Parallel Media Processors for the Billion-Transistor Era

Jason Fritts, Zhao Wu, and Wayne Wolf
Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544
{jefritts, zhaowu, wolf}@ee.princeton.edu

Abstract

This paper describes the challenges presented by single-chip parallel media processors (PMPs). These machines integrate multiple parallel function units, instruction execution, and memory hierarchies on a single chip. The combination of programmability and high performance on data parallelism is necessary to meet the demands of next-generation multimedia applications. Many research issues must be solved to realize the full potential of programmable media processors. This paper provides both a survey of research trends and issues in architecture and compiler design for programmable media processors, and an exploration of the potential performance of media processors over the next decade.

Keywords: media processor, PMP, billion transistors, parallel architecture, cluster scheduling, data partitioning

1. Introduction

This paper explores the parallel processing challenges presented by single-chip media processors. Advances in VLSI technology will make possible chips with one billion transistors within a decade [1]. There are many multi-media applications that are quite capable of absorbing this increase in available computational power. With access to a billion transistors, architectures will become feasible that will make many of these applications possible. But advances in VLSI technology will not result in advanced parallel media processors unless a variety of other research issues in architecture and compiler design are also addressed.

Current multimedia applications manipulate media such as images, video sequences, and audio channels. Many new applications, such as video libraries, MPEG-4, and MPEG-7, require a great deal of video processing using advanced algorithms [2]. To permit these new methods, multimedia is moving away from simple channel and frame-based representations towards an object-based representation of multimedia. These objects describe real-world objects, each with its own audio, visual, and graphical characteristics specifying its spatial and temporal behavior. This representation enables higher compression rates, more freedom for interactive media, and content-based processing.

The new freedom and flexibility of the object-oriented representation introduces much greater processing demands and irregularity than seen in the first generation of multimedia. This is especially true for video and computer graphics, the most data intensive media. Adequate support for the next generation of media processing will require the flexibility and computing power of high-level language (HLL) programmable media processors.

Support for the first generation of multimedia was predominantly in the form of either application-specific hardware or multimedia extensions to general-purpose processors [3]. Both methods provide support for multimedia at low cost, but neither provides both the flexibility and performance necessary for future multimedia applications. Application-specific hardware provides excellent performance, but has limited flexibility and is unable to support evolving and future applications. General-purpose processors with multimedia extensions provide the flexibility, but are designed primarily for general-purpose applications and cannot adequately support the distinctive multimedia workloads, which are characterized by large amounts of streaming data, high computation rates, and extensive data parallelism. For the next generation of multimedia, unique processors for multimedia, *parallel media processors (PMPs)*, will prove the best alternative as they can offer both the performance and flexibility through specialized high-speed, highly parallel architectures that are programmable using high-level languages.

Some programmable media processors have started to appear in the marketplace with DSP (digital signal processing) and VLIW (very long instruction word) architectures [3], but these early processors achieve only a fraction of their potential performance. They do not have the frequency or parallelism for high throughput and offer only limited programmability through special libraries or programming paradigms. The increasing frequencies and numbers of transistors over the next decade will help solve these problems, but the question remains on how to utilize these resources most effectively.

To achieve their full potential, media processors will undergo some significant changes in the coming decade. There are three primary differences that are expected between existing and future media processors:

- much more on-chip memory
- wider processors with more functional units
- more regular architectures (less dedicated hardware)

Large, aggressive on-chip memory hierarchies are necessary to accommodate the high data rates and minimize penalties from large external memory latencies. Much wider processors are required to provide the parallelism necessary for meeting the throughput demands of many multimedia applications. More regular architectures are needed both for greater flexibility over a wide range of applications, as well as better high-level language programmability. These are considerable differences and much research in architecture and compiler design is required to realize the potential of future media processors.

This paper continues with an examination of the potential of future PMPs and a survey of the current trends and research issues being explored towards achieving that potential. Section 2 will examine some architecture issues including the competing factors of high parallelism and high frequency and the open issues regarding memory hierarchy design. Section 3 will explore the compiler issues including extraction of data parallelism and parallel compilation problems such as cluster scheduling and data partitioning. Section 4 will perform experiments to investigate the potential performance of PMPs over the next decade and illustrate the limited capabilities of current compilers with respect to multimedia. Section 5 closes with the conclusions.

2. Architecture issues

Proceeding into the next century, technology is approaching the billion-transistor era. What does this imply for next-generation media processors? While it is certainly possible to put more functional units on chip, using the extra silicon estate for memory will likely prove more productive. First, functional units are small, consuming only a minor portion of the entire chip area, even for processor widths of 16 to 32 issue slots. Secondly, many multimedia applications, particularly video and graphics, tend to access memory frequently and intensively. For many video applications, it was found that memory is the bottleneck of the entire system [4]. As the performance gap between processors and memories continues to widen, its impact on media processors will become significant. Consequently, it is expected that the increasing numbers of transistors will likely be devoted primarily to larger on-chip memory hierarchies.

