Performance Monitoring and Tuning for a Single-Chip Multiprocessor Digital Signal Processor

Jihong Kim

Texas Instruments
P.O. BOX 655303, M/S 8374
Dallas, TX 75265, USA

Yongmin Kim

Dept. of Electrical Engineering, Box 352500
University of Washington
Seattle, WA 98195, USA

Abstract

A new generation of high performance programmable digital signal processors (DSPs) has a highly-integrated parallel architecture, incorporating special-purpose hardware features, on-chip memory and multiple processors into a single chip. For such single-chip multiprocessor DSPs, however, a sophisticated performance monitoring tool is essential to achieve the maximum performance. In this paper, we discuss the requirements and functionality of performance monitoring tools suitable for single-chip multiprocessor DSPs. As a specific example, we describe a performance monitoring tool developed for Texas Instruments' TMS320C80 (MVP), MVP Performance Monitor (MPM), which satisfies these requirements and functionality. The effectiveness of the MPM is demonstrated using an 8×8 block-based discrete cosine transform (DCT) implementation. An overall speed-up of 4.67 was achieved by using the MPM.

1 Introduction

Support for performance monitoring and tuning in complex systems such as parallel systems has long been an active research area. In recent years, the emphasis has been on evaluating the performance of *large-scale* parallel programs. For example, many performance tools have been developed for tuning parallel programs in shared-memory multiprocessors or distributed systems [1, 2, 3]. In this article, we discuss performance monitoring and tuning for a much smaller parallel architecture, *single-chip* multiprocessor digital signal processors (DSPs).

In order to meet the heavy computing requirements of emerging multimedia applications dealing with real-world data types such as video and voice, a new generation of high performance programmable DSPs has a highly-integrated parallel architecture, incorporating special-purpose hardware features, on-chip memory and multiple processors into a single chip. A generic architectural model of these DSPs is shown in Fig. 1. The DSP has large on-chip memory, a separate I/O controller and one or more processing elements (PEs). These functional blocks are all connected through some form of an interconnection network such as shared buses or crossbar switch network. Because of the large data requirement in multimedia applications, data movement by the I/O controller is performed in parallel with data processing by PEs, thus improving the overall performance. Most of the high performance DSPs such as Analog Devices' ADSP-21060 Super Harvard Architecture (SHARC) DSP [4], Motorola DSP96002 Processor [5] and Texas Instruments TMS320C80 (MVP) [6] belong to this architectural family.

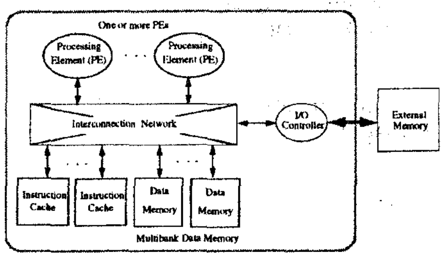

Figure 1: Single-chip multiprocessor DSP architectural model.

In order to develop an efficient DSP program for a single-chip multiprocessor DSP, good understanding of not only the algorithms and DSP's intricacies, but also the system-wide program behavior is necessary. Without system-wide performance understanding, a DSP program can suffer from various performance bottlenecks such as resource conflicts and unbalanced synchronization. This extra overhead could significantly degrade the overall performance of a DSP program. Unfortunately, this bottleneck is often difficult to predict and identify even for experienced DSP programmers. Considering that the main reason of using DSPs is to obtain high performance, there is a strong need for performance monitoring tools for a single-chip multiprocessor DSP.

In Section 2, the DSP-specific requirements for performance monitoring tools are discussed and the important performance parameters for our target DSPs are identified. Then, as a specific example, we describe a performance monitoring tool developed for Texas Instruments' TMS320C80 (MVP), MVP Performance Monitor (MPM), which satisfies these requirements and functionality. An overview of the MVP and MPM is presented in Section 3. In Section 4, the effectiveness of the MPM is demonstrated using an 8×8 block-based 2-D discrete cosine transform (DCT) implementation.

