DSPs as flexible Multimedia Accelerators

Robert Baumgartl, Hermann Härtig
Department of Computer Science
Dresden University of Technology
{robert.baumgartl, hermann.haertig}@inf.tu-dresden.de

ABSTRACT

The increase of multimedia data processing requires immense processing power and transfer bandwidth as well as the consideration of real-time requirements. To reduce the CPU load the integration of flexible coprocessors seems to be a promising approach. This paper focuses on the integration of Digital Signal Processors (DSPs) as flexible multimedia accelerators into standard PC architectures running microkernel-based systems. Three components proved essential for multimedia acceleration: First, a data transfer mechanism capable of sustaining at least 30 Mbytes/s transfer rate, second, a DSP kernel with static scheduling and a very efficient context switch and third, a microkernel server running at the host which is responsible for data transfer between DSP and CPU and for calculating DSP schedules.

1. INTRODUCTION

The increasing integration of multimedia data into applications requires immense processing power and transfer bandwidth as well as the consideration of real-time requirements. This situation results in an unacceptable slowdown of applications and high latency of the system. The usual approach of concentrating more and more computing power into the main processor does not provide a durable solution, because at the same time more and more complex multimedia algorithms (data compression, 3D graphics, music synthesis) are being deployed. One approach to this dilemma is to incorporate specialized accelerators into the computer system and remove computing-intensive code and interrupt load from the CPU.

The following features render DSPs as ideal for this task [5]:

1. DSPs allow a high degree of parallelity and incorporate special features into the instruction set and into arithmetic and addressing units resulting in an extremely powerful processor core. Most multimedia algorithms require exactly this vast amount of computing power.
2. DSPs are designed to handle large amounts of streamed data, which are common in multimedia applications.
3. DSPs are comparatively easy to operate under real-time conditions. It is much easier to predict execution times for a given piece of DSP code than for general purpose processor (GPP) code. Most multimedia tasks (e.g. video or audio codecs) run under real-time conditions.
4. The price-performance ratio of DSPs is much better than that of GPPs.
5. DSPs are designed to operate under heavy interrupt load which is very common in audio processing.
6. In contrast to special chipsets, DSPs are freely programmable. This eases migration to newly evolving algorithms and applications.

Therefore, the aim of our work is to demonstrate DSPs as multimedia accelerators and establish them as component within a microkernel-based real-time system running on standard PC hardware. This component comprises a DSP, a bus interface to the host, local memory and interfaces to the outside world (e.g., audio codec, telephony interface). It obsoletes external hardware components of the host (e.g., MPEG decoder, sound card).

A part of the project focuses on the analysis and evaluation of different bus systems (e.g., ISA, SCSI, PCI) and transfer mechanisms, to select the most efficient hardware interface between CPU and accelerator [1]. Other parts deal with benchmarking DSPs, the development of application examples [7] or the evaluation of applications with respect to their possible acceleration by DSP architectures.

We use standard PC hardware as basis for our experiments. The test system is a Pentium 166 MMX equipped with 64 MBytes of RAM and 512 kBytes 2nd level cache. The DSP subsystem consists of a PCI plug-in board equipped with a TMS320C44 DSP

module, clocked at 50 MHz and 128 kWords of static memory each for the local and the global address space. The board features an AMCC S5933 PCI controller and is able to perform PCI Busmaster DMA.

The remainder of the paper is organized as follows: Section 2 gives a brief overview of our prototype system. The following two sections go into some detail focusing at some aspects of the DSP kernel (section 3) and discussing relevant concepts of the server running at the host in section 4. Section 5 shows the current state of the project, presents some performance numbers and gives a short outlook. Finally, the paper is summarized and the main lessons learned to date are pointed out in section 6.

2. SYSTEM OVERVIEW

The aim of the project is to construct a working prototype to demonstrate the acceleration capabilities of DSPs in standard PC environments using microkernels. This section gives a brief overview of the realized system.

The basic scheme can be described best by the example in figure 1. The box on the left shows the software environment of the host. At the lowest level runs a very efficient microkernel which offers only the most basic operating system functionality. On top of it, a number of servers provide more complex operating system functionality. Servers are responsible for controlling hardware components, for instance the file system, the network interface and the multimedia accelerator, that is the DSP subsystem.