In the remainder of this section, the design issues surrounding the processor core and memory hierarchy will be explored. Also presented is the design of a datapath expected to resemble those for future media processors. This design shall be used for experiments in Section 4 that estimate the potential performance of future PMPs.

2.1. Datapath

A variety of architecture alternatives exist for parallel media processors. Many researchers have been speculating on various architectures that will become feasible with billion-transistor chips. Included in some of the proposals are wide superscalar designs, trace processors, single-chip multiprocessors, simultaneous multi-threading, and array processors [5]. It is difficult to say whether these architecture models will be desirable for media processing, but there have been additional proposals specifically for media processing, including vector IRAM [5], simultaneous multi-threading [6], and array processors [7]. While the question of architecture style for future PMPs certainly remains an open issue, one truth exists for any design: media processors will need to achieve high degrees of processing throughput to meet the intense computation and data rates found in many multimedia applications.

High parallelism and high frequency will both be necessary for achieving the processing throughput required in many multimedia applications. The data parallelism inherent in most multimedia can enable significant parallelism, but even with this parallelism, a high frequency processor is still necessary for meeting the demands. One example of the computational requirements on PMPs is given by the MPEG-2 encoder, which can require in excess of one billion operations to compress only a few frames of 720x480 resolution video. This means many billions of operations per second to achieve real-time MPEG-2 encoding. MPEG-4 encoding will exacerbate this situation, as it must also perform segmentation of video frames into objects.

Unfortunately, high parallelism and high frequency are counter-productive goals. Very high frequency processors are quite feasible with low issue widths, but as parallelism increases, the demands on the register file, memory, and datapath grow significantly, often exponentially. Each additional issue slot added to a processor adds 3-4 ports to the register file, potentially a memory port, one or more bypass paths, and extra function units and wires that increase area and loading. Consequently, increased parallelism results in decreased frequency for conventional architectures. To achieve both high parallelism and high frequency, more distributed architectures are required.

An effective way to increase the parallelism with much less impact on frequency is through a clustered architecture. A clustered architecture divides the architecture into disjoint groups of issue slots. Each group has 2 to 4 issue slots, its own register file and possibly its

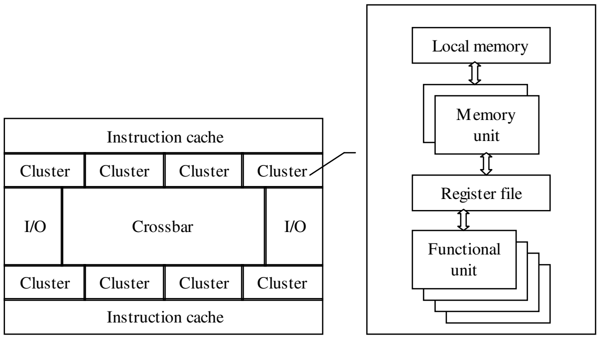

Figure 1. Clustered architecture

own local memory or cache. This distributed design allows high frequency by limiting the register ports, memory ports, bypass paths, and area within each cluster. The clusters are all connected by an interconnect network, which passes results between clusters as required. Any number of clusters may be used to achieve the desired parallelism, with the primary impact on frequency coming from the increased demands on the interconnect network, the memory hierarchy, and the control logic. In essence, a clustered architecture is a group of processors that share a common control stream and are tightly connected by a low-latency interconnect.

An example of an aggressive clustered architecture we have proposed for media processing is a 32-issue cluster architecture that is divided into 8 clusters of 4 issue slots per cluster, as shown in Figure 1. These clusters are homogeneous clusters, each offering 4 ALUs, 2 memory ports, 1 multiplier, and 1 shifter. Full bypassing is provided among all issue slots within the cluster. Local storage consists of a register file with 128 registers and 32 KB of local memory in each cluster. Results from other clusters can be easily obtained using a single-cycle 32x32 crossbar network that fully connects all issue slots between clusters. Architecture simulations indicate frequencies of up to 650 MHz using 0.25 μm technology [8]. This architecture design will be used for the performance experiments of future PMPs in Section 4.

2.2. Memory hierarchy

The memory hierarchy of a HLL programmable media processor is an ill-defined area. Whereas general-purpose processors use caches, typical DSPs often use local memory with some form of prefetching, such as DMA or stream buffers. Our initial studies on video application traces [9] have shown good performance with hybrid memory architectures that combine a stream buffer with cache. However, these studies are based on trace-driven simulations that assume perfect branch prediction and memory disambiguation.

Previous research in multimedia memory hierarchies includes a study performed by Zucker, et al. [10] at Stanford. Using the JPEG and MPEG multimedia applications, the study examined three prefetching techniques for streaming data: stream buffer, stride prediction table, and a hybrid of the two called a *stream cache*. Compared to a cache system without any prefetching, all the techniques were effective at eliminating many of the cache misses, though the effectiveness of each varied according to the size of the cache used. For small caches, the stream buffer and stream cache were more effective, while the stride prediction table performed best for large caches sizes.

