Reprinted from proceedings of ICCD 1999, International Conference on Computer Design, October 10-13, 1999, Austin Texas, pp. 586-592.

TriMedia CPU64 Architecture

J.T.J. van Eijndhoven, F.W. Sijstermans*, K.A. Vissers, E.J.D. Pol, M.J.A. Tromp*,
P. Struik, R.H.J. Bloks, P. van der Wolf, A.D. Pimentel**, H.P.E. Vranken
Philips Research Laboratories Eindhoven, The Netherlands
**Philips Semiconductors, Sunnyvale, California*
***University of Amsterdam, The Netherlands*
jos.van.eijndhoven@philips.com

Abstract

We present a new VLIW core as a successor to the TriMedia TM1000. The processor is targeted for embedded use in media-processing devices like DTVs and set-top boxes. Intended as a core, its design must be supplemented with on-chip co-processors to obtain a cost-effective system. Good performance is obtained through a uniform 64-bit 5 issue-slot VLIW design, supporting subword parallelism with an extensive instruction set optimized with respect to media-processing. Multi-slot 'super-ops' allow powerful multi-argument and multi-result operations. As an example, an IDCT algorithm shows a very low instruction count in comparison with other processors. To achieve good performance, critical sections in the application program source code need to be rewritten with vector data types and function calls for media operations. Benchmarking with several media applications was used to tune the instruction set and study cache behavior. This resulted in a VLIW architecture with wide data paths and relatively simple cpu control.

1. Introduction

Our goal was to develop a processor architecture for real-time processing of multimedia data streams. The processor is targeted to mass-produced consumer electronic devices, such as digital televisions and set-top boxes, and as such qualifies as an embedded processor. Software programmability, in combination with respectable computing power, allows a single device to be flexible in a number of ways. This flexibility ensures that the device can be applied in a range of different products (including regional differences) and can adapt to quickly evolving standards in the digital media domain. The silicon cost of high-performance programmability should furthermore be partially compensated by time multiplexing of functionality, where the product performs different functions at different times.

Although the processing power will allow significant processing of real-time video streams, the processor core itself is intended to be integrated on-chip with a set of co-processors which can perform other tasks in parallel. The co-processors will handle tasks with stringent real-time requirements for instance for off-chip communication, and/or tasks with high-throughput but regular computations which are efficiently performed in dedicated hardware. Examples of such co-processors are a 'firewire' (1394) DMA unit and an image horizontal/vertical rescale unit. The co-processors communicate with each other and with the processor core through an on-chip bus, and share access to a single off-chip memory. For different products, the processor core can be combined with different sets of such on-chip co-processors, thus obtaining more flexibility as well as a good performance/silicon-area ratio, which is extremely important for the targeted mass-produced consumer electronic devices. Figure 1 shows the TM1000 as an example. This paper will focus on the VLIW processor core only.

The newly developed core can be regarded as a higher-performance successor of the VLIW core as present inside the current Philips TriMedia TM1000 family of devices. This existing core has the following features [1]:

- a 5-issue VLIW architecture with a 32-bit word size;
- 27 functional units, offering a choice of operation types in each slot in the instruction;
- any operation can be guarded to provide conditional execution without branching;
- instruction set and functional units optimized with respect to media processing;
- a single multi-ported register file with bypass network, allowing 1-cycle latency operations;
- 32 kB, 8-way instruction cache;
- 16 kB, 8-way, quasi-dual ported, data cache;
- a variable-length (compressed) instruction set design.

