Appeared in the Proceedings of ICSPAT'96, Boston, October 1996

On the Integration of DSP Hardware into a Microkernel-based Operating System *

Robert Baumgartl, Hermann Härtig

Department of Computer Science
Dresden University of Technology, Germany
{Robert.Baumgartl, Hermann.Haertig}@inf.tu-dresden.de

Abstract

Two technological developments reraise the debate about the use of DSPs as accelerators for certain applications vs. the employment of standard off-the-shelf microprocessors:

- Tremendous development investments make general-purpose microprocessors even more powerful by the use of superscalar execution and sophisticated caching structures.
- Microkernel-based operating system architectures permit very flexible decomposition of application systems due to their efficient message passing capabilities.

Hence, this paper reports on two experiments. First, some elementary algorithms are run on a DSP and a modern superscalar microprocessor. Second, a DSP board is integrated into the message passing scheme of a modern microkernel-based operating system and a more complicated application is run on a DSP and evaluated.

1 Introduction

Multimedia applications introduce at least two problems:

- The need for high computing power and transfer bandwidth between CPU and peripheral devices.
- The need to meet realtime requirements.

To overcome these problems two approaches exist.

*This work was supported by the DFG with the Graduiertenkolleg "Tools for the effective Use of Parallel and Distributed Computing Systems" and the SFB 358.

The first one is to use an even more powerful CPU. The Native Signal Processing (NSP) initiative proposes to run all multimedia algorithms on high-end general-purpose microprocessors [1]. However, it has been shown that only a limited range of multimedia applications can be performed by today's CPUs under widely used standard operating systems [2].

The second approach is the integration of a specialized cheap coprocessor into the computer architecture. A coprocessor relieves the CPU from computationally intensive functions and directly controls peripheral devices. However, several disadvantages are frequently attributed to this approach:

1. Overhead is introduced by inter–processor communication.
2. The subsystem has a complex programming model.
3. At least two software development environments are needed.
4. The subsystem is inflexible as a consequence of specialization of the coprocessor.

The objective of our project is to quantitatively analyze and to overcome these disadvantages. To this end a fixed-point DSP board is integrated into a modern microkernel-based operating system. Special attention is paid to reduce communication overhead and to provide an easy-to-use programming model for the DSP.

This paper describes the results of the first step of the project. It demonstrates the qualification of DSPs as multimedia accelerators. A first attempt of a comparison between general-purpose processors and DSPs is made. Potential performance bottlenecks are identified and discussed.

The paper is organized as follows: Section 2 describes briefly the used hardware architecture. In section 3 several elementary algorithms are compared, and their performance on the respective processor is discussed. Section 4 introduces and discusses a JPEG decoder as an example of a more complex algorithm. Finally, section 5 summarizes the results and gives a short outlook on future work.

2 The Testbed

All tests were done on a PCI P60 PC equipped with 256 kBytes 2nd level cache and 8 MBytes RAM. A Motorola DSP56001/20 [5] resides on a standard ISA plug-in board equipped with $3 \times$ 32kWords static RAM. The two processors communicate via the host interface of the DSP, which is mapped into the port address space of the PC.

As a flexible yet powerful PC operating system VSTa [6] was chosen. It features a modern microkernel architecture as well as reasonable but not excellent performance.

3 Simple Algorithms

The first part of our analysis dealt with simple algorithms such as string comparison, vector addition and digital filters.

3.1 Method of Measurement

All algorithms were implemented and optimized in processor-specific assembly language. Then execution times of the resulting programs were determined, compared and analyzed. Finally, general criteria for the decision, whether or not a particular algorithm is suited for the processor architecture, were derived.

During the optimizing stage we tried in particular to maximize parallelism on the DSP, utilized the powerful multiply-and-accumulate instructions as often as possible and kept operands in registers to reduce load/store instructions. The Pentium code was optimized in a regular fashion, but no special attempt was made to fully exploit the superscalar integer pipeline. Data and Instruction caches were used.