In addition to providing support for streaming memory, the large data rates can place significant burdens on external memory, so the external memory bandwidth often becomes a bottleneck. To meet the necessary data rates, either memory hierarchies must be designed to reduce the external memory traffic, or scheduling methods will be required to restructure the program for improved data locality [7][11].

Aside from the issue of the type of memory hierarchy to use, there is an added complication of supporting numerous memory accesses each cycle. Large multi-ported memories are not feasible at high frequencies, so highly banked or distributed memories are required instead. This may even entail separate local memory for each cluster, with memory coherency handled by the compiler and/or higher levels of the memory hierarchy. One study provided an interesting alternative by combining a clustered architecture with a stream buffer-like prefetching structure called a *stream register file* [12]. This architecture and memory hierarchy enable a high-frequency clustered architecture coupled with the benefits of prefetching support. However, their scheme also requires a special programming paradigm, which is undesirable for generic HLL programmability.

The memory hierarchy for PMPs is poorly understood. It is unknown whether cache, memory with prefetching, cache with prefetching, or even some other novel memory structure will prove most suitable for multimedia. Furthermore, the problems of reducing external memory bandwidth and supporting numerous memory accesses per cycle are also areas open areas. Considerable research remains in the area of memory hierarchy design for media processors, with the only definitive facts being that:

- PMPs require aggressive memory hierarchies to support the large amounts of data, and
- the increasing availability of transistors on chip over the next decade will allow interesting multi-level memory hierarchy alternatives.

Many research issues regarding the media processor datapath and memory hierarchy have been presented that define key areas for realizing efficient media processor designs. These are far from the only areas of research, however. Many additional questions exist, such as whether smaller datapath sizes (less than 32 bits) would enable faster video signal processing since video signals usually operate predominantly on 8-bit and 16-bit data types. Also, the question of supporting mixed-signal multimedia is an issue. For example, video and images use small integer data types whereas graphics more commonly uses single-precision floating point. These and many other issues require further study.

3. Compiler issues

The design of HLL programmable media processors requires an aggressive optimizing compiler and a high frequency parallel architecture that complement each other to meet the intense demands of future multimedia. The high degrees of parallelism and large volumes of streaming data within multimedia applications have been well studied, but much research remains towards the design of media processors that are both programmable and can support these computation and data intensive applications. The architecture and compiler designs must be balanced so that the compiler can take full advantage of all architectural resources.

The compiler is a critical component in any HLL programmable processor. This is no less true for PMPs. The high degrees of data parallelism offer optimistic opportunities for achieving high levels of throughput, but they also offer additional problems for the compiler to tackle. These problems can be broken down into two primary categories: a) extraction of the data parallelism, and b) scheduling for highly parallel architectures.

3.1. Extracting data parallelism

The foremost problem with a compiler for PMPs is extracting the parallelism from the applications. Numerous studies have shown the existence of considerable parallelism in multimedia applications [13][14]. Taking advantage of this parallelism, significant speedups have been achieved in many implementations, including application-specific hardware [3], a software approach for multiprocessors [15], and several proposed designs for media processors [7][12][16]. However, it has not yet been shown how compilers can successfully extract similar levels of parallelism.

The problem lies in the level of granularity of the parallelism available in multimedia applications. The compilers of today are good at finding instruction level parallelism (ILP), but the data parallelism in multimedia applications is of a coarser granularity than ILP, so current compilers are unable to effectively exploit it. While multimedia applications typically have considerable data parallelism, they usually have no more ILP than general-purpose applications, as found in a recent compiler-directed workload evaluation [17]. Successful extraction of the data parallelism by compilers will be crucial for the success of PMPs.

Data parallelism is the parallelism that exists between data elements that have little or no processing dependency between them. It may exist between any data elements, but most commonly occurs between data elements that are not in close proximity to each other in either the spatial or temporal domains. The independence of such data allows computations on them to occur in parallel.

While two data elements with independent processing may be relatively close together in the spatial or temporal domains, the computations on these elements are typically several hundreds or thousands of operations apart in sequential program order. This presents a problem for compilers, which are most effective at finding the instruction level parallelism within scheduling windows comprised of a limited number of sequential operations. The size of these instruction sequences is usually no more than a hundred instructions, which is much smaller than the range of operations for most data-parallel elements. Consequently, current compilers are unable to extract any significant degrees of parallelism from multimedia applications.

An example of this is the 2-D discrete cosine transform (DCT) algorithm, which is a critical kernel for both JPEG image compression and MPEG video compression. This algorithm takes an 8x8 block of an image or video frame and turns the 2-D spatial information into the corresponding 2-D frequency domain. A straightforward implementation of this algorithm encompasses a four-level nested loop where the two outer loops process the 64 elements of an 8x8 block. The two inner loops perform the processing on a single element. It can be seen that the statement 'y[k][l] = y[k][l] + x[i][j] * c[i][k] * c[j][l]' creates a loop carried dependence in the two inner loops. However, there is no such dependence in the outer loops. Each iteration in the two outer loops is independent of all other iterations, so the 64 iterations of the two outer loops can be run in parallel. Additionally, the DCT is independent over all separate 8x8 blocks in each image or video frame, allowing for even greater degrees of parallelism. The compiler, however, has difficulty seeing the parallelism. There are nine operations in the inner loop, so an iteration of the two inner loops over a single element has at least 9*64 = 576 operations. This minimum distance between two data-parallel elements is much greater than typical scheduling window sizes in compilers, so a compiler is unlikely to find the data parallelism.