2 Performance Monitoring Tools for Digital Signal Processors

2.1 Performance Instrumentation Approach

In general, there are four performance instrumentation levels: hardware, system software, run-time system software and application code [7]. However, target application codes are typically executed directly on top of the hardware in DSP-based systems without much system software or run-time system support to minimize the overhead. So, more information from the hardware and application levels is necessary to further improve the performance of DSP applications.

For DSP-based performance optimization, none of the hardware-based monitoring and software instrumentation-based monitoring approaches are adequate. In DSP applications, overall performance is often dominated by small code segments (e.g., part of a single procedure). For these code segments, software instrumentation points are not easily defined because there are not enough meaningful primitive-level activities within these small code segments. Furthermore, these segments are often written in assembly language, making it more difficult to pinpoint primitive-level activities for performance monitoring. Since the performance of small code segments is being monitored in DSP applications, the software instrumentation code added to DSP applications can significantly distort the run-time behavior of DSP applications, resulting in inaccurate performance data. On the other hand, a hardware-based monitor does not provide enough higher-level information necessary due to its limited monitoring scope.

In order to collect accurate hardware performance data as well as higher-level information without perturbing the system's behavior by monitoring process, we believe that a software monitor approach based on a hardware simulator model is more appropriate. If an accurate hardware simulator model is available, this approach supports both detailed hardware data collection and higher-level analysis information without introducing any significant artifact into measurements. Developing an accurate hardware simulator would generally require a significant amount of effort. In DSP-based systems, however, this is not an extra burden because an accurate DSP simulator model is typically available from the DSP manufacturer. For example, most low-level DSP programs are typically developed using a simulator for a particular DSP. Thus, a performance monitoring tool for a specific DSP can be developed by extending the existing DSP simulator.

2.2 Performance Monitoring Tools Requirements

In this section, we collect the desirable features of DSP-based performance monitoring tools. Ideal performance monitoring tools to be used in developing DSP applications should have (at least) the following properties:

- Performance monitoring integrated into the debugging tools [8].
- Familiar and uniform user interface.
- Meaningful analysis results at a reasonable cost.
- User extensibility.

As discussed in [8], we believe that it is important to handle debugging and performance monitoring in a unified way; there is a continuum between debugging for correct functionality and debugging to achieve the desired performance objectives. An ideal performance monitoring tool should be seamlessly integrated into debugging tools, especially in DSP-based systems where the debugging tools play a central role in developing application programs. An ideal performance monitoring tool should also present a uniform and familiar user interface throughout all its components, so as to relieve the users from learning many different interfaces. For example, the user should be able to use the same commands to control the program's flow in both the function debugging mode and performance debugging mode.

Useful performance monitoring results should be obtained at a reasonable cost. In DSP applications, a multiple number of small code segments such as tight loops are often the candidates of in-depth performance monitoring and analysis. If performance monitoring takes an excessive amount of time to produce measurements and analysis results, its usefulness is significantly reduced. A performance monitoring tool should be extensible by the user. For example, in a single-chip multiprocessor, it is impossible to expect all the possible combinations of events in advance. There should be a provision that allows the user to select the types of events to be monitored.

2.3 Performance Monitoring Parameters

For our target DSPs whose model was described in Section 1, there are three main performance factors: (1) balance between I/O time (by I/O controller) and compute time (by processing elements (PEs)), (2) instruction cache behavior, and (3) interconnection network contentions among PEs and I/O controller.

Since our target DSPs can perform data processing and data movement concurrently, analyzing data processing and data movement requirements for a given function is very important. In order to arrive at an optimal program implementation, it is necessary to know whether a specific implementation of an individual subtask (of a program) is I/O-bound or compute-bound. Additionally, it is also necessary to know the degree of I/O-boundness or compute-boundness of each subtask. For example, if one subtask is found to be I/O-bound, its degree of I/O-boundness can guide us in implementing other subtasks of a program, trying to reach a balance between I/O time and compute time for the overall program.

