Available Parallelism in Video Applications

Heng Liao, Andrew Wolfe
Princeton University

Abstract

*Most recent research in instruction-level parallelism has focused on general-purpose applications such as the SPEC benchmarks. Many quantitative experiments have been performed over the years measuring the impact of different execution models and optimization techniques on these applications. Recently, however, researchers have been developing various ILP architectures for media processors in order to exploit parallelism in audio, video, and graphics applications. It has been assumed that these applications contain far more pot*ential parallelism than general-*purpose code, but there have been few attempts to quantify the available parallelism. In this paper, we present a linear complexity global scheduling algorithm that can process very long traces up to 1 billion operations. Therefore, traces of video applications such as MPEG1, MPEG2, MPEG4 and H.263 encoders and decoders can be analyzed. Using an idealized execution model, speedups of over 1000 have been found in some applications. The experiment shows that eliminating currently identifiable bottlenecks can allow the exploitation of huge amounts of ILP in audio and video applications.*

1) Introduction

Digital video is rapidly becoming a mainstream medium for representing information. Real time video signal processing requires extensive computational power and the evolving nature of video applications has encouraged the development of programmable processors for video and other real-time information media. In order to provide high performance for these applications, a number of instruction-level parallel programmable architectures have been proposed including VLIW architectures [9,10], vector architectures [11], and SIMD extensions to superscalar implementations of sequential architectures [4]. The amount of concurrency within instructions varies greatly among various proposed architectures. Numerous studies have appeared in the literature measuring both practical and impractical bounds on available instruction-level parallelism for general-purpose integer programs and for scientific code, but to date there have been no similar analyses for multimedia applications. This paper describes experiments that measure the available parallelism for several digital videos and one digital audio application.

Oracle experiments [Nicolau] have been used on numerous occasions to study the bounds of instruction-level parallelism available in various types of applications. In many cases, these experiments have developed a poor reputation due to the tendency to misinterpret the significance of the results. An oracle experiment can never derive an actual upper bound on the parallelism available in a particular application. The capabilities of an oracle experiment are much more limited. What such an experiment does do is to construct a model of an application whereby one or more potential performance bottlenecks are identified. The experiment then asks the question "if the performance penalties due to these bottlenecks were completely eliminated, how much performance could be achieved?" From this result, we can derive the maximum average parallelism under the conditions described. The experiment always includes limitations that restrict the maximum performance, such as limits on the amount of work that can be performed by one operation in one cycle. The experiment also only considers the impact of relieving specific known bottlenecks, for example inability to perfectly predict branches or finite memory bandwidth. It is important to recognize that there may exist other hardware or software optimizations, some as simple as better common sub-expression elimination, that can

1072-4451/97 $10.00 © 1997 IEEE

improve performance beyond the level detectable in these experiments. This explains why different researchers have achieved different results studying the same application set. They have made different assumptions about the oracle execution model and generally applied different levels of software optimization.

The primary benefit of an oracle experiment is in guiding further research. If we can identify a set of known bottlenecks and show via an oracle experiment that relieving those bottlenecks significantly improves performance, then we can focus research efforts into hardware and software mechanisms to deal with these known issues. If relieving the known bottlenecks does not provide a meaningful increase in performance, and then research must focus on identification of new bottlenecks, in particular those that must be solved prior to code generation.

Our research uses trace-driven simulation on instruction streams generated by real applications and an efficient scheduling algorithm, capable of searching the entire instruction stream for independent operations. The algorithm can generate the optimal critical path length of the detailed dependency graph (DDG) of the trace with linear complexity. Video applications including MPEG-1, MPEG-2, MPEG-4, and H.263 encoders and decoders are tested. Section 2 describes previous experiments in available parallelism. Section 3 gives a description of the experiment and the scheduling algorithm. Section 4 presents the results found in the experiments and some analysis. Finally section 5 concludes this study.

2) Previous Work

Numerous studies have been done to measure the available parallelism in scientific and general-purpose programs. Nicolau performed one of the first oracle experiments to measure available ILP for a VLIW architecture [3]. Trace streams of up to 50,000 operations were measured and an average speedup of around 90 was observed. Smith used trace-driven simulations to measure the effective limits of multiple instruction issue in superscalar architectures and observed an instruction issue rate of about two instructions per cycle [1]. Wall also tested ILP limit with various assumptions using a wide variety of hardware and software techniques including branch prediction, register renaming, and alias analysis, and concluded that average parallelism rarely exceeds 7 [2].