The issue of finding data parallelism in multimedia applications is closely related to the issues of automatic parallelization in multiprocessors. Significant research has been performed in parallelizing compilers for taking ad-

```
for (k = 0; k < 8; k++)
   for (l = 0; l < 8; l++)
   {

   y[k][l] = 0.0;

   for (i = 0; i < 8; i++)
      for (j = 0; j < 8; j++)
         y[k][l] = y[k][l] + x[i][j] * c[i][k] * c[j][l];
   }
```

Figure 2. Straightforward implementation of the 2-D DCT Algorithm

vantage of loop-level parallelism at various loop levels in program code [18][19]. However, these methods were initially designed for large-scale parallel machines that provide coarser granularities of parallelism, and will need to be adapted to the finer granularities of parallelism in single-chip parallel media processors.

3.2. Compiling to highly parallel architectures

As mentioned earlier, highly parallel architectures require a distributed design to provide the high frequencies necessary for multimedia. For single-chip media processors, it is expected that the distributed architectures will be tightly coupled like the clustered architecture proposed earlier. This tight coupling enables clusters to communicate data and results to each other with a minimal latency of only one or two cycles.

Compiler scheduling is complicated by clustered architectures because operation results are not uniformly available to the entire processor at the same time. Instead, each result is available first on the cluster that performed the operation and must be copied to other clusters via an inter-cluster *copy* operation as needed. Other clusters needing this result must incur additional latency while *copy* operations transfer the result over the interconnect network. To keep the additional latencies from impacting execution time, it is desirable to schedule critical dependent operations on a single cluster and minimize the communication between clusters.

A number of studies have been performed in the area of scheduling for clustered architectures. Two such works include list scheduling of acyclic dependency graphs [20], and modulo scheduling of cyclic dependency graphs [21]. These studies have produced good methods for scheduling operations over multiple clusters and scheduling the necessary *copy* operations, while minimizing the impact of inter-cluster communication on overall performance. Typically the performance degradation using a clustered architecture is no more than 20-30% compared to a non-clustered architecture of the same issue width.

There is a benefit to scheduling to a clustered architecture for media processing versus general-purpose processing, in that a compiler for media processing will be able to exploit data parallelism. Whereas instruction level parallelism defines parallelism only at the level of individual operations, data parallelism defines parallelism at the level of a group of operations. By definition, data parallelism requires that the processing between data elements be relatively independent, which indicates the groups of operations processing separate data elements are also relatively independent. The relative independence of each group corresponds well with the independence of each cluster, so each group can be scheduled onto a separate cluster. The independence of separate groups eliminates most inter-cluster communication, so there is negligible degradation in overall performance. Using data parallelism, a PMP compiler can perform cluster scheduling with little impact on the performance, and still maintain exceptional parallelism.

A second problem that exists in compiling to a clustered architecture is the need for data partitioning. As discussed in Section 2.2, supporting numerous memory accesses per cycle will likely require either a highly banked or distributed memory structure. However, it is expected that a highly banked memory will only work for up to 4 (or at most 8) parallel memory accesses, because with any more parallel memory accesses the additional logic for control and bank conflict checking will be prohibitive at high frequencies. Consequently, the use of separate local memory structures in each cluster is more likely for a 32-issue processor with 16 parallel memory accesses, as proposed in Section 2.1. Effective scheduling to clusters with separate memories requires intelligent partitioning of data among the clusters to minimize the communication overhead.

An ideal example of data partitioning is the DCT algorithm examined earlier. Recall from Figure 2 that the algorithm uses 4 levels of loops to compute the DCT over a single 8x8 block. The two outer loops (indices *k* and *l*) have independent iterations, while the two inner loops (indices *i* and *j*) have dependent iterations. Also, the processing of separate 8x8 blocks is independent as well. The data parallelism in this example therefore exists at three different levels:

a) single iteration of the *l* loop,
b) single iteration of the *k* loop (all 8 iterations of the *l* loop), or
c) a single 8x8 block (all 8 iterations of the *k* loop)

In each case, data in the *c*, *x*, and *y* arrays is either read or written, but different amounts of these arrays are required depending upon the level of data parallelism. The appropriate data elements required from each array for the different data parallelism levels are shown in Figure 3.

