Reprinted from proceedings of ICCD 1999, International Conference on Computer Design, October 10-13, 1999, Austin Texas, pp. 580-585.

TriMedia CPU64 Application Domain and Benchmark Suite

A.K. Riemens, K.A. Vissers, R.J. Schutten, F.W. Sijstermans*, G.J. Hekstra, G.D. La Hei
Philips Research Labs, Eindhoven, The Netherlands
**Philips Semiconductors TriMedia, Sunnyvale CA, USA*
riemens@natlab.research.philips.com

Abstract

At Philips Research Labs, we are investigating the 64-bit VLIW core (also called CPU64) for future TriMedia processors. This processor is targeted towards embedded multimedia applications. In order to be able to perform a quantitative design space exploration, a set of benchmark applications has been developed which is representative of the application domain. This article describes the way the benchmark set was developed, as well as providing a more detailed description of one of the benchmarks. As a result, it is shown that this new cpu has a high performance for the application domain while the programming interface is very convenient. Furthermore, such a programmable media processor may offer new options and challenges to algorithm designers.

1. Introduction

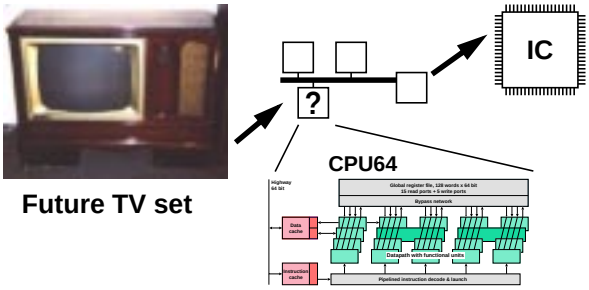

Figure 1. The design problem

Philips Electronics designs and manufactures a wide range of products for the consumer electronics markets. These products are becoming increasingly complex due to ever more stringent quality requirements and increasing functionality. Furthermore, due to price erosion in these markets, the cost of the bill of materials and manufacturing put very tight constrains on the design of such systems. For example, the DVD standard not only supports the playback of a prerecorded movie, but also allows much background information to be stored on the DVD disk. This information consists of text, images etc. So, a DVD player has to process various information streams: the movie itself, various video and graphics objects and the control data that structures the background information. Another example is the future television or set-top box market, where connectivity and interactivity are becoming increasingly important and digital transmission has started. For example, a future TV might be connected to the Internet and, apart from superb video processing, it needs to run a web browser and support a wide variety of graphics and (both analog and digital) video formats.

From these examples it should be clear that future products have to be able to deal with *video, audio, graphics* and *communication* data in a flexible way. In order to facilitate this, a 64 bit VLIW *embedded media processor* has been architected at Philips Research Labs. It is called CPU64.

Philips TriMedia is a product group which currently has the TM1000/TM1100 in the market. One of the building blocks in these products is a 32 bit, 5 issue slot VLIW cpu (see [1], [2]). The development of CPU64 was started from the TM1000 series architecture and was strongly influenced by the knowledge and experience that was gained during the development of these products.

This article is one of four papers describing the complete design of CPU64. The accompanying papers describe the cpu architecture [3], the programming environment [4] and finally the detailed design space exploration [5]. In this article we describe how a benchmark suite has been developed to facilitate the design of the processor. Furthermore, a particular benchmark in the field of video signal processing is described in more detail, including initial performance measurements based on simulations and a comparison with existing technology. Section 2 defines the problem and introduces the "layered natural motion" benchmark. This benchmark is used as example throughout the paper. Section 3 describes the solution approach; it introduces the benchmark characteristics and the benchmark development process. Section 4 shows the results, and is followed by a discussion on these results, conclusions and future work.

2. Formulation of problem

The application domain of the processor imposes strict requirements on all resources that have to be available. Let us consider applications in the field of video signal processing. Standard resolution images contain approximately 350 Kbyte pixel data, at a picture rate of 60 Hz. This causes a data throughput of 20 Mbyte/s. High resolution television signals may require 6 times this amount. Novel signal processing algorithms often access a number of pictures and require several 100 operations per pixel. This results in a memory bandwidth of hundreds of Mbyte/s and a cpu load of many GOPS.