We considered code generation with compilers as inappropriate for our purposes, because we wanted to compare the raw processing abilities of the processors. Of course, almost all Pentium software is developed using a compiler, whereas for DSP programming assembly language is much more common, but we considered the usage of a compiler for Pentium code generation as an unacceptable performance penalty. Besides that, the overhead introduced by DSP compilers is reported to be relatively big compared to general-purpose CPU compilers [9].

Complex performance-enhancing components used in the Pentium CPU, such as caches, branch prediction unit or deep processor pipelines make it virtually impossible to analytically determine the exact number of clock cycles for the execution of a given fragment of machine code. On the other hand it is easy to calculate clock cycle numbers for DSP 56001 machine code for the processor's comparatively simple structure. Therefore, two different approaches for the determination of the execution times were employed.

DSP execution times were determined by counting clock cycles as described in [5]. As a result it was also possible to calculate the execution times for faster versions of the DSP. We did this for a hypothetical version of the DSP clocked at 60 MHz to compare processors with identical clock rate.

Pentium execution times were measured using so-called Model Specific Registers (MSRs) which can deliver a substantial amount of information about the executed code [4]. A program wrapper stores the respective cycle clock counter before and after execution of the code fragment to be examined. First and second level processor caches are invalidated immediately before execution of the analyzed code. By calculating the difference between the stored counter values it is possible to obtain exact cycle counts for Pentium code. For longer execution times the measured values still differ, so an average value over at least 5 independent measurements was taken. All Pentium measurements were done in Real Mode using MS-DOS.

The subsequent sections of this chapter each describe briefly the individual algorithm, present the execution times of the respective processor and discuss the measured results.

3.2 String Comparison

The first task to be analyzed is string comparison. The algorithm simply subtracts the corresponding bytes until a nonzero result is obtained. It is assumed that the strings differ at an arbitrary position.

Table 1 shows the resulting execution times. The first different character in the strings is denoted by n, all times are given in microseconds.

The program for the Pentium processor is very short. After initialization only the instruction `repz cmpsb` is executed to perform the desired operation. On the other hand, the instruction cannot use the Pentium's superscalar structure. The initialization is relatively expensive.

n	56001/20	56001/60	P60
1	1.3	0.43	0.92
5	2.9	0.97	1.18
10	4.9	1.63	1.83
100	40.9	13.6	9.4
1000	401	134	88
10000	4001	1334	956

Table 1: Execution times for string comparison

Because both strings reside in the same memory space, the Harvard structure of the DSP cannot be utilized fully. The restrictions concerning the position of a premature loop end furthermore prevent a fast implementation. The DSP needs $18+8n$ clock cycles for the completion of the task. Obviously, the DSP is slightly inferior to the Pentium for this kind of algorithm.

3.3 Vector Addition

The next problem to be investigated is the addition of two vectors with n integer components. The size of the individual components is 16 (Pentium) and 24 (DSP 56001) bits respectively.

Assuming one register is used to address all operands, the operation requires 5 instruction cycles per component addition on the Pentium processor.

n	56001/20	56001/60	P60
3	1.6	0.5	2.0
8	3.1	1.0	2.6
32	10.3	3.4	7.2
128	39.1	13.0	25.7
512	154	51.3	99.1
2048	615	205	445
8192	2458	819	1777

Table 2: Execution times for vector addition

In contrast to the preceding example the implementation for the DSP utilizes both memory spaces. The highly parallel DSP architecture permits a component addition in only 3 instruction cycles. The DSP code requires some additional overhead during initialization (the first operands must be fetched to the corresponding registers before entering the loop) but the loop itself runs faster. The DSP needs exactly $14+6n$ clock cycles for the entire vector addition which is actually faster than the string comparison.

