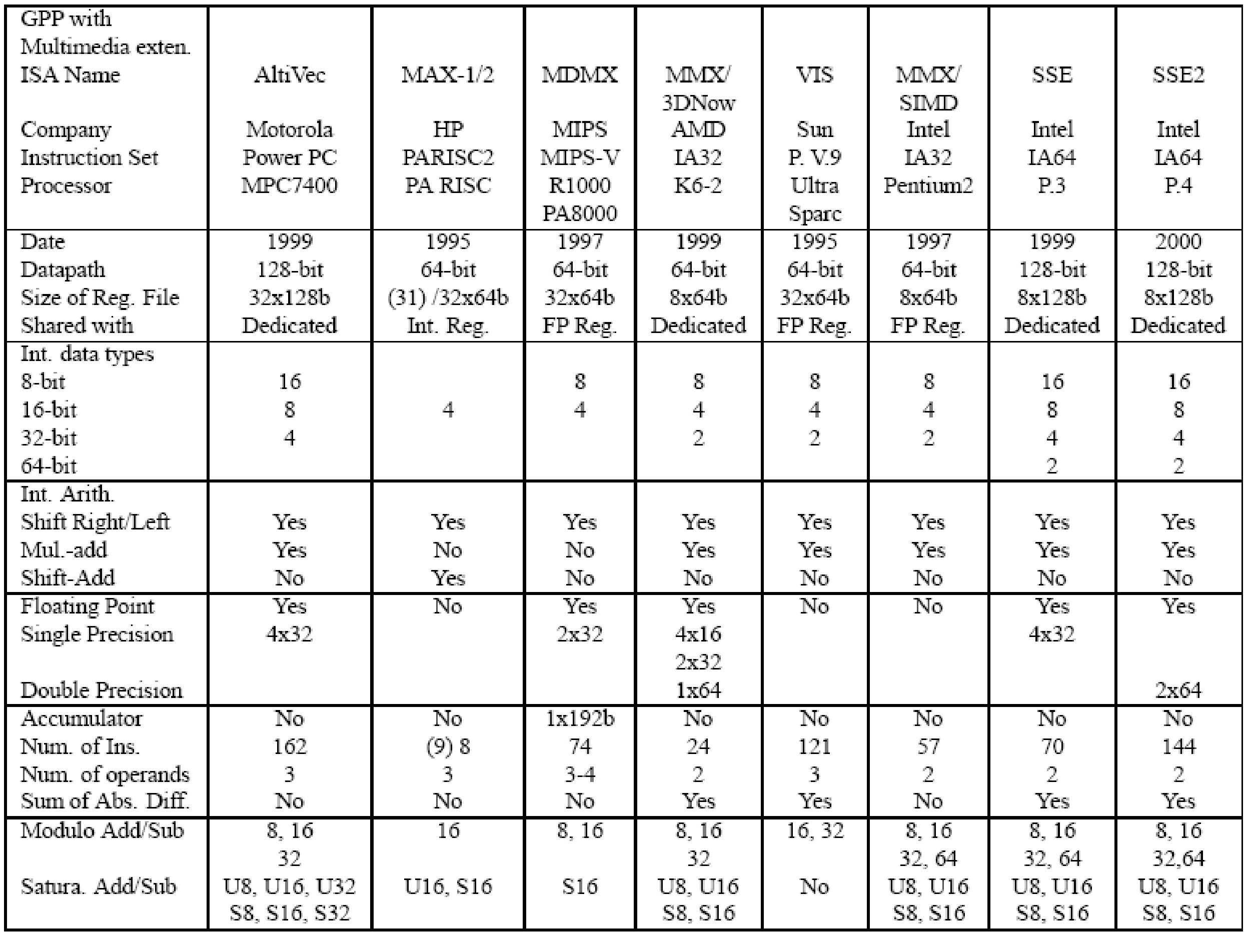


Fig. 6 Processor cluster

Table 1 Available GPPs with multimedia instruction set extensions



GPP with								
Multimedia exten.								
ISA Name	AltiVec	MAX-1/2	MDMX	MMX/	VIS	MMX/	SSE	SSE2
				3DNow		SIMD		
Company	Motorola	HP	MIPS	AMD	Sun	Intel	Intel	Intel
Instruction Set	Power PC	PARISC2	MIPS-V	IA32	P. V.9	IA32	IA64	IA64
Processor	MPC7400	PA RISC	R1000	K6-2	Ultra	Pentium2	P.3	P.4
			PA8000		Sparc			
Date	1999	1995	1997	1999	1995	1997	1999	2000
Datapath	128-bit	64-bit	64-bit	64-bit	64-bit	64-bit	128-bit	128-bit
Size of Reg. File	32x128b	(31) /32x64b	32x64b	8x64b	32x64b	8x64b	8x128b	8x128b
Shared with	Dedicated	Int. Reg.	FP Reg.	Dedicated	FP Reg.	FP Reg.	Dedicated	Dedicated
Int. data types								
8-bit	16		8	8	8	8	16	16
16-bit	8	4	4	4	4	4	8	8
32-bit	4			2	2	2	4	4
64-bit							2	2
Int. Arith.								
Shift Right/Left	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Mul.-add	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Shift-Add	No	Yes	No	No	No	No	No	No
Floating Point	Yes	No	Yes	Yes	No	No	Yes	Yes
Single Precision	4x32		2x32	4x16			4x32	
				2x32				
Double Precision				1x64				2x64
Accumulator	No	No	1x192b	No	No	No	No	No
Num. of Ins.	162	(9) 8	74	24	121	57	70	144
Num. of operands	3	3	3-4	2	3	2	2	2
Sum of Abs. Diff.	No	No	No	Yes	Yes	No	Yes	Yes
Modulo Add/Sub	8, 16	16	8, 16	8, 16	16, 32	8, 16	8, 16	8, 16
	32			32		32, 64	32, 64	32,64
Satura. Add/Sub	U8, U16, U32	U16, S16	S16	U8, U16	No	U8, U16	U8, U16	U8, U16
	S8, S16, S32			S8, S16		S8, S16	S8, S16	S8, S16