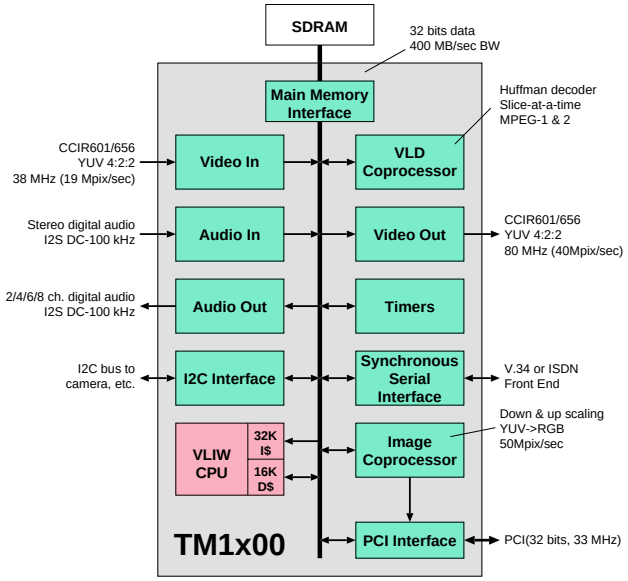

Figure 1. TM1000 overview diagram

This design was taken as a starting point and reference for the new processor. Considerable freedom was however allowed for modifications, with only limited requirements for backward compatibility. A significantly higher processing power was targeted to move more functionality to software, and to handle higher-resolution images in real-time video streams.

2. Related processors

Several other companies have products or developments in the field of embedded media processors. A good overview of techniques for media processing was presented by Faraboschi, Desoli, and Fischer [2].

A relatively new entry is by Equator Technologies, which, in co-operation with Hitachi, announced its MAP1000 processor in December 1998 [3]. This machine has a 4-way VLIW processor with an instruction set optimized for media processing, also supporting subword parallelism in an overall 64-bit design. On-chip co-processors are used for other tasks such as media stream I/O. The resulting performance levels are suitable for HDTV audio/video processing.

In digital signal processing, Texas Instruments clearly has a strong foothold. The TMS320C6x processors can achieve video-speed processing [4]. They have an 8-slot VLIW design executing RISC-like operations that fits well with compiler technology. As in the TriMedia, all operations can be guarded.

NEC developed a 2-way superscalar processor for embedded multimedia use, the V830R/AV. It has one 32-bit pipeline for integer operations, and one 64-bit pipeline for media operations supporting subword parallelism and 3-argument operations [5].

Interestingly, these three processors all operate on a bi-partitioned register file. This contrasts with the single global register file of the TriMedia processors.

Most general-purpose processors have extensions in their instruction set for media processing. Many of them do not target the embedded market, as their price and power settings are not suitable for mass-produced consumer electronic devices. However, the PowerPC with Altivec extension surely deserves mention here [6]. The PowerPC is a superscalar design, with an additional new execution pipeline with a dedicated register file for media processing, having a 128-bit wide datapath supporting 3-argument operations. The Altivec instruction set features an extensive choice of media operations, showing similarities with our TriMedia CPU64 instruction set.

3. Architecture

The goal for the new processor core was to achieve roughly 6 times the throughput of the first TM1000 operating at 100MHz by means of architectural changes and by utilizing new process technologies, while not increasing the transistor count beyond a factor of 2. A new process technology was estimated to allow almost a three fold increase in on-chip cpu clock frequency.

To achieve architectural speed-up, one could consider raising the number of issue-slots in the VLIW architecture. This track was however not followed for the following reasons:

- exploitation of more parallel slots through compiler detection of ILP becomes increasingly difficult;
- a single multi-ported register file and bypass network is convenient from a programming/compilation point of view, but becomes increasingly difficult to implement in terms of area and speed in the case of a larger number of ports.

The architectural speedup was achieved by different means:

- the wordsize was increased overall from 32 to 64 bits, to achieve more data throughput by exploiting SIMD-style (subword) parallelism;
- the instruction set was extended with a large set of media operations, to allow mapping of critical loops in relatively few operations;
- the data cache was improved to maintain a balanced design, since the cpu throughput increases significantly more than the external memory bandwidth.

Besides higher throughput, the new processor will have a memory management unit, providing virtual memory translation and inter-task memory protection.

Backward compatibility with the TM1000 (32-bit) family is achieved by recompiling the software. Care was taken to ensure that all media operations in such code are mapped with bit-wise identical behavior into the new (64-bit) instruction set.

A few architectural highlights will be treated in more detail in the next sections.

3.1. VLIW + SIMD

The TriMedia CPU64 architecture is a 5-slot VLIW machine, in principle launching a long instruction every clock cycle. It has a uniform 64-bit wordsize through all functional units, the register file, load/store units, on-chip highway and external memory. The 5 operations in a single instruction can in principle each read 2 register arguments and write one register result every clock cycle. In addition, each operation can be guarded with an optional (4th) register for conditional execution without branch penalty.