Audio and video applications use different enough algorithms from either general-purpose or scientific code that they justify a fresh look at available parallelism. Moreover, these applications involve orders of magnitude more calculations than many of the previously studied benchmarks. Depending on the actual algorithms used an MPEG2 encoder will require from a few hundred million to several billion RISC-equivalent operations per frame. Since these applications are intended for continuous operation, we must analyze a sample time period, but it is critical that this sample period reflects the entire algorithm. Most real multimedia applications require multiple computational phases and include deeply nested multi-level loops. Course-grained parallelism often exists between operations that would have been millions of cycles apart in a sequential implementation. This can be exploited within an ILP framework, but will be missed if the analysis samples are too small.

Earlier studies [6] have shown that parallelism is seriously constrained by control flow and memory address ambiguity. Most of the control flows and memory access patterns of video applications are statically deterministic; thus a great deal of parallelism exists within these applications, although it may be difficult to extract. An oracle experiment using extremely large traces (up to 10^9 operations) exposes potential parallelism at the basic block, loop, and/or procedural levels.

The limitations of local scheduling can be seen in recent attempts to study ILP video signal processors. Wolfe, et al. hand-scheduled some VSP kernels from an MPEG-2 encoder on a 33-issue VLIW architecture [8]. Speedups ranged from 2.5x-30.3x depending on the kernel and the pipeline configuration. This would at first seem to apply that there are sequential phases in the application that limit overall speedup, but this ignores the potential for inter-procedural parallelism. Global analysis of the applications is required to better understand the available concurrency.

3) Methodology

Machine Architecture

We have assumed that the execution model is a VLIW architecture assisted by a very powerful compiler with some 100% accurate prediction capabilities. An equivalent superscalar architecture with an infinite instruction window would provide identical results.

We use the MIPS-III instruction set in the experiments for several important reasons. The MIPS is a typical RISC architecture that has been widely used in previous ILP. Second, the MIPS machines are equipped with an excellent optimizing compiler; in our experiment, we use the highest degree of optimization provided by **cc**. Third, the trace generation tool pixie is available for the IRIX operating system; it is a standard tool for trace analysis. Finally, public source codes for various video encoders and decoders are widely available on SGI platforms.

Trace Driven Simulation

We use trace driven simulation to evaluate the benchmark programs as shown in figure 3.1. Given a test program, we first compile and optimize it with the SGI C compiler and generate object code, for example **prog**. We then annotate it with pixie to add profiling and trace generation code to the binary program and generate the executable annotated code **prog.pixie**. Running the annotated binary file generates an execution trace. Additionally, we use the SGI disassembly tool **dis** on the binary program **prog** to reconstruct assembly code **prog.asm**. Our simulator **parsrc** uses the trace and the assembly code as input to schedule the trace to find the critical path in the dependency graph (DG) of the trace.

Instead of storing the trace in a file, we run the annotated program concurrently with the simulator. The trace is generated dynamically and redirected into the simulator via a pipe. Since the pipe can only be accessed sequentially, the trace information can only be processed in a one-pass manner. This restricts the scheduler to access the trace stream in order, but it saves considerable storage resources when processing traces on the order of 10^9 operations.

Benchmark Applications

The field of video processing is evolving rapidly, and there are no commonly accepted benchmarks for video applications. However, several video standards are being developed to meet a wide range of video processing requirements including HDTV, video conferencing, and DVD. We have used public-domain prototype implementations of these standards as our benchmark set. Table 3.1 gives a brief description on the benchmark programs.

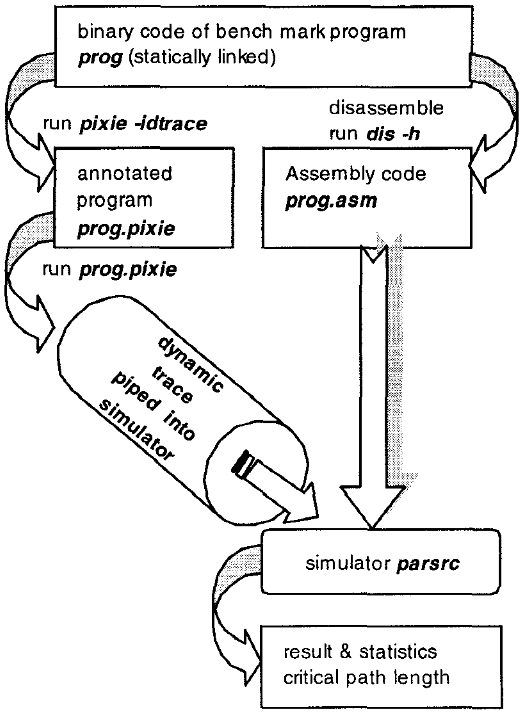
binary code of bench mark program
prog (statically linked)
run *pixie -idtrace*
disassemble
run *dis -h*
annotated program *prog.pixie*
Assembly code *prog.asm*
run *prog.pixie*
dynamic trace piped into simulator
simulator *parsrc*
result & statistics critical path length