Table 2 shows the obtained execution times in microseconds. For short vectors, even the slow DSP executes faster than the three times higher clocked Pentium. With longer vectors the Pentium gains an advantage due to its cache mechanisms, but never reaches the results of the 60 MHz DSP version.

3.4 Vector-Matrix Multiplication

We extended the investigation of elementary algorithms to the multiplication of a vector with a matrix. Here the special case of multiplying a 3-element vector with a 3x3 matrix was analyzed because of its frequent usage in 3D computer graphics.

The computation of a component of the resulting vector needs three multiplies and two additions. The DSP implementation takes full advantage of the powerful MAC instruction, which performs a multiplication, an addition, (up to) two memory references and two address register updates in two clock cycles. The completion of the whole operation requires $6+20n$ clock cycles on the DSP.

The Pentium version of the code needs much more transfer operations. Because of the dependencies between the operands the processor cannot take advantage of its superscalar architecture, its second integer pipeline is useless.

n	56001/20	56001/60	P60
1	2.3	0.8	4.1
8	8.3	2.8	23.4
64	64.3	21.4	177
512	512	171	1405
4096	4096	1365	11230

Table 3: Execution times for vector-matrix multiplication

Table 3 shows the obtained execution times in microseconds. The parameter n denotes the number of multiplications performed. The DSP is clearly more suitable for this kind of algorithms. Even the slow chip version is more than twice as fast as the Pentium processor.

3.5 FIR Filter

The next problem investigated is the well-known digital FIR filter from the class of algorithms, DSPs were originally developed for.

The DSP version benefits from the hardware loop capability, the MAC-instruction, the Harvard structure and the circular buffer addressing mode of the processor. One filter tap is calculated in one assembly instruction. Therefore, the DSP needs $22 + 2n$ clock cycles for an n-Tap FIR filter.

The main drawback of the Pentium realization is the need for sequential processing. Furthermore, the chip lacks the above mentioned special hardware, so the execution of the filter is expected to be much less effective. The obtained processing times for the computation of one output value are given in microseconds in table 4, n denotes the number of taps.

n	56001/20	56001/60	P60
16	2.7	0.9	7.1
32	4.3	1.4	14.7
64	7.5	2.5	28.7
128	13.9	4.6	51.7
256	26.7	8.9	97.9
512	52.3	17.4	190.6
1024	103.5	34.5	386.3

Table 4: Execution times for FIR filter

Apart from Fast Fourier Transforms, this example demonstrates the maximum advantage the DSP can gain over a general-purpose microprocessor like the Intel Pentium.

3.6 Summary

As demonstrated, the answer whether or not a particular algorithm is likely to be sped up by a DSP, fully depends on the implementation. The better the program fulfills the following criteria, the larger the possibility for acceleration by a DSP.

- The algorithm allows to perform an arithmetic operation and one or two memory references simultaneously.
- The algorithm consists mainly of additions, multiplications and convolutions.
- The algorithm contains few or no conditional branches. Instead, it uses loops with known numbers of iterations.
- The number format of the DSP can be used.
- Data to be processed adheres to a constant format and fixed length.
- Only a constant fraction of the processed data is needed at one distinct processing step.
- The algorithm requires no divisions.

In addition, these results encourage to build dedicated DSPs for specific applications.

4 A more complex Experiment

In the second part of our work we tried to migrate real-world multimedia functionality from the general-purpose CPU of a PC to the DSP. We chose a JPEG still image decoder as an example of a more complex algorithm.

4.1 Integration into VSTa

In VSTa all work outside the microkernel is done by servers, so we decided to implement a simple server for performing the JPEG decode operation.