All functional units provide vector-style subword parallelism on byte, half-word, or word entities. This SIMD-style operation in each of the 5 slots in parallel allows for a very high media processing throughput. There is almost no support for arithmetic on 64-bit integers, 64-bit (double precision) floating point numbers, or 64-bit address ranges, since this was not considered important for the intended application area.

With the exception of floating point divide and square root, all functional units are pipelined, allowing a restart every cycle. The latencies vary from 1 (for operations like add, compare, bitand, bitshift, byteshuffle) to 4 (word multiply with round). A register-file bypass allows an operation result to be used as an argument for a next operation without having to wait for registerfile storage and retrieval. The overall cpu architecture is depicted in Figure 2.

3.2. Branch control

The C compiler generates assembly code which before scheduling consists of trees of basic blocks [7]. Every tree has a single entry point, forks for modeling ‘if-then-else’ constructs, and a jump operation for terminating each tree leaf. The instruction scheduler performs speculation by moving operations up in the tree. The scheduler can apply guarding on any operation with side effects, and/or generate additional intra-tree jumps.

To efficiently handle many jump operations in a single tree (and to reduce slot assignment constraints), there are 3 slots with branch units to allow scheduling of 3 conditional jump operations in a single instruction. In such cases the compiler must take care to ensure that at most one jump can actually be taken in any cycle. A jump operation has 3 branch delay cycles. The branch units are also pipelined, allowing a jump to be taken every cycle.

There is no provision for dynamic branch prediction. A compile-profile-compile cycle allows the C compiler and instruction scheduler to optimize the scheduling of the jumps and fill the branch-delay slots. These design choices result in a cpu execution control that is relatively simple in comparison with state-of-the-art superscalar processors.

Interrupt requests normally remain pending until an interruptible jump is taken. In that case, the original branch target address is saved, and a new target address is selected from a set of interrupt vectors. At these moments the number of alive register values is limited by the compiler, allowing a more efficient interrupt servicing.

Instruction compression is used to maintain reasonable code sizes despite having 5 issue slots with potentially 4 registers per slot, leading to instructions of variable size. In the cpu pipeline, one fetch stage is used for instruction decompression, accounting for one of the three branch delay cycles.

3.3. Instruction set

A large set of operations is implemented to support SIMD-style processing on smaller entities inside a 64-bit word. These operations are normally selected by the application programmer at the C language level by corresponding function calls, or in C++ by appropriate operator overloading. The instruction set covers most combinations of the following options:

- treating a 64-bit word as a vector of 8-, 16-, or 32-bit elements;
- providing two’s complement C-style wrap-around arithmetic or clipping against maximum and minimum integer values on the result;
- interpreting the data as signed or unsigned values, making a difference for operations that clip their results, and for operations which perform accurate rounding of least significant bits that would otherwise be lost;
- providing a complete set of operations, such as:
  add, subtract, min, max, abs, equal, greater, shift, multiply, type conversion, element shuffles, loads and stores.

There are moreover various operations specially geared to the signal-processing domain:

- multiplication operations which return only the most significant half of the multiply result;
- multiply-and-sum to perform an element-wise multiply, and summing together all products, to return a single integer with the ‘inner product’ value in full precision;

- sum-of-absolute-differences, determining the absolute values of element-wise differences, and summing them together for a single unsigned integer result;
- special vector shuffle operations for transpose operations on matrices (images), to support 2-dimensional filtering;
- look-up-table operations, where a single vector provides in parallel 4 short unsigned integers as indices to a table, and 4 new values are read from the table to constitute the resulting vector. This facility is extremely useful for many purposes such as color space conversion, intensity correction, etc.

Load/store operations are also provided in a wide variety. The semantics of a 64-bit load (or store) differs for the different vector element sizes (byte, half-word, word) because the cpu supports bi-endian operation: the vector support is designed to match C-style arrays, where increasing indices map to increasing memory addresses. As result, the order of the vector elements in memory is fixed, although the byte order within a (half-word or word) element depends upon the endian setting. Vector stores are also provided in a 'masked' form, providing write-enable bits for each vector element. In addition to the vector load/stores, single element (scalar) load/stores are provided. Such loads differ for signed versus unsigned loads due to sign extension. Table 1 presents an overview of the number of operations implemented in different categories.

