A Case for Merging the ILP and DLP Paradigms

Francisca Quintana* Roger Espasa† Mateo Valero

Computer Science Dept.
U. de Las Palmas de Gran Canaria

Computer Architecture Dept.
U. Politècnica de Catalunya-Barcelona

{paqui,roger,mateo}@ac.upc.es
http://www.ac.upc.es/hpc

Abstract

The goal of this paper is to show that instruction level parallelism (ILP) and data-level parallelism (DLP) can be merged in a single architecture to execute vectorizable code at a performance level that can not be achieved using either paradigm on its own. We will show that the combination of the two techniques yields very high performance at a low cost and a low complexity. We will show that this architecture can reach a performance equivalent to a superscalar processor that sustained 10 instructions per cycle. We will see that the machine exploiting both types of parallelism improves upon the ILP-only machine by factors of 1.5–1.8. We also present a study on the scalability of both paradigms and show that, when we increase resources to reach a 16-issue machine, the advantage of the ILP+DLP machine over the ILP-only machine increases up to 2.0–3.45. While the peak achieved IPC for the ILP machine is 4, the ILP+DLP machine exceeds 10 instructions per cycle.

1 Introduction

Historically, there have been two different approaches to high performance computing: *instruction-level parallelism (ILP)* and *data-level parallelism (DLP)*. The ILP paradigm seeks to execute several instructions each cycle by exploring a sequential instruction stream and extracting independent instructions that can be sent to several execution units in parallel. The DLP paradigm, on the other hand, uses vectorization techniques to specify with a single instruction (a *vector* instruction) a large number of operations to be performed on independent data.

The ILP paradigm has been exploited using combinations of several high performance techniques: superscalar out-of-order execution [2, 21], decoupling [17, 12], VLIW execution [20, 6] and multithreading [1, 19, 7]. The current generation of microprocessors all use superscalar execution coupled with a complex memory hierarchy based on several cache levels to attempt executing several instructions per cycle. VLIW processors have long been researched but have not reached the mass market due to their software compatibility problems. Multithreading is a technique being actively researched that might appear in commercial products in a few processor generations.

Measurements of actual performance of applications running on machines exploiting the ILP paradigm [5], show that the actual IPC achieved falls very short of the theoretical peak performance of the machine. Many studies have pointed out that this lack of performance can be due to different effects, such as cache Misses, i-cache misses, branch mispredictions, memory dependences, lack of program parallelism, etc.)

The DLP paradigm has been exploited using vector instruction sets and appears primarily in parallel vector supercomputers [16, 11, 13]. The DLP model has many advantages: a small number of instructions can specify many independent operations, yields simple control units, has efficient instructions to access the memory system and can be easily scaled up to execute many operations per cycle. The main drawback of the DLP paradigm is that it is not as general purpose as the ILP paradigm. It can provide large speedups mostly for highly regular, vectorizable, applications. Interestingly enough, the ILP and DLP paradigms have been always exploited independently.

The goal of this paper is to show that ILP and DLP can be merged in a single architecture to execute regular vectorizable code at a performance level that can not be achieved using either paradigm on its own. We will try to show that the combination of the two techniques yields very high performance at a low cost and a low complexity: the resulting architecture has a relatively simple control unit, tolerates very well memory latency and can be easily partitioned into regular blocks to overcome the wire delay problem of

*This work was supported by the DGUI of Canarian Autonomous Comunity, Spain

†This work was supported by the Ministry of Education of Spain under contract TIC 0429/95 and by the CEPBA.

0-8186-8332-5/98 $10.00 © 1998 IEEE

future VLSI implementations. Also, the control simplicity and the implementation regularity both help in achieving very short cycle times. Moreover, we will show that this architecture can be scaled up very easily, while scaling up an ILP processor is very costly in terms of hardware (and, at some point, may even not be feasible). Even if one scales up a superscalar, we will show that their performance falls behind the performance of the machine exploiting both ILP and DLP.

This paper tries to make the case that, given enough transistor resources, both paradigms should be implemented together in the same chip. Our view of the future is that, in a first step, vector coprocessors will appear closely coupled to a superscalar cpu. When enough real estate becomes available, a vector pipeline will be introduced in most microprocessors. The tasks assigned to this vector pipeline will be the traditional vectorizable floating point applications plus the ever-growing number of computationally and bandwidth intensive media tasks: 3D rendering, MPEG processing, DSP functions, encryption, etc.