The second important performance parameter is on-chip instruction cache behavior. Since one I/O controller serves both instruction cache misses and data movement requests from PEs in target DSPs, instruction cache misses affect not only the compute time of a specific function but also its I/O time. Thus, cache misses directly affect the overall program's execution time. Because DSP programs are typically dominated by small code segments and the cache-miss service time varies depending on the I/O controller's workload, a simple count of the total number of cache misses would not provide enough information to understand and improve the program's cache behavior. More detailed information such as source address, frequency and service time for each cache miss is necessary.

The third performance factor is interconnection network contentions among PEs and I/O controller. Contentions among PEs would increase the total compute time while contentions between PEs and I/O controller would increase either the compute time or I/O time depending

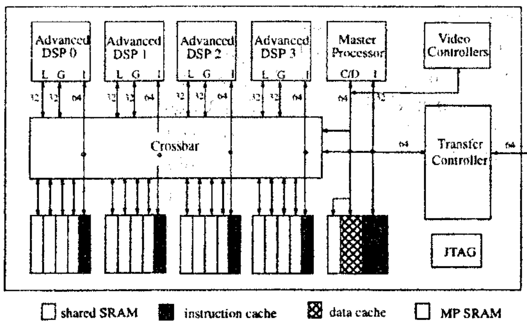

Figure 2: High-level block diagram of the TMS320C80.

on the interconnection network priorities of PEs' and I/O controller's accesses. For interconnection network contentions, the total number of contentions for each PE and I/O controller provides enough information to improve DSP programs because the interconnection network priorities for PEs and I/O controller tend to remain relatively constant and they can be modified based on the total number of contentions.

3 MVP Performance Monitor (MPM)

In this section, we describe the MVP Performance Monitor (MPM) as an example DSP-based performance monitoring tool. The MPM is based on a cycle-accurate MVP simulator. It satisfies the requirements of the DSP-based monitoring tools and supports three important performance parameters discussed in Section 2. Before the MPM is described, we briefly describe the TMS320C80 (MVP) processor first. (For the detailed description, see a reference [6].)

3.1 Overview of TMS320C80 (MVP) Processor

The TMS320C80 can be described as a single-chip, heterogeneous, MIMD multiprocessor connected via a crossbar to multiple on-chip shared memory modules. It combines a RISC processor and four advanced DSPs as well as an intelligent direct memory access (DMA) controller and two video controllers into a single-chip device. It is capable of processing more than 2 billion operations per second (BOPS) with its 2.4 Gbytes/sec on-chip data transfer rate. In order to reduce the data transfer overhead with the external memory/devices, a large on-chip memory (25 2-kbyte modules) is provided as well.

Fig. 2 shows a high-level block diagram of the major functional blocks of the TMS320C80. The Master Processor (MP) is a general-purpose RISC processor with an integral IEEE 754 compatible floating-point unit. In a typical operation mode, the MP serves as the main supervisor and distributor of tasks within the TMS320C80. The four Parallel Processors (PPs) or advanced DSPs (ADSP 0-3) have a highly parallel architecture optimized for multimedia, video/image compression, image/signal processing and computer graphics. Each ADSP is capable of performing up to 15 RISC-equivalent operations in a single clock cycle via a long instruction word (64 bits) mechanism and has many powerful features not found in conventional DSPs. For example, each ADSP has three-operand 32-bit arithmetic and logical unit (ALU) which can be optionally split into two 16-bit or four 8-bit units. The Video Controllers (VC) provide supports for programmable video timing to control both capture and display. The processors and on-chip shared-memory modules are fully interconnected through the high-performance crossbar switch network.

While five processors (the MP and four ADSPs) provide the computing power for the TMS320C80, the Transfer Controller (TC), a dedicated memory controller with sophisticated data transfer logic, manages all the data transfer requests and cache misses from these processors. The TC prioritizes different types of data transfer requests and transfers data within and between the on-chip and external memories. Because of the high data bandwidth required for multimedia applications and the overhead of accessing off-chip memory directly, five processors typically work with data brought into the on-chip shared memory by the TC. Since the processors and the TC can operate in parallel, most data movement by the TC is hidden from the processors in the optimized implementation; while a processor is working on the current block of data residing in the shared memory, the TC is servicing a request for the next block in parallel. The TC which works as a dedicated memory controller for the whole MVP chip supports highly sophisticated data transfer logic. The TC's main data transfer mechanism is a packet transfer (PT), a transfer of data blocks between two areas of the MVP memory. Packet transfers are initiated by the MP, ADSPs, VC or external devices as requested to the TC under the software or hardware control. Once a processor has submitted a transfer request, it can continue program execution without waiting for the completion of the transfer.