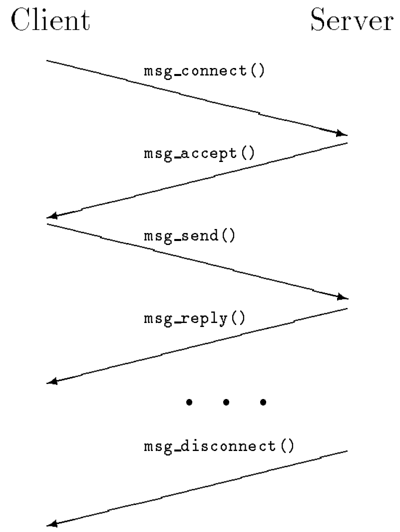

Figure 1: Inter-process communication in VSTa

Communication between a client and a server always happens in a connection-oriented manner as depicted in figure 1. First, a client tries to establish a connection to the server with the call `msg_connect()`, which is either accepted or refused by the server. After the acceptance of a connection the client is authorized to transmit data to and from the server via the call `msg_send()`. Every `msg_send()` is answered by the server with a call to `msg_reply()`. The connection can be aborted by the client or the server using `msg_disconnect()`. In our example the client sends a JPEG-encoded file to the server that decodes the data and sends it back on request.

A JPEG decoder roughly consists of the three modules shown in figure 2. For complexity reasons we did not try to implement the full decode functionality on the DSP. Instead, we concentrated on fundamental parts of the JPEG decode process. The most expensive operation is the Inverse Discrete Cosine Transform. To limit the complexity of the DSP software, we implemented the IDCT (sub-optimally) as two successive matrix multiplications. The dotted rectangle in figure 2 contains the parts of the implementation which were moved to the DSP.

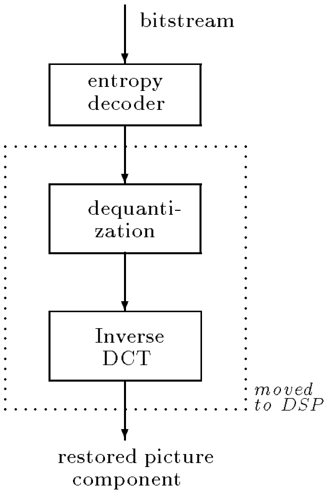

Figure 2: Modules of a simple JPEG decoder

The decoder version utilizing the DSP transmits and receives data in the following way: A client opens a JPEG file to be decoded and sends it to the server, which performs entropy decoding. The obtained coefficient matrices are sent to the DSP. There dequantization and Inverse DCT take place. The resulting pixel matrices are sent back row by row to the server and finally transmitted to the client.

Data is transferred to and from the DSP plug-in card with a polling mechanism. More complex, interrupt-driven solutions were designed but not implemented due to the lack of debugging support under VSTa.

4.2 Results

We measured the time needed to decode JPEG pictures at various resolutions and subtracted from the obtained value the time needed to process an empty file to eliminate the overhead introduced by file access.

Table 5 contains the obtained decoding times for both variants of the decoder in seconds; the DSP runs at 20 MHz.

resolution	pixels	P60	P60+DSP
640x480	307200	0.2	10.1
768x512	393216	1.4	14.9
800x600	480000	0.8	17.4
1024x768	786432	2.9	29.4
1200x1024	1228800	4.7	45.7

Table 5: Execution times for JPEG decoder

At a first glance, the results are extremely disappointing. The version utilizing the DSP needs approximately 10 times the execution time to accomplish the same task as the pure Pentium solution.

Therefore we investigated performance bottlenecks in the DSP version. The decoding functionality was removed from the DSP program. Instead, the DSP sent the received data unchanged back to the PC. In this way we were able to separate the 'raw' DSP decoding time from communication overhead. Table 6 shows the obtained results, t_{comm} represents the time needed for processor communication and t_{dec} is the time for decoding of the picture, both values are given in seconds. The Pentium processing times are given for comparison.

resolution	pixels	t_{comm}	t_{dec}	P60
640x480	307200	6.8	3.3	0.2
768x512	393216	9.0	5.9	1.4
800x600	480000	11.0	6.4	0.8
1024x768	786432	17.8	11.6	2.9
1200x1024	1228800	27.8 i	17.9	4.7

Table 6: Communication vs. Decode time for DSP-based decoder version (DSP clocked at 20 MHz)

Obviously, communication between the Pentium and the DSP is the most inefficient part of the implementation. Up to 67 percent of the whole DSP processing time is wasted during data transmission.