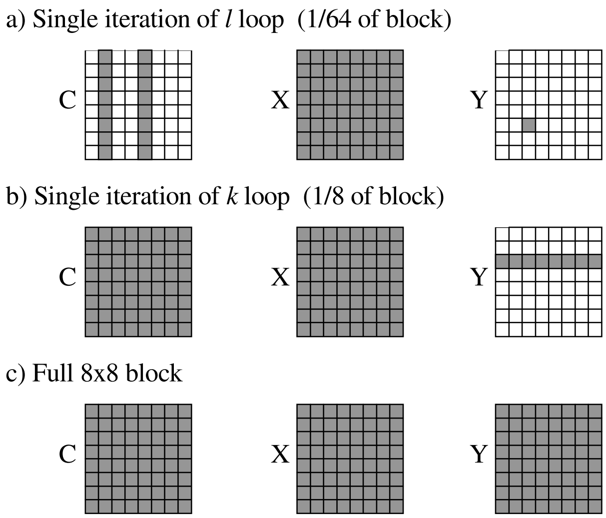

Figure 3. Data required for DCT algorithm according to level of data parallelism

From the figure, it can be seen that alternatives a) and b), where the data parallelism size is less than a full 8x8 block, require a much finer granularity of data. This is particularly evident in the *y* array. Another important distinction of the *y* array is that it is a data array to which data is being written, not simply read. Finer levels of granularity are easier to accommodate when the data is only being read as it can be shared among multiple clusters without violating memory coherency. Data sets that may be written cannot be shared without potentially violating memory coherency. In the case of the DCT, the entire *y* array cannot be easily shared among the clusters. Using alternative a) or b), it may instead be necessary for one cluster to act as a 'collector', a cluster that alone has write privileges to the *y* array, which collects the *y* results from other clusters and stores them into the *y* array.

The problem of data partitioning by the compiler is a complicated issue that still requires much research. The DCT is a relatively trivial example. For algorithms with more irregular data access, the problem can become very convoluted. Also, many tradeoffs exist between the level of data parallelism and the granularity of the data set. While coarser granularities of data are typically desirable, larger levels of data parallelism may not always be the most efficient. Furthermore, coarser granularities of data may also lead to extensive read sharing that might consume much more local memory than another level data parallelism might require. Examples are tables or arrays of constants such as the *c* array in the DCT. MPEG-2 in particular has numerous large look-up tables for variable bit rate coding which would take up considerable space if a copy was required in each cluster's local memory.

Media processors are gradually moving to wider architectures, presenting new problems for compilers to tackle. Foremost among these is extraction of data parallelism, since effective support of multimedia requires high parallelism as well as high frequency to achieve the necessary throughput. Once the data parallelism is found, there are also issues to resolve for compiling to highly parallel architectures, including cluster scheduling and data partitioning. Again, these are not the only outstanding problems. Other problems such as the role of the compiler in handling streaming data are still open issues, but the problem of finding parallelism in multimedia applications is crucial to the success of programmable media processors.

4. Experiments

This section describes the results of experiments that track the potential performance of parallel media processors over the next decade. The first experiment compares the performance of a trace-driven simulator and a compiler to show the considerable difference between the currently achievable parallelism and the potential parallelism. To bridge this gap and fully realize the potential parallelism, compilers will need to exploit the data parallelism available in multimedia applications, as discussed in section 3. To determine where the data parallelism resides in multimedia applications, a second experiment examines the granularities of parallelism using trace-driven simulation over different-sized scheduling windows. This experiment demonstrates the level of granularity of data parallelism that a compiler must be able to extract. A final experiment tracks the expected progress of PMPs as transistor densities and processor frequencies continue to improve over the next decade.

4.1. Trace-driven simulation vs. compilation

There are many ways to evaluate an architecture, trace-driven simulation and compilation being the two most popular methods. In trace-driven simulation, a program is instrumented and then run on a processor to generate a program trace, which contains all the operations used during the execution of the program. To simulate an architecture, the simulator takes in the trace and attempts to schedule the operations to achieve the greatest parallelism. Meanwhile the simulator performs a variety of experiments such as cache and pipeline simulation. The procedure on an SGI workstation is depicted in Figure 4. Details of the trace-driven simulation method can be found in our previous work [14]. The second method of evaluation is compilation. The basic idea is to compile the program for the target architecture and then simulate execution of the assembly code.

Both approaches have pros and cons. While trace-driven simulation is fast and relatively easy to implement,