The latter is depicted at the right side of figure 1. Within the DSP, a small coprocessor-optimized kernel is run, which is responsible for activating diverse tasks offering acceleration functionality for host processes. Application examples include multimedia data encoders and decoders, kryptography software or music synthesizers.

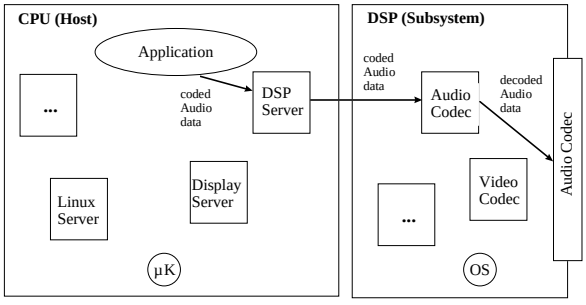

Figure 1: Example of Acceleration Mechanism

Now consider an application running at the host and requiring accelerator functionality, for instance decoding of an MPEG audio stream. This application will initiate a Quality-of-Service negotiation with the DSP server, which is informed about the actual workload of the DSP subsystem. Based on this knowledge, the server either rejects or grants DSP access to the application. It analyzes the stream to be processed and estimates the necessary DSP resources (CPU time, memory, access to periphery). If necessary, it downloads application code to the DSP. Then, a new DSP schedule is generated and transmitted to the DSP. Finally, after the new DSP task has been integrated into the DSP task activation mechanism, the DSP server manages data transfer from the application process to the DSP (and vice versa, if necessary).

The application process is allowed to interrupt the connection at any time, resulting in the server determining a new DSP schedule (excluding the now unnecessary task from further processing) and transmitting it to the DSP. The task's code remains in DSP memory until the memory is required by another task.

3. THE DSP KERNEL

The immense processing power of actual DSPs [3] led to the idea of distributing acceleration functionality across more than one host process at the same time, which requires a kind of operating system for the DSP. To make as much processing power available as possible, this kernel has to be implemented very efficiently. The main design decisions were:

1. We limit DSP tasks to periodic processing of streams with constant deadlines, which is typical for most current multimedia applications [2]. Therefore, tasks to be run at the DSP can be described by execution length and activation frequency.
2. The computation of schedules is complex and not very well suited to DSP architectures. Hence, this computation is moved into the host server. The DSP itself does only static scheduling. A desired side effect is a very short context switch time of the DSP kernel.
3. To lower the kernel's requirements of the sparse resource (static) DSP memory, it realizes only mechanisms fundamentally necessary for coprocessing. For instance, dynamic memory management and inter-process communication are excluded from the DSP kernel.
4. It is absolutely mandantory to preserve the communication performance of the underlying

hardware. In [1] we described, how an efficient implementation of PCI busmaster DMA affects the transfer rate. By a careful optimization process we were able to raise the achievable transfer rate from approximately 3 Mbytes/s to 33 Mbytes/s. We chose PCI Busmaster DMA as appropriate transfer mechanism for data to be processed, whereas signalling between host and DSP kernel is done via conventional mailboxes.

5. Due to the constant deadlines of the DSP tasks a timer-activated context switch is useful. The kernel manages an according table of timing values for tasks. To prevent a task violating its deadline of corrupting the whole system, preemptive scheduling must be used.
6. For overhead and efficiency reasons, we consider high level languages (HLL) as „C“ inappropriate for implementation of the kernel. Therefore, the whole kernel software as well as our application examples were implemented in assembler language.

Hence, we identified the minimum necessary kernel functionality as follows:

- installation, start and deletion of tasks,
- context switch,
- communication with host and necessary synchronization,
- management of buffer memory for communication,
- management of peripheral devices (e.g., audio codecs or modem hardware).

Note that the installation of a task is distinct from its start. Installation includes download of the code, initialization of the necessary kernel structures and transmission of the timer value. The task is started immediately after transmission of the new schedule calculated by the host server.