3.2 Architectural Overview of MPM

The MVP Performance Monitor (MPM) supports three types of performance parameters identified in Section 2.3 with custom monitoring, cache monitoring, and contention monitoring, respectively. The overall architecture of the MPM is shown in Fig. 3. The MPM was tightly integrated with the Texas Instruments MVP Debugger tool which is widely used in developing MVP programs. The user can interactively switch between the performance monitoring mode using the MPM and the function debugging mode using the MVP Debugger. The performance monitoring mode runs about three times slower than the function debugging mode which runs 700 to 1,000 instructions per second on a SUN SPARCstation 20. This speed has been found to be adequate for monitoring the performance of multiple code segments switching interactively between the two modes. The core of the MPM is the extended MVP simulator. The extended MVP simulator consists of the MVP C++ simulator which accurately models the MVP up to a half-cycle[1] resolution, customized MPM extensions to the MVP C++ simulator which include three types of monitoring support and TC debugging capability, and a communication and synchronization module (CSM) which is responsible for communicating with the MPM user interface. The MPM user interface spawns a child process which monitors the user-specified monitoring event.

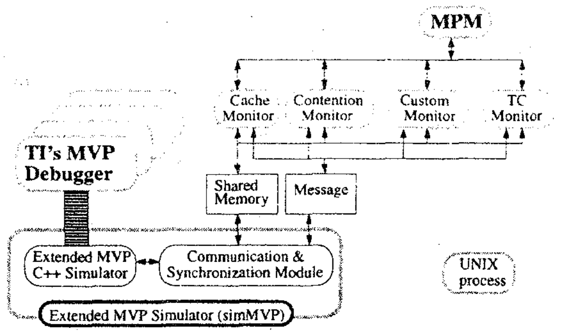

Figure 3: Overall software architecture of the MPM.

For cache monitoring, the collected information on cache misses for the code segment S is displayed, including the total number of cache misses, the total cache-miss service time, the total number of non-compulsory cache misses, and the total cache-miss service time for non-compulsory cache misses. It also displays the summary of all the cache misses in a table where the source and destination addresses for each cache miss, the average service time for each cache miss, and its frequency are listed. Based on this information, the user can restructure the MVP program or adjust the program size to reduce the number of non-compulsory cache misses. Contention monitoring displays the total number of crossbar switch contentions

[1]In case of the MVP running at 50 MHz, a full clock cycle is 20 ns, so a half cycle is 10 ns.

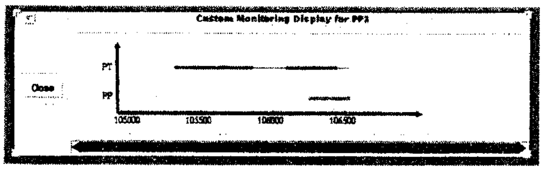

Figure 4: An example display for the custom monitoring result.

for each processor.

Custom monitoring is used to observe a user-defined event unlike cache and contention monitoring where monitoring events are predetermined by the MPM. In the current version of the MPM, a user-defined event is specified by ADSP checkpoints which are the addresses of selected ADSP instructions. During custom monitoring, the execution of ADSP checkpoints is recorded. The result from custom monitoring is displayed graphically as shown in Fig. 4. The x axis of this graph indicates the MVP clock cycle numbers. The lines in the upper row of Fig. 4 display the status of data movements (i.e., packet transfers) requested by a specific ADSP. The thick line indicates a time interval when the packet transfer service is delayed because the TC is busy servicing higher-priority requests while the thin line represents a time interval when the requested packet transfer is actually serviced by the TC. The line[2] in the lower row of Fig. 4 shows a time interval when the specified ADSP checkpoint is being executed. The user can measure each of these intervals by clicking the mouse button.