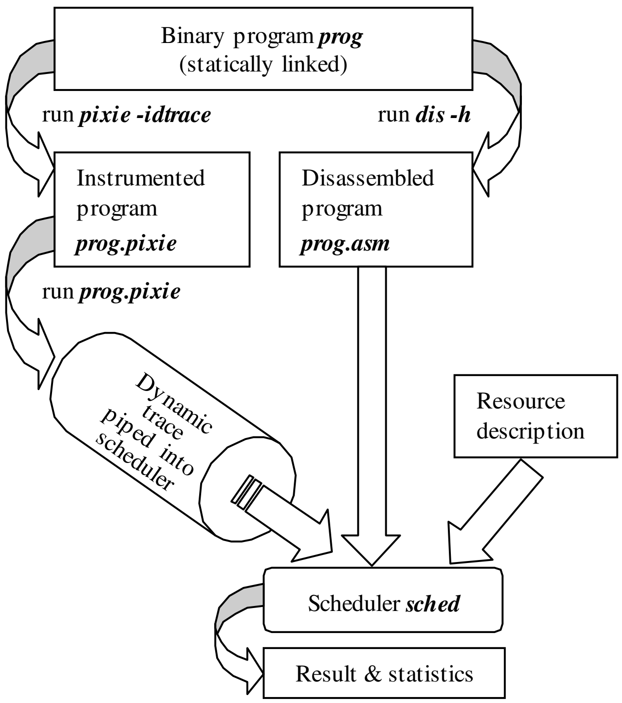

Figure 4. Flow chart of trace-driven simulation on SGI workstations

it has some difficulties dealing with issues like branch prediction and memory disambiguation. This is because the input trace only contains instruction and data addresses. When data is accessed its address is immediately available in the trace, but in reality that address may not be disambiguated from other accesses, which could limit the actual parallelism. On the other hand, despite providing precise results in terms of parallelism measurement, a good retargetable compiler is very sophisticated and hard to develop. Moreover, it takes considerably more time to compile and simulate real applications. Generally speaking, trace-driven simulation may over-estimate the parallelism (due to lack of information about branch prediction and address disambiguation), whereas a compiler usually under-estimates the parallelism (due to lack of optimality in the compiler).

For multimedia applications, the difference in the performance of these two methods is considerable. To illustrate this disparity, three applications, H.263, MPEG-2, and MPEG-4, are simulated using both the trace-driven simulator and the IMPACT compiler [22]. Both methods targeted the same media processor architecture, the 32-issue 8-cluster architecture described in Section 2.2. The IMPACT compiler optimized the code with aggressive ILP optimizations, but was only able to target a 32-issue unclustered architecture, as it does not currently support cluster scheduling.

Table 1. Operation latencies

Operation	Latency
ALU	1
Memory	2
Shift	1
Multiply	7
Divide	34

The target media processor architecture has 4 ALUs, 2 memory units, 1 multiplier, and 1 shifter per cluster. It contains 128 registers and 32 KB of memory per cluster. The latencies of the operations are modeled after the Alpha 21264 [23], as presented in Table 1. Floating-point is not currently supported in this architecture.

The results of this experiment, shown in Figure 5, display a significant difference between the compiler and trace-driven simulator. The parallelism from the compiler varies from 1.5 to 2.6 instructions per cycle, indicating that no data parallelism, only ILP, is being exploited. The trace-driven simulator indicates much greater parallelism, though we do not believe these results over-estimate the available parallelism. It is expected that with improved compiler technology, particularly with compiler algorithms that can take advantage of the data parallelism exhibited by multimedia, media processors can operate at parallelism levels close to the potential shown by the trace-driven studies.

4.2. Level of data parallelism

To determine the potential performance obtainable by a compiler from data parallelism, an experiment was performed with the trace-driven simulator, examining the parallelism available in the application traces using different scheduling window sizes. As indicated in Section 3.1, the level of granularity of data parallelism in multimedia applications is on the order of many hundreds

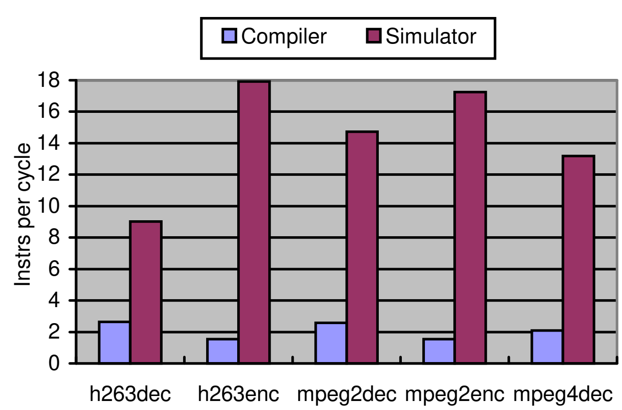

Figure 5. Average instruction issue rate for various resources in an MPEG-2 encoder

or thousands of operations. By varying the scheduling window size, it is possible to determine the amount of parallelism available in different-sized section of the program trace. Comparing the parallelism available in different-sized scheduling windows will illustrate the different granularities of parallelism in the application.

The experiment was performed using the MPEG-2 encoder and decoder application traces, varying the scheduling window size from 16 to one billion operations. The results are shown in Figure 6. At the smaller scheduling window sizes, only instruction level parallelism is available, so the parallelism from ILP is 8 for the encoder, and 1 for the decoder. Once the instruction scheduling window size increases above 4k operations, data parallelism begins to become available, and the parallelism increases to 14 for both the encoder and decoder. Consequently, for the MPEG-2 application, the results indicate the minimum level of data parallelism is on the order of 4k+ operations, and that significant data parallelism occurs at scheduling window sizes of around 64-256k operations.