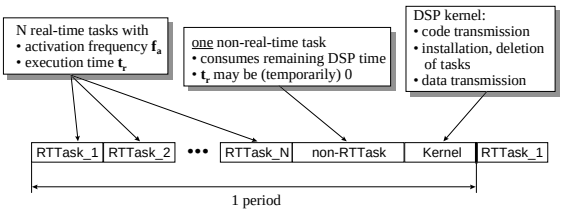

Figure 2: DSP Scheduling

Figure 2 summarizes some of the architectural aspects of the kernel. One period of scheduling consists of a number of N real-time tasks, being activated one after another. Then, exactly one non-real-time task is called, provided there is still DSP execution time available within the period. This task consumes the within the period remaining DSP processing time. A constant number of clock cycles at the end of every period is reserved for the kernel, which manages data transfer between host and DSP.

To summarize, our kernel has the following features:

- static, preemptive scheduling,
- dynamic load of applications at system's runtime,
- very exactly predictable timing of system services,
- implemented in optimized assembler language.

4. MICROKERNEL SERVER

On the host side, we rely on microkernel technology for flexibility, security and performance reasons. The basic idea is that a very lean (and therefore very fast) *microkernel* provides only a minimal set of operating system abstractions. More complex functionality is built using these abstractions in *servers* running in user space. Assuming a highly optimized implementation, a port of a classical monolithic operating system to a microkernel is almost as efficient as its pure monolithic counterpart [4]. For further information on microkernels we refer to [6].

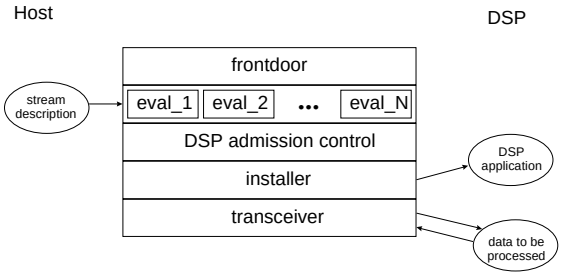

Figure 3: Host Server Architecture

Figure 3 shows the architecture of the DSP server running at the host. In principal it consists of 5 components. An application wanting to access accelerator functionality opens communication with the server via the `frontdoor` component. The `frontdoor` realizes a name server with whom applications may find out, which acceleration services the DSP subsystem offers (that means, for which algorithms DSP code exists).

If the requested functionality is available, one of N `evaluators` (one `evaluator` for every DSP application) is activated. Its purpose is to determine the needed DSP resources:

- by reading an appropriate stream description of the file to be processed by the DSP, or

- by directly analyzing the header of the file and extracting relevant parametes (e.g., for an MPEG audio encoder, this may be layer number, target bitrate and aural mode).

The `evaluator` estimates the prospected DSP processing time and memory requirements and transmits them to the `admission control` component.

The `admission control` keeps track of the current DSP workload and tries to determine a new schedule for the DSP based on the new task to be integrated. If this is not possible (for instance due to exceeding CPU requirements), the acceleration service is denied. Otherwise, the new DSP schedule is transferred to the `installer` which transmits it to the DSP.

Additionally, the `installer` checks, whether or not the according DSP code is already present in DSP memory and initiates the download, if necessary.

The DSP now integrates the new task into its scheduling cycle. On the host side, the `transceiver` takes control and organizes data transfer between application process and DSP task. Transfers can be uni- or bidirectionally.

5. RESULTS

In this section we want to discuss some performance aspects of our protoype. First we will demonstrate data concerning hardware data transfer mechanisms, befor we present first performance numbers of the DSP kernel.

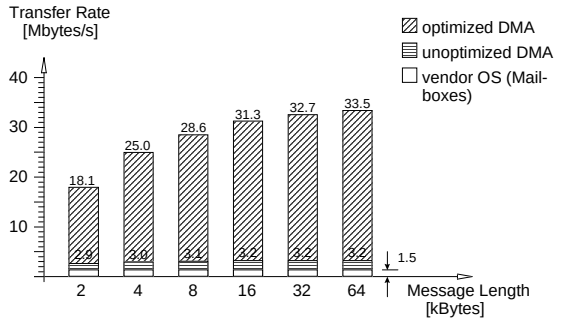

Figure 4: Communication Performance

Figure 4 illustrates the achieved transfer rates for different message sizes of our hardware with different mechanisms and implementations.