4.3 Discussion

The JPEG decoder using the DSP did not achieve competitive results. The reasons for the bad performance are as follows:

Communication between the two processors is highly ineffective. The host Interface of the Motorola DSPs is originally intended for the exchange of debugging and control information during software development, not for fast transmission of data to be processed. Also, the port addressing of the PC is slow for high-speed data transmission. Furthermore, the programmed mechanism for sending data to the Signal Processor wastes DSP time. It is believed that a high-speed link interface in conjunction with a reasonable PCI plug-in board or dual-ported memory, mapped into the PC's address space, could yield far better transmission rates.

The implemented algorithm needs bidirectional data transmission. Obviously, functions requiring only unidirectional transmission cut the communication time to the half. In these functions the DSP acts as front-end for input data or as output device. To propagate this type of application the integration of D/A and A/D converters into the DSP subsystem is essential.

The pure processing performance of the DSP is acceptable (remember, we can reduce the processing time to a quarter if we utilize the state-of-the-art version of the DSP, the 56002 clocked at 80 MHz).

Furthermore, actual state-of-the-art microkernels achieve at least 10 times better communication performance than current operating systems implementations [7]. An integration of the DSP hardware into such an environment is expected to render substantially improved performance.

5 Conclusions

In this paper we analyzed, how Digital Signal Processors can be exploited to relieve a system's main CPU from computationally intensive and time-critical algorithms. To this end, a DSP board has been integrated in a modern microkernel-based system and a real-world application has been tested in the new environment.

At a first glance, the obtained execution times were disappointing. However, close analysis revealed that hardware- and software-induced communication overhead is a major cause for the performance penalty.

The availability of more efficiently integrated DSP boards and especially recent advances in operating systems technology for general-purpose microprocessors indicate that this overhead can be reduced drastically. Hence, in the second stage of our project a state-of-the-art DSP board is currently integrated into L4, the fastest second generation microkernel available [7]. At a later stage a POSIX-compliant Linux implementation[1] on top of the L4 microkernel serves as comfortable DSP development platform and simultaneously exploits all the accelerator functionality.

References

[1] Spectron Microsystems, Inc. *IA-SPOX : Native Signal Processing Software Development Kit.* Santa Barbara, CA. 1995

[2] Dave Andrews. *Multimedia Goes Native.* Byte 3(1995), p.38

[3] Laurence C.Stewart, Andrew C. Payne, Thomas M. Levergood. *Are DSP Chips obsolete?* Cambridge Research Laboratory, Technical Paper, 1992

[4] Terje Mathiesen. *Pentium Secrets* Byte 7(1994), pp. 191-192

[5] Motorola, Inc. *DSP56000/56001 Digital Signal Processor User's Manual.* Phoenix, AZ. 1990

[6] Andrew Valencia. *An Overview of the VSTa Microkernel.* Technical Report, available from ftp.cygnus.com, 1995

[7] Jochen Liedke. *On μ-Kernel Construction.* Proceedings of the 15. ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain Resort, 1995, pp. 237-250

[8] Gregory K. Wallace. *The JPEG Still Picture Compression Standard.* Communications of the ACM 4(1991), pp. 30-44

[9] V.Živojnović, H.Schraut, M.Willems, R. Schoenen. *DSPs, GPPs, and Multimedia Applications - An Evaluation Using DSPstone.* Proceedings of ICSPAT'95, Boston, 1995

[1] Information about this project can be obtained from http://www.inf.tu-dresden.de/TU/Informatik/IBDR/lsbs/L4/LinuxOnL4/linux_port.html