Traditionally, there is a strict partitioning between control processing and signal processing in a system. Control processing includes the hardware control, user interface, etc. This is implemented using microcontrollers or embedded RISC processors. The signal processing, on the other hand, is implemented in dedicated ICs. These are derived using the conventional method of a uni-directional path from the algorithm into the low level IC design. High-level synthesis tools can be used to improve this design process.

For novel media processing products, this strict partitioning between programmable control and hardwired signal processing is fading. Some media processing will be done on a programmable cpu, to increase flexibility and facilitate advanced connectivity in the systems. Apart from the cpu, much of the media processing will be implemented in flexible, but function-specific hardware units. In this way, the system architecture will become increasingly heterogeneous. In this article, we will focus on the design of the media processing cpu.

Given the application domain of the processor, the major design considerations are determined by:

- cost, since the processor will be embedded in high volume electronics products
- performance *for the application domain*
- programming model and interface

In order to be able to asses the performance, a benchmark suite needs to be available. This leads to the formulation of the first problem: *define (and develop) a benchmark suite which is representative of the given application domain*. Secondly, the *first order design choices* have to be made based on in-depth knowledge of the application domain and current solutions. These high-level design choices will also be addressed in this paper. And finally, we will use the “layered natural motion” benchmark as example that is studied in more detail.

“Layered natural motion” is an application in the field of video signal processing ([6]). It performs picture rate conversion. When a movie is recorded at the studios, usually only 24 pictures per second are available. When it is transmitted as a television signal, these pictures are repeated to generate 50 or 60 images per second. This repetition process causes motion artifacts. The “layered natural motion” program replaces the repeated images by new images that have to be calculated. In order to calculate these images properly, motion information is required. Therefore, the program performs motion estimation on the original images and applies this motion information in the calculation of the new images. In this way, the motion artifacts are reduced.

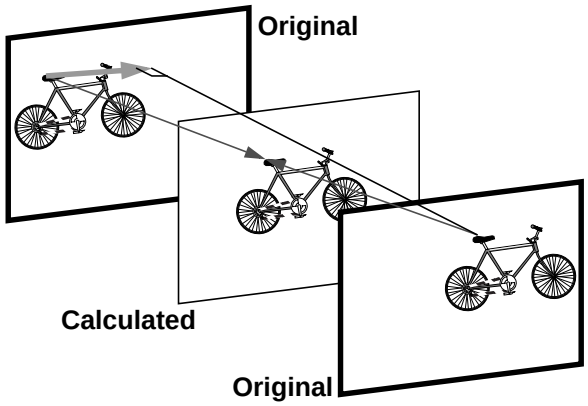

Figure 2. Motion estimation/compensation

3. Our solution approach

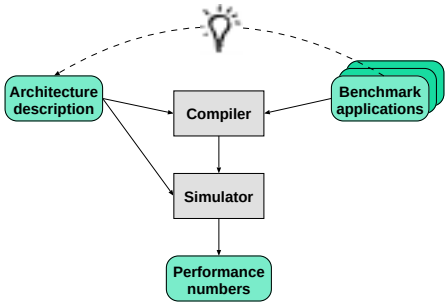

Figure 3. Y-chart

We are convinced that only those processors that have an easy-to-use and high-level programming interface will survive in the market place. For this reason, only those design options that can be supported by an efficient and optimizing compiler are considered useful. This statement was never compromised, neither to reduce cost, nor to increase the performance. The real challenge is to design the processor in such a way that all the characteristics, like cost-effectiveness, performance and ease of use, are combined in a clean, coherent design. For this reason, a multi-disciplinary team of people was formed, consisting of specialists in the application domain, cpu architects, compiler experts and software engineers to build the required tools.

The Y-chart describes the methodology. A retargetable C/C++ compiler is used to program the processor. A retargetable simulator can simulate the behavior of the benchmark applications on the proposed processor and generates detailed profile information. By modifying the machine description, various design options can be explored and analyzed. See also [7].