One of main uses of custom monitoring is to evaluate whether an implemented MVP program is compute-bound or I/O-bound. In the MVP, ADSPs (or MP) submit data transfer requests to its I/O controller (the TC) and check for the data transfer completion by polling a predetermined register. Therefore, by custom monitoring the ADSP polling instruction for the packet transfer completion, it can be determined if the requested data transfer has been completed.

4 Performance Tuning Example: 8×8 Block-Based 2-D Discrete Cosine Transform (DCT) Implementation

In this section, we describe our experience in using the MPM to improve the performance of the 8×8 block-based 2-D discrete cosine transform (DCT) implementation on the MediaStation 5000 (MS5000). The MS5000 is a TMS320C80-based multimedia system [9]. The program first divides an $N \times M$ 8-bit input image into many $(N/8 \times M/8)$ nonoverlapping 8×8 blocks. Then, 2-D DCT is performed on each 8×8 block by row-wise 8-point DCTs followed by column-wise 8-point DCTs. The output block has the same spatial resolution (8×8) as the input block. With the 8 bits/pixel input gray scale image, the range of 2-D DCT output coefficients is from -2048 to +2047 due to the repeated multiplications (with the cosine values) and accumulations. Thus, the DCT coefficients are stored as 16-bit fixed-point numbers. Out of 16 bits, the four least significant bits represent the fractional part while the upper twelve bits represent the integer part.

The 2-D DCT program consists of two tasks, 8-bit to 16-bit conversion (convert task) and 16-bit 8×8 block-based 2-D DCT (dct task). The separate 8-bit to 16-bit convert task is necessary to prepare the input data properly for the 16-bit 2-D dct task.[3] Among various fast DCT algorithms, Lee's algorithm was used in the dct task [10]. Two processing cores for the convert and dct tasks were highly optimized, heavily utilizing many advanced features of the ADSP. The convert processing core takes 1.25 cycles per output pixel while the dct processing core takes 352 cycles per 8×8 block, or 5.5 cycles per output pixel. Four ADSPs running in parallel at 50 MHz, with each ADSP processing a quarter of the image, would take a total of 8.84 ms to perform the 8×8 2-D DCT for a 512×512 input image. 1.63 ms would be taken for the convert task while the dct task would take 7.21 ms. However, these estimates are pure processing times and do not include any overhead at all.

Using the two routines as building blocks, the first version of the integrated program was implemented by combining the two processing cores within a single ADSP-level function as shown in Fig. 5. One 8×8 8-bit input block is brought into the on-chip memory at a time and one 8×8 16-bit DCT coefficient block is written out to the external memory. The convert processing core and dct processing core share the same data flow in this implementation. The performance of this version was surprisingly bad. It took 44.5 ms, about 35.7 ms slower

[2]If an ADSP instruction at the checkpoint is executed only once, a point (instead of a line) will be displayed in the graph.

[3]If we use 8-bit input pixels in the 16-bit 2-D dct task producing the 16-bit fixed-point DCT coefficients, an extra shifting is required after every multiplication between a 16-bit cosine value and an 8-bit input pixel to change the multiplication result back to a 16-bit number. Additionally, in order to perform two 16-bit arithmetic operations simultaneously by splitting the ALU, two 16-bit multiplication results need to be packed into a 32-bit word. With the preconverted 16-bit input pixels, however, the extra shifting and packing operations become unnecessary by utilizing the hardware swapper of the ADSP's multiplier unit, which cannot be used with 8-bit pixels.

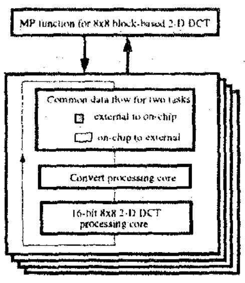
MP function for 8x8 block-based 2-D DCT
Common data flow for two tasks
external to on-chip
on-chip to external
Convert processing core
16-bit 8x8 2-D DCT processing core

Figure 5: Program structure of the first-version 2-D DCT program.