While the observed granularity of data parallelism in these applications seems somewhat large, the true granularity of the data parallelism is actually a bit smaller than this experiment indicates. The trace-driven simulator is unable to define the actual level of data parallelism, because it schedules the operations in the trace exactly as they occur and does not perform any optimizations on the code, as a compiler would. Many parallelizable loops may not be run in parallel because of the loop-carried dependences from loop induction variables. Therefore, the trace-driven simulator is not able to model the actual level of data parallelism. The effect on the results in Figure 6 would be to shift the slope at 4-64k operations a bit to the left. The actual minimum level of data parallelism will likely be on the order of 1-4k operations.

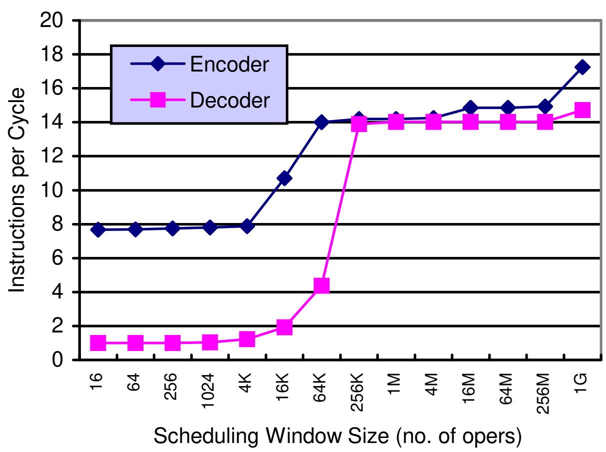

Figure 6. Instructions per cycle of MPEG-2 coding for different scheduling window sizes

4.3. Increasing numbers of on-chip transistors

Having realized the importance of memory systems, many CPU designers are putting much more on-chip memory in their microprocessors (e.g. the Intel mobile Pentium II has 256 KB second-level cache integrated on the same processor die). With the increasing silicon resources during the next decade, it is expected that media processors will use the majority of the additional real estate for larger on-chip memory hierarchies. More on-chip memory will allow higher datapath-to-memory bandwidths and lower average latencies for memory accesses.

As mentioned earlier, choosing the appropriate memory structures and devising innovative memory architectures for application-specific systems are subjects of active research. For purposes of this experiment, we will only consider on-chip cache memories, as this is the most widely used and thoroughly studied memory model. Evaluations of other memory types require corresponding changes in the compilers and simulators, which are not addressed in this paper due to space limitations. Comparisons of different memory architectures and their impact on multimedia applications can be found in the literature [4][10].

In all cache simulations, a block size of 64 bytes and two-way set associativity are used, as this configuration outperforms others cache configurations of similar area [9]. The datapath model is that presented in Section 2.1, i.e. 8 clusters, each containing 128 registers, 4 ALUs, 2 memory units, 1 multiplier and 1 shifter.

In the deep sub-micron era, interconnect wires also play an essential role in addition to transistors, so we use area instead of transistor count as a metric. The area measurements are based on Dutta's work [8]. Using a 0.25μm CMOS technology, Dutta designed parameterizable versions of some key modules, including

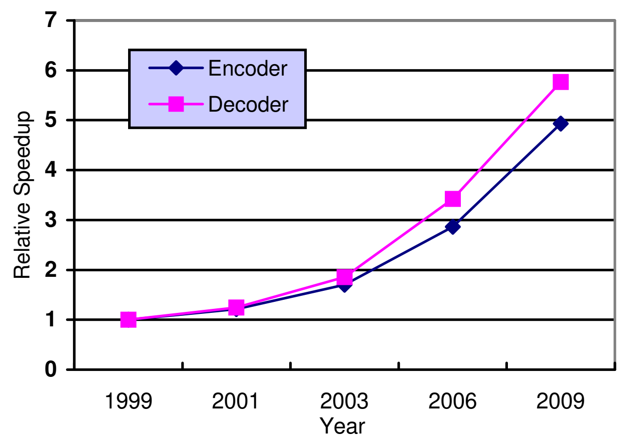

Figure 7. Relative speedup of MPEG-2 coding for increasing on-chip cache over next decade

memory. The simulation results, taking into account clock rate increase according to the *National Semiconductor Technology Roadmap* [1], are shown in Figure 6. As can be seen, the curves are slightly better than linear. As chip area (number of transistors) and speed increase, so does performance. We assume that a 32 KB on-chip cache is used in 1999 and the size doubles every two or three years, and finally reaches 1024 KB after ten years. Notice that this only considers a single-die, so the actual performance could be even higher with more aggressive techniques such as a multi-chip module (MCM).

5. Conclusions

Advances in VLSI technology are making possible single-chip parallel media processors (PMPs). These processors offer significant challenges in parallel processing: architectures must be able to operate at high frequencies with high degrees of parallelism while making efficient use of memory hierarchies; compilers must be able to take advantage of the large amounts of data parallelism available in multimedia and perform efficient scheduling onto a clustered architecture. Parallel media processors will be early examples of single-chip parallel processors and will offer significant research opportunities. Such chips will enable continued growth and many new advances in the multimedia computing industry.