3.1. Benchmark set characteristics

Ideally, when designing a processor, one needs *all* applications to tune the system. This is impossible. The amount of effort would be tremendous, and many applications are yet unknown. In order to overcome these difficulties, the benchmark suite is used that must comply with the following characteristics.

- Each benchmark shows processing characteristics typical of a class of applications within the application domain.
- The set covers a significant area of the application domain.
- Each benchmark is sufficiently well tuned (optimized) to the architecture to allow performance evaluation of the architecture.

Table 1. The benchmark suite

Category	Application
Data communication	Viterbi decoding
Audio coding	AC3 decode
Video coding	MPEG2 encode
	DVC decode
Video processing	Layered natural motion
	Dynamic noise reduction
	Peaking
Graphics	3D renderer back-end
	Mesa OpenGL library

Note further that any benchmark set is a compromise between sufficiently accurate modeling of the application domain and the amount of effort required to develop the suite.

An overview of the benchmarks is given in Table 1. Most of these applications were developed in our research labs, while for some of them we were assisted by other departments within Philips. Various characteristics can be identified in this set of applications.

- Various signal rates: ranging from audio to video signals at block rate and at pixel rate.
- Various basic data types.
  - Bytes (8 bit) in the video pixel domain.
  - Halfwords (16 bit) in the audio domain and as intermediate video data.
  - Words (32 bit) in all control processing.
  - Floating point in the viewpoint, lighting and perspective transformations of the mesa library.
- Various data access patterns.
  - Straightforward stream processing at the signal sample rate (peaking, dnr, parts of layered natural motion).
  - Data compression/decompression, where a stream of data is transformed into/from a bitstream (ac3, mpeg2, dvc).
  - Random access of blocks of video data (the motion estimator of layered natural motion, mpeg2).
- Both data content independent (viterbi, dnr, peaking) and data content dependent load.
- Both control processing (header analysis/generation in the compression/decompression applications) and signal processing.

Especially for the applications where the cpu load is data dependent, special care was taken to design a data set which is sufficiently representative as well as sufficiently small in order to avoid excessive simulation times.

3.2. Benchmark development

The development of CPU64 did not start from scratch. It is strongly founded on the existing TM1000 product range of the Philips TriMedia product group. Therefore, we could also profit from applications which were originally designed for these existing products. Figure 4 shows the development trajectory. We have experienced that there is a very interesting playing field in the area of algorithm design and architecture design. On one hand, the architecture can be tuned towards the algorithm. On the other hand, on a given architecture, one can design novel algorithms to optimally utilize the machine.

The “layered natural motion” application is a typical example. Up till now, motion estimation/compensation functions for the consumer electronics market could only be implemented in dedicated IC designs (see [8]). In order to execute in real-time on a TM1000 series processor, the number of operations per pixel had to be reduced *by an order of magnitude*. One step to achieve this, was a limited reduction of the image quality. But much more important has been the design of a complete new motion estimator algorithm (see [6]).

An algorithm is described in “reference C”. This is C code which precisely describes the functionality of the algorithm, without containing any machine specifics. This will be used during the algorithm development, to facilitate e.g. functional video simulations that can be used to assess

the signal quality after the algorithm has modified the signal. When initial reference code is available, a machine specific optimization is required. During this stage, various methods are applied to increase the execution speed on the target machine. Specific instructions ("custom operations" or "vector instructions") will be applied to benefit from data parallelism (single instruction, multiple data: SIMD). Use of these instructions requires modification of the program. In some situations, we have seen that minor functional modifications might result in significant performance improvements. In such cases, the reference code also needs to be modified, and functional simulations are required to validate the algorithm change. Furthermore, various (manual or compiler) techniques are applied to achieve sufficient instruction parallelism (ILP). These techniques include function inlining, software pipelining, loop fusion, loop unrolling, etc. An average ILP of more than 4 for a 5 issue slot VLIW can usually be achieved with a reasonable amount of effort.