Figure 3.1 Flow chart of simulation

MPEG, which stands for Moving Picture Experts Group, is the name of a family of international standards for coding audio-visual information in a digital compressed format. The MPEG family of standards includes MPEG-1, MPEG-2 and upcoming MPEG-4, formally known as ISO/IEC-11172, ISO/IEC-13818 and ISO/IEC-14496. The first standard developed by the group, MPEG-1, was the coding of the combined audio-visual signal at bit rates around 1.5 Mb/s. This was motivated by the prospect of storing video signals on a compact disc with a quality comparable to VHS cassettes. MPEG-2 is an extension to MPEG-1 supporting higher quality pictures at higher bit-rates and multiple-channel audio targeting a wide variety of applications including digital TV. MPEG-4, motivated by

	Application Name	Description
1	MPEG-2 video encoder	Compression standard for broadcast quality video, DVD.
2	MPEG-2 video decoder	
3	MPEG-4 video encoder	A proposed standard for video conferencing and beyond
4	H.263 video encoder	An ITU standard for low bit rate videophone
5	H.263 video decoder	
6	MPEG-1 video encoder	A standard for SIF quality video such as Video CD
7	MPEG-1 video decoder	
8	MPEG-1 audio decoder	Decodes the MPEG-1 audio sequence.
9	Wavelet image encoder	An alternative approach for video & image compression

Table 3.1 Benchmark programs

Resource assumptions	1. Infinite hardware VLIW	The machine is a basic VLIW architecture with infinite number of functional units including many Load/Store units that can access the same memory space concurrently.
	2. Perfect register renaming	There is enough physical register for register renaming thus all register anti-dependency can be resolved.
	3. Perfect memory renaming	There is enough memory space for memory renaming to eliminate memory anti-dependency.
Oracle	4. Perfect disambiguation	All memory address ambiguity can be resolved.
	5. Perfect branch prediction	All conditional branch can be perfectly predicted.
Parametric assumptions	6. One cycle execution	Each functional unit can execute an instruction per cycle; the result can be used by the instruction issued in the next cycle.
	7. MIPS instruction set	The machine executes MIPS R4000 instructions; each instruction takes 2 operands at most and generates one result.
	8. Code transformation	The scheduler can perform code transformations such as constant propagation and tree height reduction as long as correct execution result is guaranteed.

Table 3.2 Oracle Experiment Assumptions

multimedia communication at various bit rates, is still under construction and the application domain is still not clearly defined. ITU H.263 Recommendation specifies a coded representation that can be used for compressing the moving picture component of audio-visual services at low bit rates. H.263 is defined to support videophone application, it is less flexible than MPEG and H.261 standard, but therefore requires much less overhead. Wavelet algorithms process data at different scales or resolutions and provide a new approach to image and video processing. We use a typical Daubechies wavelet basis to study the parallelism in the discrete wavelet transform.

The source codes for MPEG-1, MPEG-2, and H.263 encoders and decoders are available in public domain at http://www.mpeg.org and http://www.nta.no/brukere/DVC/h263_software. The

MPEG-4 source code comes from the European ACTS project MoMuSys. These codes implement test verification models for the proposed standards and are widely used for studying the standards. We use those public source codes in our study.

Oracle Experiment Assumptions

To estimate a empirical upper bound on ILP, Nicolau used the Oracle Experiment assuming an all-knowing oracle is present to guide the scheduling mechanism, telling us which way every conditional jump will go and resolving all ambiguous memory references [3]. He also observed that parallelism found by the oracle is roughly equivalent to the execution of an idealized data flow machine assuming infinite resources, no synchronization cost, and no ambiguity.

We use a similar set of assumptions in our experiment. Table 3.2 lists the assumptions we used in the experimental machine model and scheduling algorithm.