The original routines of the board´s vendor are based on the traditional mailbox mechanism, that is a word-by-word transfer using polling by both processors. Regardless of the message size this mechanism delivers a transfer rate of 1.5 Mbytes/s only which is unacceptable for multimedia applications. Additionally it wastes CPU and DSP resources by actively waiting for communication.

Surprisingly, switching to PCI busmaster DMA (by using additionally supplied vendor routines) resulted in approximately doubling the transfer rate to 3.2 Mbytes/s only which is far away from the 132 Mbytes/s theoretically possible with 32 bit wide PCI.

We then applied a careful analysis to the communication libraries and implemented our own hand-optimized version. By exploiting the main architectural features of the DSP we lowered the length of the innermost transmission loop from 15 to 1 instruction cycles. As a result, we achieved almost ten times the transfer rate as with the unoptimized routines with a peak performance of 33.5 Mbytes/s for messages of 64kBytes length. This result emphasizes the importance of a very efficient implementation of the basic data transfer mechanism. Discussions with the PCI controller vendor and with other researchers hinted that this is the maximum transfer rate achievable with this PCI controller.

Due to the still ongoing development process we are able to present preliminary kernel performance numbers only. For more up-to-date results we refer to our web pages (cf. section 6).

Firstly, the context switch needs 110 instruction cycles. Because the TMS320C44 DSP comprises 40 registers necessary to preserve per task, 80 cycles are necessary to save and restore them. The remaining 30 cycles implement the switch mechanism, demonstrating a very low overhead. Based on a clock frequency of 50 MHz, the context switch needs 4.4 microseconds.

Second, the interrupt latency (the time between raising the interrupt signal on the processor pin and the execution of the first instruction of the interrupt service routine) is bound by the processor's hardware only. In contrast to other DSP operating systems, we do not need to switch off the processor's interrupts. Therefore, the latency is as low as 3 instruction cycles.

Third, the current kernel implementation needs approximately 4 kwords of DSP memory, which we consider acceptable for the offered functionality.

6. CONCLUSION & OUTLOOK

As we have demonstrated, it makes sense to integrate DSPs as flexible accelerators into standard PC architectures, if the following conditions are satisfied:

- hardware communication mechanism capable of delivering at least 30 Mbytes/s (e.g., PCI or FireWire),
- efficient DSP kernel with minimal coprocessor abstractions,
- efficient control component running at the host,
- wide software base for the DSP.

As a next stage of the project we want to explore methods and tools for the automated generation of DSP kernel and microkernel server for heterogeneous DSP and microkernel architectures. Second, the possible adaptation of the kernel to a DSP multiprocessor system will be investigated.

To find out more about our project, we refer to our webpages which can be found at:

`http://os.inf.tu-dresden.de/~baumgrtl/work.html`

ACKNOWLEDGEMENTS

The authors want to thank Jean Wolter, Michael Hohmuth, Martin Borriss, Sebastian Schönberg and Lars Reuther for discussion, proofreading and helpful comments.

REFERENCES

[1] Baumgartl, Robert; Härtig, Hermann: *Efficient Communication Mechanisms for DSP-based Multimedia Accelerators*, Proceedings of the 8th International Conference on Signal Processing Applications & Technology (ICSPAT´97), San Diego, 1997

[2] Baumgartl, Robert; Stuhlemmer, Kai; Härtig, Hermann: *Data Dependency in MPEG Audio Frame Decoding Time on a DSP Accelerator*, accepted for publication at the 9th International Conference on Signal Processing Applications & Technology (ICSPAT'98), Toronto, September 1998

[3] Bier, Jeff et al: *Buyer's Guide to DSP Processors*, Berkeley Design Technology Inc., Berkeley, 1997

[4] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, J. Wolter: *The Performance of µ-Kernel-based Systems*, Proceedings of the 6th Symposium on Operating System Principals (SOSP); St. Malo, 1997

[5] Lapsley, Phil: *DSP Chips enable PC Multimedia*, Microprocessor Report, 8(1994)

[6] Liedtke, Jochen: *On Micro-Kernel Construction*, Proceedings of the 5th Symposium on Operating System Principles (SOSP), Copper Mountain Resort, 1996

[7] Stuhlemmer, Kai: *Echtzeitfähiger MPEG-Audio-Encoder für DSP*, Master´s Thesis, Dresden University of Technology, 1998 (in german)