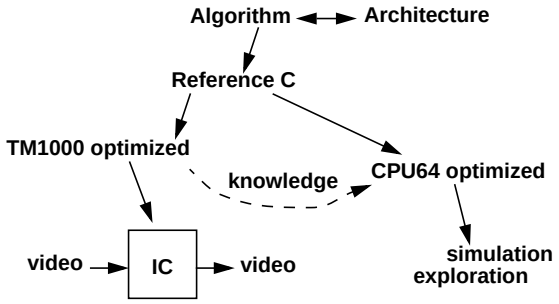

Figure 4. Benchmark development

This optimization process for the TM1000 series processor has been carried out for the layered natural motion program. The reference code consists of approximately 15,000 lines of C code. More than 95% of the CPU load (after optimization), however, is caused by approximately 3,500 lines of code. This part of the program contains the critical signal processing functions, and only for this part is the software optimization required.

For CPU64, a similar optimization process is carried out. A careful analysis of the existing designs and application programs led to an initial design of CPU64 and its instruction set. Once the retargetable compiler and simulator were available, the optimization for this new machine was carried out.

In contrast to the method described in this paper, the design process of general purpose computers is usually driven by a set of fixed and predefined benchmarks, like the "specint" benchmarks. This results in a design process where only the architecture is tuned towards the applications. In our approach, we tune both the application set and the architecture towards each other. There are two forms of application tuning. In some situations, the functionality can be adapted to the architecture (e.g. with video enhancement functions). In other cases, the functionality is fixed, but various optimization strategies can be applied (e.g. for mpeg2 decoding).

3.3. Initial design stages

Within Philips, considerable expertise in VLIW architectures and the associated compiler technology is available. VLIW architectures enable high-performance processors, utilizing instruction parallelism. This is combined with a relatively straightforward execution pipe structure. For the next design, the main goal was a significant performance improvement in the order of 6 to 8 times over the TM1000 series, combined with a limited complexity increase. It is expected that progress in IC technology and a new design of the processor, will result in an increased clock speed of a factor of 3. Further performance improvement has to be realized by an improved architecture. For several reasons we decided to build again a VLIW architecture.

- Both the instruction parallelism and the data parallelism that is available in the application domain can be utilized.
- Limited complexity, thus well suited for an embedded processor.
- Significant parts of the existing programming tools could be reused.
- It fits well in the current product line.

Compatibility with the existing products is achieved through recompilation. This enabled the design of a new and significantly improved instruction set.

Initial calculations showed that performance improvements can be achieved by a uniform 64 bit design.

- Doubling the vector length from 32 to 64 bit gives a performance gain of 2 for those programs that can be vectorized well. This is true for all media processing, though sometimes a small penalty has to be paid for extra data formatting. Control processing will not benefit from this increased word size.
- An enriched media instruction set causes the applications to perform the same functionality with fewer instructions.
- The uniform register file avoids potential complexity in the programming tools.

Special attention was paid to the data cache. A traditional data cache may not work well for media processing. In this application domain, very often large consecutively stored data items are read or written only once. Therefore local storage of the data in the cache will not reduce the bandwidth to the background memory nor will it reduce cpu

stalls caused by the data cache while it is fetching the data from the memory. However, local storage combined with *prefetching* of the media data can be used to avoid cpu stalls due to the memory latency. A number of high-level quantitative experiments showed that a single data cache with additional prefetch and copyback support is a good solution.

A careful study and analysis of the application domain has resulted in an initial instruction set. Instruction set design is a matter of careful balancing between various trade-offs. Considerations are the following.

- An instruction should be sufficiently generic to avoid being useful for only one single application.
- There should be sufficiently powerful instructions to allow efficient coding of the application.
- Consistency and orthogonality. Make the same instructions available for the different data/vector types.
- Keep the number of instructions limited to control the design complexity.

During the instruction set design, many carefully chosen "corner stone" experiments were carried out to verify and validate different instruction choices. Often specific "what if" questions could be quantified by small modifications of one of the benchmarks. To prove the value of proposed instructions, the modified benchmark was compiled with and without the new instruction. Simulation runs then give a clear insight into the performance consequences.