Assumptions 6~8 are parametric assumptions we make to clearly formulate our problem and define the simulation machine model. Note that the resource and oracle assumptions are not realistic. Resource assumptions 1~3 assume the machine has infinite hardware resources including functional units, registers, and memory. Oracle assumptions 4 and 5 assume perfect branch prediction and memory disambiguation. It is impossible to construct such a machine with infinite hardware resource and the control flow, and memory access patterns of some programs are unpredictable. However, in practice the resources that can be incorporated into a processor are growing at a rapid rate and the memory access patterns of most of the kernel video algorithms are statically deterministic. Given these facts, this oracle experiment can isolate some hardware and software bottlenecks that can be effectively addressed in the near future.

Simulator

As shown in Figure 3.2, the simulator consists of three major modules: the parser, the dependency analyzer and the scheduler. The program image, register scoreboard, memory scoreboard, and scheduling record are the primary data structures use by the simulator.

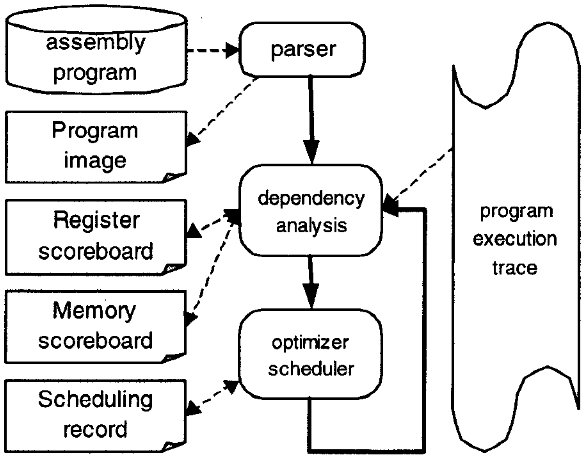

Figure 3.2 Simulator diagram

The parser accepts the assembly program as its input and generates a program image for later use. The dependency analysis module takes one instruction at a time from the program execution stream and finds the instructions it depends on using the scoreboards. The register scoreboard keeps a record of the latest instruction that writes each register. Similarly, the memory scoreboard keeps a record of the latest instruction that writes each memory location. Since the memory space is large, we use a hash table to organize the memory scoreboard. The scheduling record is a large array. Each array element corresponds to an instruction in the trace and contains the scheduled issue time of that instruction. Since the trace might be very long, it consumes a lot of memory space to store the scheduling record. That is why we can only simulate trace up to 10^9 operations.

The simulator is described in *algorithm 3.1.*

1) The parser reads in the assembly program and generates a program image in the memory.
2) Clears the register scoreboard and memory scoreboard.
3) Reads in an instruction address from the trace and look up the instruction in the program image. From the program image and trace, we can get the operand register indexes, the result register index, and the memory address if the instruction accesses memory.
4) Look up the scoreboards according to the indexes or memory address and get the instructions it depends on.
5) The scheduler accepts the dependency information and generates issue time of the instruction and the issue time is stored in the scheduling record.
6) Update the scoreboards.

7) If not end of trace, go to step 3.