Table 1. Instruction set overview

type of operation	number
loads and stores	39
byte shuffles	67
bit shifts	48
multiplies	54
integer ALU ops	104
floating point ops	59
branch ops	10
table lookup ops	6
special register ops	23
Total	410

We found that by rewriting some critical inner loops of the tested applications, we could implement their functionality in about half the number of operations used with optimized TM1000 source code. Section 5 presents the results of an implementation of an 'IDCT' algorithm, showing the strength of the instruction set in comparison with other processors.

3.4. Multi-slot operations

To allow more powerful media operations, more than 2 arguments and/or more than one result are desirable. This is why multi-slot operations ('super-ops') are incorporated. They occupy 2 neighboring slots in the instruction format, and map to a double-width functional unit in the architecture. This way the 'super-op' neatly fits in the existing instruction format, fits the existing connectivity structure to the register file, and hence requires very little hardware overhead. Examples of such operations are:

- a 'transpose' operation, requiring 4 argument vectors and producing 2 result vectors, to perform (an upper or lower half of) a matrix transposition;
- an 'average' operation of 4 argument vectors to obtain an accurate average value of the arguments, preventing the rounding errors in intermediate results obtained with a sequence of 2-way average operations.

Figure 2 shows the resulting datapath architecture. Two instances of functional units which can accommodate two-slot super-ops are shown among a larger set of normal single-slot functional units. (The constellation of functional units in the figure is given only as an example; the number of units in the real design will be different.)

3.5. Cache and MMU design

The new data cache maintains the 16KB size of the TM1000, the 64B cache blocks in an 8-way set associative scheme, and a fetch-on-write and read critical-word-first policy. Simulations have indicated that doubling the cache size hardly helps in reducing the miss rates in the selected media processing applications. This is due mainly to their streaming data nature, characterized by large data sets with little re-use. We obtained a better cache performance by using:

- a true dual-port cache design to fit the two load/store units in the VLIW data path.

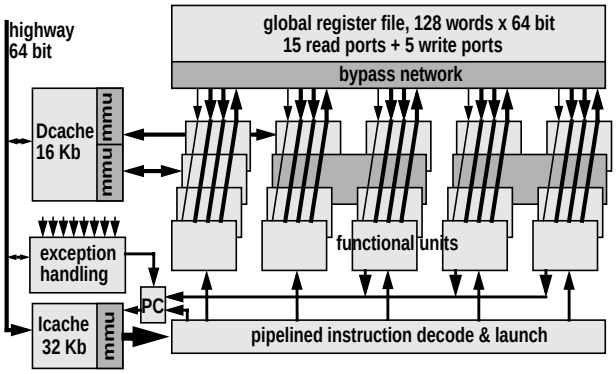

Figure 2. Datapath architecture

The TM1000 cache was a quasi-dual port design. Two load/store requests could only be served in parallel if they happened to access different banks. Otherwise, a stall cycle was imposed on the entire VLIW datapath, allowing serialization of the two requests. This penalty is removed by true dual r/w ports;

- a (more powerful) set of operations to control cache behavior: a 'prefetch' request to indicate the data to be fetched from memory, a 'victimize' request to indicate cache blocks which are no longer needed, and an 'alloc' request to suppress the normal fetch-on-write when an entire block will be written.

New memory management units provide 32-bit virtual-to-physical address translation and paged access protection. The dual-ported d-mmu is truly separate from the single-ported i-mmu. One design objective was to provide seamless co-existence with MIPS processors in the same system, which influenced the mmu design choices. The resulting d-mmu has:

- variable page sizes, in powers of 4, 4kB to 16MB;
- 64 entry fully associative TLB, software managed;
- page descriptors having i.a. an 8-bit task ID and a write-enable bit.

In addition, the upper 3 bits of the 32-bit virtual address choose different memory segments with supervisor mode only, unmapped, or non-cached properties.

Supporting variable page sizes is deemed necessary for obtaining good TLB hit rates, in particular when an application program runs quickly through large chunks of data, as is the case in video image processing. Simulations were performed in an attempt to automatically generate large-page table entries from the application 'malloc( )' requests with good results. Figure 3 shows simulation results obtained with a high-definition video processing application. The vertical axis presents the tlb miss-rate in % of load/store operations. TLB sizes are shown from 16 entries to 128 entries. The adjacent columns show three page-size architectures: variable page sizes of 4Kb and up with powers of 4, 4Kb uniform, and 16Kb uniform.