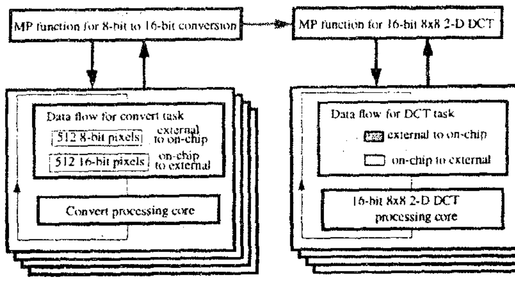

Figure 6: Program structure of the second-version 2-D DCT program.

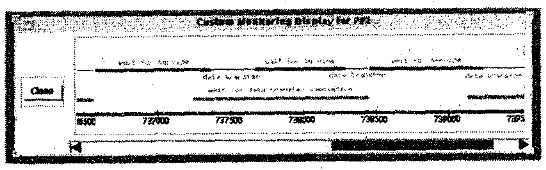

Figure 7: Custom monitoring result in the ADSP2 for the 8-bit to 16-bit conversion ADSP-level function.

than the theoretical estimate of 8.84 ms.

Through cache monitoring in the two processing cores, we identified that there were about 1,100 to 1,200 extra cycles spent for servicing instruction cache misses in each iteration (each 8×8 block processing). We did not expect this large number of cache-miss service cycles because the combined number of ADSP assembly-language instructions used in the convert (20 instructions) and dct (55 instructions) processing cores is 75, which is much less than the maximum number (256) of instructions that can fit into the ADSP's 2-kbyte instruction cache.[4] The large number of cache misses happened between the successive subroutine calls to the convert and dct processing core routines. The MVP linker was putting these routines in the instruction cache without any 64-instruction block consideration, e.g., starting a routine in the beginning of a cache block. Furthermore, the MVP linker did not place these two routines closely when building an executable module. So each core routine required two 64-instruction cache blocks. Four 64-instruction blocks for two core routines and a couple of additional 64-instruction blocks for the ADSP-level C code (i.e., **for** statement block) caused cache misses for each subroutine call. The overhead due to these non-compulsory cache misses caused the performance loss of between 22.5 ms and 24.6 ms.

In order to reduce the number of cache misses between the convert and dct tasks, in the second version, the two processing cores were put into separate ADSP-level functions as shown in Fig. 6. In this version, the converted image was stored in the external memory of the MS5000 before it was discrete cosine-transformed by the second ADSP-level function. The convert task for each ADSP brings 512 8-bit pixels (or four rows of 128 pixels) into the on-chip memory at a time instead of a single 8×8 8-bit input block for more efficient data movements. For this organization, cache monitoring showed no extra cache miss in both ADSP-level functions. The execution time was reduced to 16.3 ms from 44.5 ms, an improvement of 28.2 ms. The improvement was about 4 to 6 ms larger than one that can be explained by the large number of cache misses. This is because unnecessary cache misses affect the transfer time as well as the processing time, and the significant reduction in cache misses improves the overall I/O performance as well.

However, both ADSP-level functions became I/O-bound. For example, Fig. 7 shows the custom monitoring result in the ADSP2 after the convert task. It clearly shows that the convert task is I/O-bound. In custom monitoring, we set an ADSP checkpoint to be the address of the ADSP polling instruction which checks for the packet transfer completion. The solid lines in the lower row in Fig. 7 indicate that the ADSP2 is executing this polling instruction for a large number of cycles before the requested packet transfer is completed. The long thick lines in the upper row of Fig. 7 show that the packet transfer service is delayed for many cycles due to the TC's unavailability. The thin lines indicate the time intervals when the requested packet transfer is actually serviced by the TC. Because of its simple computation steps in the processing core, the convert task spends about two thirds of its

[4]The 2-kbyte instruction cache is divided into four blocks, each of which can store 64 64-bit instructions. Therefore, four different code segments can be simultaneously stored in the instruction cache.