Algorithm 3.1 The simulator

```
s=0;
for (i=0;i<10000;i++)
     s=s+i;
```

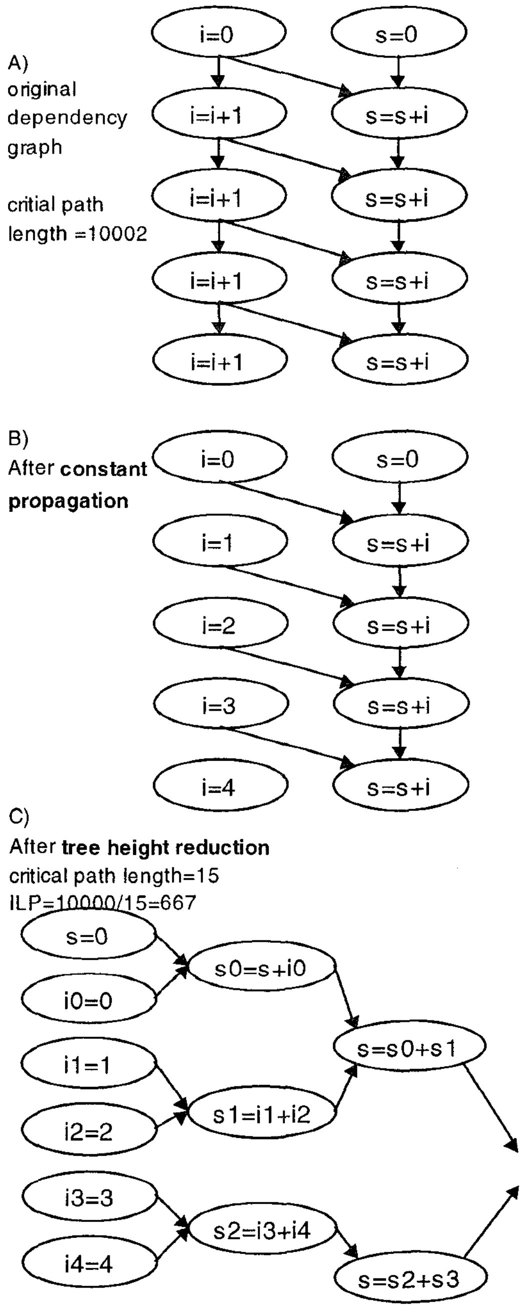

Figure 3.3 Optimization of dependencies.

Optimization Methods

In order to expose more parallelism, we allow code transformation and optimization under assumption 8 in table 3.2. Since high-level program information is not available to the optimizer, we can only use the dependency information generated by the dependency analysis module. We use two simple optimization methods to reduce the critical path length – constant propagation and tree height reduction.

Figure 3.3 gives a simple example to illustrate the optimizing methods. In figure 3.3.A, the original dependency graph is given. There are two long dependency chain: i=i+1 and s=s+i.

Constant propagation is used to reduce dependency chain i=i+1. From a statement sequence like: (a=0; b=a+1; c=b+1), we can deduce that (a=0; b=1; c=2). This kind of transformation is called constant propagation. After the transformation, the dependencies between consecutive statements are eliminated. Figure 3.3.B shows the result of constant propagation. Our optimizer can detect arithmetic operations with immediate operands such as ADDI, SUBI. If the dependent instruction is also a constant assignment instruction or an instruction with constant propagation, constant propagation can also be applied on that instruction.

Normally, the critical path length of accumulating N variables is N. Using tree height reduction, we can sum the numbers pairwise in a tree, then the critical path length can be reduced to $\log_2 N$. In Figure 3.3.C, after performing the tree height reduction the critical path length of the whole program is reduced to 15. A speed up of 667 is found in this example. The scheduler detects the consecutive accumulation instruction pattern and performs tree height reduction on the detected dependency chain. These simple optimization methods can substantially reduce the critical path length in the trace.

4) Results and Analysis

The simulator is implemented on a SGI Power Challenge running the IRIX 6.2 64-bit operating system. An 8GB virtual memory space is used to store the scheduling record thus traces containing up to 10^9 operations can be simulated. We require a 64-bit operating system instead of a 32-bit system since 32-bit addresses can only access 4GB of virtual memory space, therefore constraining the size of the traces.

The complexity of the simulator algorithm is linear; thus very large traces can be analyzed within reasonable amount of time. To be exact, analysis of each example takes less than one day to finish. Dynamic generation of trace streams and one pass analysis allows the huge traces to be manipulated quickly without large disk files. The simulator generates an optimal scheduling under the constraints shown in table 3.2 and outputs the critical path length of the trace.

To compare the results of different scheduling options, we simulate the benchmark programs in three different ways:

1) Without branch prediction, scheduling takes place within basic block boundaries.
2) With perfect branch prediction, but without post-compiler code optimizations.
3) With perfect branch prediction and with *constant propagation* and *tree height reduction* optimizing techniques.

Table 4.1 shows the simulation result under assumption of no branch prediction. The results are not sensitive to application and total trace length. Almost all of the benchmark programs show a speedup of around 3. This result again confirms the conclusion drawn from previous studies that the ILP is constrained by the basic block size without effective global scheduling or precise branch prediction. The control dependencies have a severe impact on the instruction level parallelism.

Table 4.2 shows the result of simulation with perfect branch prediction, but no code transformation technique is applied. With perfect branch prediction assumption, the whole trace can be scheduled as a big basic block. One or two degrees of magnitude of performance enhancement are achieved.