3.6. Precise exceptions

Software-managed TLBs require the capability of the cpu to handle precise exceptions. Precise exceptions are furthermore useful for real-time interrupt response behavior, and for software debugging. Designing the exception mechanism was a challenge due to the combination of:

- a pipelined VLIW design, with a potentially large number of simultaneously active functional units, each having a different latency;

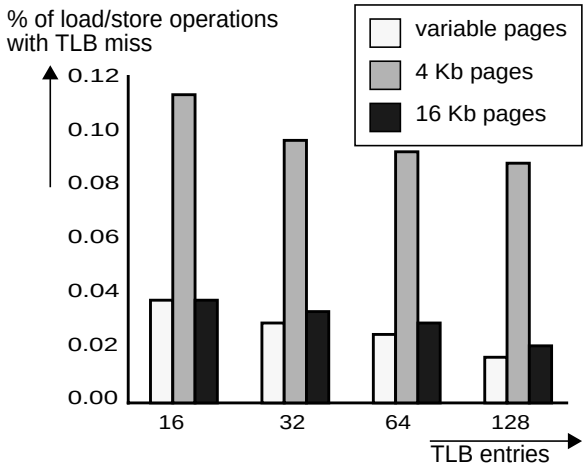

Figure 3. TLB miss rate simulation

- the possibility of a caught exception being overridden by another exception occurring later in time from an earlier instruction in a later pipeline stage;
- having multiple jumps active in the pipeline, which complicates the recovery at exception return;
- a supervisor/user mode protection mechanism, with several such modes simultaneously active in the instruction pipeline, since such mode changes occur with jumps.

The method implemented to handle this is still relatively straightforward: the pipeline is emptied by executing some more clock ticks, while saving the last few program counter values with some extra status bits. On return from the exception, they are restored into the pipeline through a sequence of jumps. Besides for TLB misses, precise exceptions can be generated for opcode errors, privilege violations, debugging or high-priority timer events.

4. Performance evaluation

A basic set of tools was built to evaluate different design options. An available C/C++ compiler was adapted to support 64-bit vector types, and a scheduler for the TM1000 was adapted to handle new constraints and features of the CPU64. In addition, a linker, loader, assembler and disassembler were made. The resulting code can be executed by the newly developed simulator [8]. The simulator has relatively high-level, C language, functional models, hooked into a cycle-based simulation kernel. This allows functional and cycle-correct simulations of the cpu with caches and mmus, on-chip highway, and external memory. Finally, simulation runs allow to exercise the compile-profile-compile loop, needed for good compiler optimization.

As an example of the power of the combined approach of VLIW, subword parallelism, and an extensive set of operations, the IDCT computation is an interesting bench-

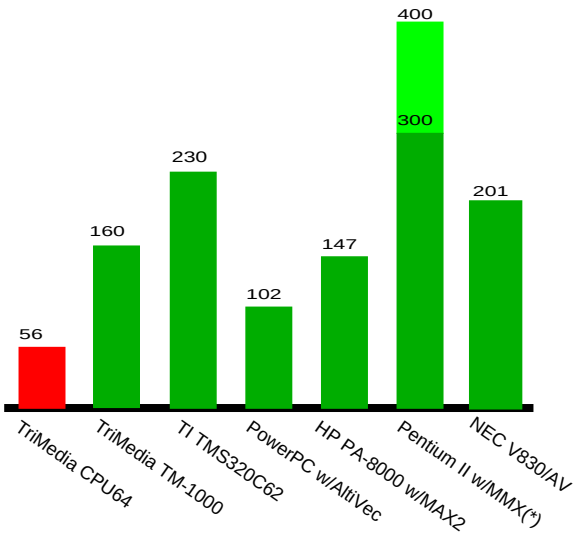

Figure 4. cycles for 2D-IDCT

mark from the targeted application domain. The IDCT is coded with the 'Loeffler' algorithm [9], using a function call syntax to select specific media operations. The resulting C source code is compiled with the developed tool chain and executed in the simulator. The result showed compliancy with the IEEE numerical accuracy requirements. The IDCT code is compiled and scheduled into 56 VLIW instructions, including function call/return overhead, loading the 8x8 coefficients, performing the 2-dimensional IDCT, normalization of the end result, and storing the 8x8 pixels back in memory. All computations are done with 16-bit values, which proved sufficient for achieving accuracy compliancy in our implementation.