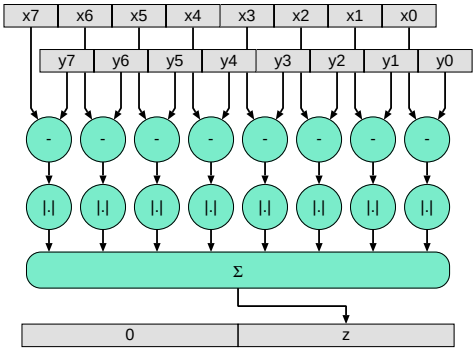

Figure 5. Vector SAD instruction ("ub_me")

From the very start of the processor design, the programming model was taken into consideration. First of all, it is crucial that the compiler is completely ANSI-C or C++ compliant. We did not deviate from these standards, nor did we extend them. Specific vector types are built into the compiler to achieve a high level of readability and to facilitate strong typing to improve the code quality. Furthermore, the special media instructions are built into the compiler via function call semantics. These function calls are mapped into machine instructions by the compiler.

As an example of a media instruction, Figure 5 shows the calculation of the sum of absolute differences (SAD) of two vectors. This is a typical instruction used in the motion estimator part of the layered natural motion benchmark. In Figure 6 a C function is shown that calculates the sad between two blocks of 8x8 pixels video data. This example clearly illustrates the power and ease of use of this programming model.

```
int calc_sad(vec64ub *prv, vec64ub *cur, int stride)
{
    vec64ub left, right;
    int i, sad;
    sad = 0;
    for (i=0; i<8; i++) {
        left = prv[i*stride];
        right = cur[i*stride];
        sad += ub_me(left, right);
    }
    return(sad);
}
```

Figure 6. Application program using vector sad

4. Results

The described optimization method was applied on the "layered natural motion" program on the performance-critical parts. This was done, both for the TM1000 series processor and for the initial design of CPU64. The resulting profile information, measured in cpu cycles, on a typical test sequence of 113 standard resolution images is shown in Figure 7.

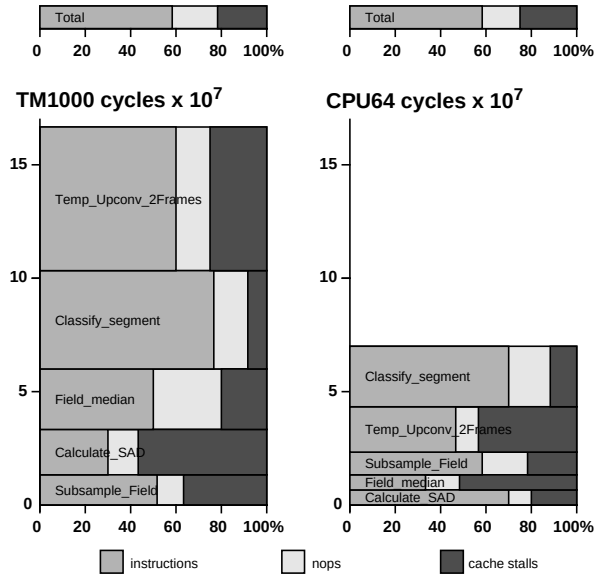

Figure 7. Proview of TM1000 and CPU64

Figure 8 shows the cpu load of the program on both processors for each of the input fields. In this figure, the increased clock frequency is taken into account. The differ-

ent load patterns are caused by the content of the input video stream. This sequence contains both video and different film modes in order to be statistically representative. Note that the performance improvement does *not* scale linear: some functions benefit more from the new instruction set than others. In the CPU64 design, all functions map relatively well on the instruction set.

Similar developments are done for the complete benchmark suite. Based on these applications and the initial design, a consistent, tuned instruction set and initial cpu architecture has been designed. And subsequently a highly automated design space exploration has been carried out (see [5]).