The result of simulation with perfect branch prediction plus code optimization is shown in table 4.3. The code optimization can reduce some dependency chains in the critical path and achieve some degree of performance enhancement. We should note the optimization techniques applied are

No	Name	Trace length	Critical path length	Speedup
1	Mpeg2e	541M	261080901	2.17
2	Mpeg2d	7.73M	2200409	3.68
3	Mpeg4e	128M	43594859	3.09
4	H263e	1224M	474291179	2.71
5	H263d	150M	48346008	3.25
6	Mpeg1e	827M	269918859	3.21
7	Mpeg1d	19.2M	7328627	2.75
8	Mpeg1au	6.7M	3017450	2.33
9	Wavelet	376M	132602451	2.97

Table 4.1 Results (without branch prediction)

No	Name	Trace length	Critical path length	Speedup
1	Mpeg2e	541M	4715781	120.2
2	Mpeg2d	7.73M	254163	31.9
3	Mpeg4e	128M	2562339	52.6
4	H263e	1224M	2610554	491.7
5	H263d	150M	6418141	24.5
6	Mpeg1e	827M	2183754	397.2
7	Mpeg1d	19.2M	417189	48.4
8	Mpeg1au	6.7M	391557	18.0
9	Wavelet	376M	6173882	63.9

Table 4.2 Results (with perfect branch prediction, no optimization)

No	Name	Trace length	Critical path length	Speedup
1	Mpeg2e	541M	4715779	120.2
2	Mpeg2d	7.73M	125813	64.4
3	Mpeg4e	128M	2829286	73.6
4	H263e	1224M	1267538	1012.7
5	H263d	150M	621378	252.5
6	Mpeg1e	827M	1842530	470.6
7	Mpeg1d	19.2M	242443	83.2
8	Mpeg1au	6.7M	214124	32.8
9	Wavelet	376M	2574863	153.1

Table 4.3 Results (with perfect branch prediction and code optimization)

simple methods to reduce the dependency chain of loop control variables. We believe that more advanced optimization technique may be developed to further reduce the critical path therefore achieve higher degree of parallelism.

5) Conclusion

This experiment has found ILP from 32.8 to over 1000 in the tested video applications using an ambitious machine model. The results are based on the unrealistic assumptions. In reality, we have no infinite hardware resource, some instruction can not be executed in one cycle, and it is not possible to have perfect branch prediction and memory disambiguation, however; there are many straight-forward hardware and software techniques that can increase capabilities in these areas. Although removing any of these assumptions will affect the parallelism, it is apparent a lot of parallelism exists in the video applications and can be extracted using known optimizations and scheduling mechanisms. Exploiting this parallelism does, however, depend on scheduling technology that can globally schedule entire applications. Given these results, it is clear that multimedia applications can benefit from ILP architectures that support a far greater number of concurrent operations than would be sensible for other application domains.

Reference

[1] Michael D. Smith, Mike Johnson, and Mark A. Horowitz, "Limits on Multiple Instruction Issue", Third International Symposium on Architectural Support for Programming Languages and operating Systems, pp. 290-302, April 1989.

[2] David W. Wall, "Limits of Instruction-Level Parallelism", Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 176-188, April 1991.

[3] Alexandru Nicolau, Joseph A. Fisher, "Measuring the Parallelism Available for Very Long Instruction Word Architectures", IEEE Transactions on Computers, Vol. C-33, No. 11, pp. 968-976, November 1984.

[4] Intel Corp., "Three Vectors of Performance", also http://www.mmx.com.

[5] S. Dutta, A. Wolfe, W. Wolf, "Design Issue for a Very-Long-Instruction-Word VLSI Video Signal Processor," in VLSI Signal Processing IX, pp. 95-104, October 1996.

[6] Monica S. Lam, Robert P. Wilson, "Limits of Control Flow on Parallelism", Proceedings of the 19th Annual International Symposium on Computer Architecture, pp. 45-57, May 1992.

[7] Norman P. Jouppi and David W. Wall, "Available Instruction-Level Parallelism for Superscalar and Super-pipelined machines," Third International Symposium on Architectural Support for Programming Languages and Operating Systems, pp. 272-282, April 1989.

[8] Andrew Wolfe, J. Fritts, S. Dutta, and E. S. T. Fernandes, "Datapath Design for a VLIW Video Signal Processor", Proceedings of the Third International Symposium on High-Performance Computer Architecture, Feb. 1997, pp24-35.

[9] Gerrit A. Slavenburg, "The Trimedia TM-1 PCI VLIW Mediaprocessor", pp. 171-178, IEEE Hot Chips 8 Symposium on High-Performance Chips, Aug. 1996.

[10] Paul Kalapathy, "Hardware/Software Interactions on Mpact", pp. 20-26, IEEE Micro, Vol. 17, No. 2, March-April 1997.

[11] L. T. Nguyen, M. Mohamed, H. Park, Y. Pai, R. Wong, A. Qureshi, P. Psong, F. Valesco, H. D. Truong, C. Reader, "Multi-media Signal Processor (MSP) Summary" pp. 217-226, IEEE Hot Chips 8 Symposium on High-Performance Chips, Aug. 1996.