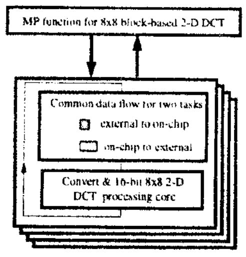
MP function for 8x8 block-based 2-D DCT
Common data flow for two tasks
external to on-chip
on-chip to external
Convert & 16-bit 8x8 2-D DCT processing core

Figure 8: Program structure of the third-version 2-D DCT program.

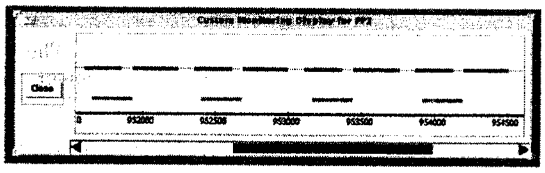

Figure 9: Custom monitoring result in the ADSP2 for the ADSP-level function with the two processing cores combined into a single one.

computing cycles waiting for the packet transfer completion according to Fig. 7 while the dct task spends more than 50% waiting for the packet transfer completion.

In order to reduce the overall I/O time, we combined the two processing cores into a single one as shown in Fig. 8 in the third version program. With the same data flow, the execution time was reduced to 11.6 ms due to the decreased I/O time. The new combined processing core did not have any non-compulsory cache miss. However, custom monitoring indicated that the implementation is still I/O-bound. Fig. 9 illustrates this I/O-boundness for the ADSP2 although a better balance has been achieved between the I/O time and processing time by combining two I/O-bound routines into a single routine. Comparing Fig. 9 to Fig. 7, the total length of the solid lines in the lower row is much shorter in Fig. 9, meaning that the ADSP2 spends much less time checking for the packet transfer completion. However, the ideal 2-D DCT implementation should not have any solid line in the lower row.

The fact that the third version was still I/O-bound forced us to make an in-depth analysis of the current data flow. We did not expect this version to be I/O-bound because the combined processing core was supposed to take only 6.75 cycles per output pixel while the number of pure I/O

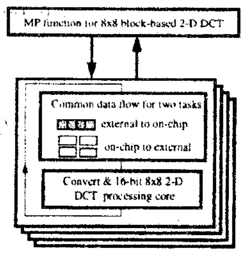
MP function for 8x8 block-based 2-D DCT
Common data flow for two tasks
external to on-chip
on-chip to external
Convert & 16-bit 8x8 2-D DCT processing core

Figure 10: Program structure of the fourth-version 2-D DCT program.

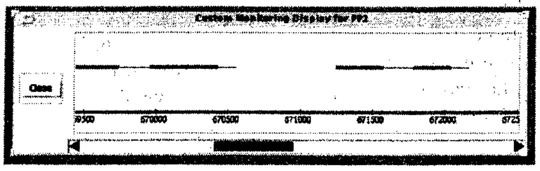

Figure 11: Custom monitoring result in the ADSP2 for the ADSP-level function with two processing cores combined into a single one and processing four 8×8 blocks at a time.

cycles was only 48 cycles/64 pixels,[5] or 0.75 cycles per output pixel. With four ADSPs submitting data transfer requests at the same time, the effective data transfer rate for the MVP is four times larger than 0.75 cycles per output pixel. On a close examination, we found out that the row-time access overhead was a big contributor to the large I/O time because accessing an 8×8 block requires frequent memory page boundary crossings.[6]

In order to reduce the row-time access overhead, we increased the number of 8×8 blocks being brought in and processed at a time to 4 from 1. The overall program structure for this version is shown in Fig. 10. This version became compute-bound as shown in Fig. 11. An ADSP checkpoint was executed only once well after the requested packet transfer was completed. This can be

[5]Using two cycles per memory access and 64-bit data width, it takes 16 cycles to read in 64 8-bit data (64 bytes) and takes another 32 cycles to write back 64 16-bit data, a total of 128 bytes.