Figure 4 compares our results with results reported by others. For the Pentium-II with MMX(*) Intel reports a value of 500 cycles for a compliant IDCT including additional arithmetic for quantization [10]. Since quantization is a less complex calculation, we estimate this IDCT to be between 300 and 400 cycles. Alternatively, a comparison can be made with Intel's reporting of 240 cycles with a faster non-compliant algorithm for a Pentium-I with MMX [11]. Note furthermore that the figures for (at least) TI and Intel were obtained through assembly programming.

As far as the numbers listed in Figure 4 are concerned, the TriMedia CPU64, the PentiumII, and the V830 refer to an accuracy compliant implementation. Compliancy is not claimed for the other results, so computationally cheaper algorithms were probably used [12][13].

Figure 4 has been included for comparison of instruction sets, and is not suited for performance comparisons. The CPU64 is still in development, whereas the other processors are already in production. Moreover, clock frequency differences are significant.

As a larger driver application, a program was used which performs video de-interlacing using a motion estimation and compensation approach. This program contains more than 10,000 lines of C code, and also has a considerable amount of code which is scalar/sequential by nature. With this program a speedup over the TM1000 was obtained of a factor of 7.8. Taking into account the tripled cpu clock frequency, the doubled data path width, and the relatively slower external memory, this performance gain shows a good overall efficiency.

5. Conclusion

A new processor architecture has been developed as successor to the TriMedia TM1000. Initial goals of 6x performance gain were roughly met: well vectorized inner loops show a significantly higher speedup, while the speedup for scalar/sequential C code approximately equals the targeted clock frequency increase, i.e., a factor of 3.

The power of VLIW parallelism in combination with subword parallelism and an extensive instruction set have been demonstrated with an extremely compact IDCT function. The good performance on vectorizable inner loops is achieved by optimizing C source code, requiring the use of vector data types and function calls for media operations. Manual optimization of critical sections in the source code is accepted for embedded use in high-volume applications, which has led to excellent performance in media processing tasks on a processor with wide data paths and relatively simple cpu control.

6. References

[1] S. Rathnam, G. Slavenburg, "An architectural overview of the programmable multimedia processor, TM-1", *proc. Compcon '96*, Santa Clara CA, pp. 319-326. Feb. 1996

[2] P. Faraboschi, G. Desoli, J.A. Fisher, "The Latest Word in Digital and Media processing", *IEEE Signal Processing Mag*., pp. 59-85. March 1998.

[3] Equator Technologies Inc.: A backgrounder, http://www.equator.com/equator/html/Press_Release/981201bg.pdf

[4] N. Seshan, "High VelociTI Processing", *IEEE Signal Processing Mag*., pp. 86-101. March 1998.

[5] K. Suzuki, T. Arai, et.al., V830R/AV: Embedded Multimedia Superscalar RISC processor, *IEEE Micro*, pp. 36-47. March 1998.

[6] M. Phillip, "Altivec technology: A Second Generation SIMD Microprocessor Architecture", *HOT Chips 10 symposium*, pp. 111-121, August 1998

[7] J. Hoogerbrugge, A. Augusteijn, "Instruction Scheduling for Trimedia", *J. for Instruction-Level Parallelism*, Vol. 1 (to appear), 1999

[8] E.J.D. Pol, J.T.J. v. Eijndhoven, B. Aarts et al, “TriMedia CPU64 application development environment”, *these proceedings.*

[9] C. Loeffler, A. Ligtenberg, G. Moschytz, “Practical Fast 1-D DCT Algorithms with 11 Multiplications”, *proc. Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP ’89)*, pp. 988-991. 1989

[10] Intel Pentium II processor Application notes, “JPEG Inverse DCT and Dequantization Optimized for Pentium II processor”, http://www.intel.com/drg/pentiumII/appnotes/886.htm

[11] Intel MMX technology application notes, “Using MMX Instructions in a Fast iDCT Algorithm for MPEG Decoding”, http://www.intel.com/drg/mmx/appnotes/ap528.htm

[12] PowerPC: Altivec Performance Table, 1997. http://developer.apple.com/hardware/altivec/performance-Table.html

[13] R. Lee, “Effectiveness of the MAX-2 Multimedia Extensions for PA-RISC 2.0 processors”, *HotChips IX symposium*, pp. 135-148. Aug. 1997.