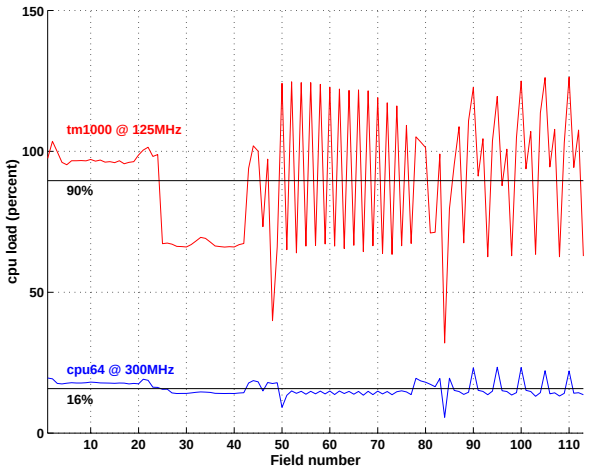

Figure 8. Cpu load of TM1000 and CPU64

5. Discussion and related work

In consumer electronics applications, the typical trade-off is cost versus quality. Cost should be low, while image or sound quality should be very high. In the design of a cpu for this market, the cost is related to silicon area. But when designing a function for such a cpu, cost is measured in cpu cycles. The amount of cpu cycles needed for a certain function depends very much on the instruction set. Tuning of the algorithm towards cost means tuning it to the instruction set of the cpu. This means that instruction set design is crucial in the processor design. In order to design the instruction set, in-depth understanding of the typical applications is required. The instruction set should be sufficiently specific for the applications, while being sufficiently generic to avoid an explosion of the number of instructions.

The way of working has been to develop a number of benchmark applications in detail. After that, to analyze the required instructions and, based on the obtained knowledge, to generalize the instruction set to make it suitable for the *application domain*.

6. Conclusions and future research

For new products, media processing will become increasingly important. Therefore, a high-performance embedded media processor will be a crucial component in these products.

It is shown how we developed a multimedia benchmark suite, and used it for the initial design of the processor. Furthermore, the high-level programming interface, which contains an advanced vector programming model, improves the ease of use of the processor significantly.

The availability of high-performance media processors allows functions to be implemented in software, rather than hardwired. For future function implementations, the trade-off between the hardwired part and the programmable part will gradually shift into more programmable. This will have a number of consequences for future systems. First of all, this trend automatically implies that future systems will become more heterogeneous. System level design using quantitative methods is still an area where much research is required. Secondly, this trend will both add new challenges and offer new opportunities to function and algorithm designers. As a generic trend, one might use the processor to analyze higher level characteristics of the signals. As a specific example, the “layered natural motion” algorithm can be improved by employing a more advanced modeling of the motion in the scene. This kind of higher level analysis might open a whole new field of algorithm research.

7. 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] A.K. Riemens, R.J. Schutten, K.A. Vissers, “High speed video de-interlacing with a programmable TriMedia VLIW core”, *International Conference on Signal Processign Applications and Technology*, Sept. 1997, San Diego, CA, USA, pp.1375-1380.

[3] J.T.J. v. Eijndhoven et al., “TriMedia CPU64 Architecture”, *these proceedings*.

[4] E.J.D. Pol et al, “TriMedia CPU64 Application Development Environment”, *these proceedings*.

[5] G.J. Hekstra, G.D. La Hei, P. Bingley, “TriMedia CPU64 Design Space Exploration”, *these proceedings*.

[6] R.J. Schutten, G. de Haan, “Real-time 2-3 pull-down elimination applying motion estimation/compensation in a programmable device”, *IEEE Transaction on Consumer Electronics*, Volume 44, Number 3, August 1988, pp. 930-938.

[7] John L. Hennessy and David A.Patterson, *Computer Architecture: A Quantitative Approach*, 1995, Morgan Kaufmann Publishers, ISBN 1-55860-329-8

[8] G. de Haan, “IC for motion compensated de-interlacing, noise reduction, and picture rate conversion”, *IEEE International Conference on Consumer Electronics, June 1999*, pp. 212-213.