6. Acknowledgements

This work was funded by the New Jersey Center for Multimedia Research and by the National Science Foundation.

7. Bibliography

[1] "*The National Technology Roadmap for Semiconductors*", 1997 Edition, Semiconductor Industry Association, 1997.

[2] Wayne Wolf, Yiqing Liang, Michael Kozuch, Heather Yu, Michael Phillips, Marcel Weekes, and Andrew Debruyne, "A digital video library on the World Wide Web," *Proceedings, ACM Multimedia '96*, ACM Press, 1996.

[3] Zhao Wu and Wayne Wolf, "Parallel Architectures for Programmable Video Signal Processing," Technical Report, Princeton University, 1998.

[4] Zhao Wu and Wayne Wolf, "Design study of shared memory in VLIW video signal processors", *Proc. IEEE Int'l Conf. on Parallel Architectures and Compilation Techniques*, pp. 52-59, Oct. 1998.

[5] C. Kozyrakis and D. Patterson, "A New Direction for Computer Architecture Research," *IEEE Computer*, November 1998, vol. 31, no. 11, pp. 24-32.

[6] M. Berekovic, P. Pirsch, and J. Kneip, "An Algorithm-Hardware-System Approach to VLIW Multimedia Processors," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 20, pp. 163-180, 1998.

[7] Y. K. Chen and S. Y. Kung, "Multimedia Signal Processors: An Architectural Platform with Algorithmic Compilation," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 20, pp. 181-204, 1998.

[8] Santanu Dutta, "VLSI issues and architectural tradeoffs in advanced video signal processors", Ph.D. Thesis, Princeton University, Nov. 1996.

[9] Zhao Wu and Wayne Wolf, "Study of cache system in video signal processors", *Proc. IEEE Workshop on Signal Processing Systems*, pp. 23-32, Oct. 1998.

[10] D. Zucker, M. Flynn, and R. Lee, "A comparison of hardware prefetching techniques for multimedia benchmarks", *Technical Report CSL-TR-95-683*, Stanford Univ. Dec. 1995.

[11] C. Kulkarni, K. Danckaert, F. Catthoor, and M. Gupta, "Interaction Between Data Parallel Compilation and Data Transfer and Storage Cost for Multimedia Applications," IBM Research Report, February 1999.

[12] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucek Khailany, Abelardo Lopez-Lagunas, Peter R. Mattson, and John D. Owens, "Media Processors Using Streams," *SPIE Photonics West: Media Processors '99*, San Jose, California, January 28-29, 1999, pp. 122-134.

[13] Heng Liao and Andrew Wolfe, "Avaliable Parallelism in Video Applications," *Proceedings of the 30th Annual International Symposium on Microarchitecture*, December 1997.

[14] Zhao Wu and Wayne Wolf, "Trace-driven studies of VLIW video signal processors", *Proc. 10th ACM Symp. on Parallel Algorithms and Architectures*, pp. 289-297, June 1998.

[15] Angelos Bilas, Jason Fritts, and Jaswinder P. Singh, "Real-Time Parallel MPEG-2 Decoding in Software," *11th International Parallel Processing Symposium*, Geneva, Switzerland, April 1997.

[16] A. Wolfe, J. Fritts, S. Dutta, and E. S. T. Fernandes, "Datapath Design for a VLIW Video Signal Processor," *3rd International Symposium on High-Performance Computer Architecture*, San Antonio, Texas, Feb. 1997, pp. 24-35.

[17] Jason Fritts, Wayne Wolf, and Bede Liu, "Understanding multimedia application characteristics for designing programmable media processors," *SPIE Photonics West, Media Processors '99*, San Jose, CA, January 1999, pp. 2-13.

[18] U. Banerjee, R. Eigenmann, A. Nicolau, and D. Padua, "Automatic Program Parallelization," *Proceedings of the IEEE*, February 1993, vol. 81, no. 2, pp. 211-243.

[19] M. Hall, S. Amarasinghe, B. Murphy, S. Liao, and M. Lam, "Detecting Coarse-Grain Parallelism Using and Interprocedural Parallelizing Compiler," *Proceedings of Supercomputing '95*, December 1995.

[20] Sanjeev Banerjia, "Instruction scheduling and fetch mechanisms for clustered VLIW processors," PhD thesis, Department of Electrical and Computer Engineering, North Carolina State University, 1998.

[21] Erik Nystrom and Alexandre E. Eichenberger, "Effective Cluster Assignment for Modulo Scheduling," *Proceedings of the 31st Annual International Symposium on Microarchitecture*, December 1998.

[22] IMPACT compiler group: http://www.crhc.uiuc.edu/Impact

[23] L. Gwennap, "Digital 21264 sets new standard," *Microprocessor Report*, 10(14), October 1996.