[6]The main memory subsystem of the MS5000 supports 2-kbyte pages. With 512×512 images, two row-time accesses are necessary to read an input 8×8 block (8 bits/pixel) and four row-time accesses are necessary to write back the 8×8 results (16 bits/pixel). With two cycles per memory access and 64-bit data width, pure data transfer would take 48 cycles. The extra cycles necessary for six row-time accesses are about 90. Thus, it takes 2.16 cycles per output pixel on each ADSP. With four ADSPs running in parallel, the total data transfer rate for the MVP becomes 8.63 cycles per output pixel, which is greater than the combined processing core's 6.75 cycles per output pixel.

Table 1: Summary of several optimization steps for the 8×8 2-D DCT program and its performance on a 512×512 input image

Program Organization	Performance	Speed-up[a]
First version	44.5 ms	1.00
Two ADSP-level functions	16.3 ms	2.73
Combined processing core	11.6 ms	3.84
Four 8×8 blocks at a time	9.53 ms	4.67

[a]Speed-up over the first version

observed by a single dot in the middle of the lower row of Fig. 11. This single checkpoint execution was performed much later than the packet transfer completion (which is represented by the end of thin line in the upper row.). The execution time for the fourth version program was 9.53 ms.

Table 1 summarizes the four versions of the 2-D DCT program. The overall speed-up of 4.67 was achieved by tuning the performance based on the monitoring results from the MPM. The final version's execution time of 9.53 ms is slightly larger than the theoretical minimum of 8.84 ms. However, further performance improvement would be much more difficult, and the achievable performance gain would be small compared to the amount of effort necessary.

5 Conclusion

Achieving good performance on high performance single-chip multiprocessor DSPs is challenging. There is almost no end in optimizing any complex algorithm. We have discussed the need, requirements and functionality of performance monitoring tools for the DSP-based systems, and have developed the MPM, an integrated MVP performance monitor as an example.

The MPM helps the programmer in identifying the DSP-specific performance bottlenecks. It presents the monitored results in a way that gives the programmer a clear view of program execution and areas of potential improvement so that the overall performance can be improved and optimized with the judicious use of the MPM and gained experience/intuition. The tight integration between the familiar functional debugger and performance monitor makes the MPM easy and efficient in fine-tuning the DSP applications. We have demonstrated that the performance of image computing algorithms can be improved significantly using the MPM with a reasonable amount of effort.

References

[1] T. Anderson and E. Lazowska, "Quartz: a tool for tuning parallel program performance," in *Proc. 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1990, pp. 115-125.

[2] D. Reed, R. Aydt, R. Noe, P. Roth, K. Shields, B. Schwartz and L. Tavera, "Scalable performance analysis: the pablo performance analysis environment," in *Proc. Scalable Parallel Libraries Conference*, 1993, pp. 104-113.

[3] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam and T. Newhall, "The paradyn parallel performance measurement tool," *Computer*, vol. 28, no. 11, pp. 37-46, 1995.

[4] J. Leonard, "EDN's 1994 DSP-chip directory," *EDN*, vol. 39, no. 12, pp. 75-135, 1994.

[5] M. El-Sharkawy, *Signal Processing, Image Processing and Graphics Applications with Motorola's DSP96002 Processor*, PTR Prentice-Hall, Englewood Cliffs, NJ, 1994.

[6] K. Guttag, R. Gove, and J. Van Aken, "A single-p multiprocessor for multimedia: The MVP," *IEEE Computer Graphics & Applications*, vol. 12, no. 6, pp. 53-64, 1992.

[7] D. Reed, "Performance instrumentation techniques for parallel systems," in *Performance Evaluation of Computer and Communications Systems*, L. Donatiello and R. Nelson, Eds., Lecture Notes in Computer Science **729**, Springer-Verlag, Berlin, 1993, pp. 463-490.

[8] D. Krumme and A. Couch, "Integrated debugging and performance monitoring for parallel programs," in *Proc. 15th Annual International Computer Software and Applications Conference*, September 1991, pp. 317-318.

[9] W. Lee, Y. Kim, R. Gove and C. Read, "MediaStation 5000: integrating video and audio," *IEEE MultiMedia*, vol. 1, no. 2, pp. 50-61, 1994.

[10] B. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-32, no. 6, pp. 1243-1245, 1984.