2 Strengths of the DLP model

Exploiting DLP has many advantages that can be classified in three areas: Instruction fetch bandwidth, memory system performance (latency and bandwidth), and datapath control. This section will outline the benefits of using a vector instruction set in each of these areas.

Instruction fetch bandwidth. The main difference between a vector and a scalar instruction is that the vector instruction contains a higher semantic content in terms of operations specified. This difference translates into a myriad of related advantages. First, to perform a given task, a vector program executes many fewer instructions than a scalar program, since the scalar program has to specify many more address computations, loop counter increments and branch computations that are typically implicit in vector instructions (section 4 provides quantitative support for this claim). As a direct consequence, the instruction fetch bandwidth required, the pressure on the fetch engine and the negative impact of branches are all three reduced in comparison to an ILP processor. Also, a relatively simple control unit is enough to dispatch a large number of operations in a single go, whereas a superscalar processor devotes an always increasing part of its area to manage out-of-order execution and multiple issue. This simple control, in turn, can potentially yield a faster clocking of the whole datapath.

Memory system performance. Due to the ever increasing gap between memory and cpu speed, current superscalar micros need increasingly large caches to keep up performance. Nonetheless, despite out-of-order execution, non blocking caches and prefetching, superscalar micros do not make an efficient use of their memory hierarchies. The main reason for this inefficient use comes from the inherently predictive model embedded in cache designs. Whenever a line is brought from the next level in the memory hierarchy, it is not known if all data will be needed or not. Moreover, it is very uncommon for superscalar machines to sustain the full bandwidth that their first level caches can potentially deliver. Since load/store instructions are mixed with computation and setup code, dependencies and resource constraints prevent a memory operation to be launched every cycle.

In contrast, in the DLP style of accessing memory every single data item requested by the processor is actually needed. There is no implicit prefetching due to lines. Moreover, the information on the pattern used to access memory is conveyed to the hardware through the stride information and it can be used to improve memory system performance [15].

Memory Latency: When it comes to memory latency, a vector memory instruction can amortize long memory latencies over many different elements. By using some ILP techniques coupled with a DLP engine, up to 100 cycles of main memory latency can be tolerated with a very small performance degradation [8, 10, 9].

Memory Bandwidth: Regarding memory bandwidth, a DLP machine can make a much more effective usage of whatever amount of bandwidth it is provided with. While a superscalar processor requires extra issue slots and decode hardware to exploit more ports to the first level cache, a DLP machine can request several data items with a single memory address. For example, when doing a stride-1 vector memory access, a DLP machine need not send every single address to the memory system. Simply sending every Nth address, a bandwidth of N words per cycle can be achieved.

Datapath Control. In order to scale current superscalar performance up to, say, 20 instructions per cycle, an inordinate amount of effort is needed. The dispatch window and reorder buffers required for such a machine are very complex. The wakeup and select logic grows quadratically with the number of entries, so the larger the window the more difficult is to build such an engine [14]. If current superscalars use 4-wide dispatch logic and barely sustain 1 instruction per cycle, a superscalar machine that sustained 20 operations per cycle seems not feasible.

On the other hand, a vector engine can be easily scaled to higher levels of parallelism by simply replicating the functional units and adding wider paths from the vector registers to the functional units. All this without increasing a single bit the complexity or the pressure on the decode unit. The semantic contents of the vector instructions already include the notion of parallel operations.

3 Methodology

This study will compare the relative merits of the ILP and ILP+DLP models using both trace-driven simulation and data gathered from hardware counters during real executions. We use instruction and memory traces from a Convex C3400 vector machine [4] and from a Mips R10000 microprocessor [21]. Traces on the Convex machine where gathered using the Dixie tool, while the R10000 measurements were obtained using the SimpleScalar toolset [3]. We start by briefly describing our benchmarks, the relevant aspects of both architectures, and then we discuss our performance measures.

Program	#insns S	#insns V	#ops V	% Vect	avg. VL
swm256	6.2	74.5	9534.3	99.9	127
hydro2d	41.5	39.2	3973.8	99.0	101
nasa7	152.4	67.3	3911.9	96.2	58
su2cor	152.6	26.8	3356.8	95.7	125
tomcatv	125.8	7.2	916.8	87.9	127
wave	676.5	41.3	1807.2	72.8	43
mdljdp2	1495.9	89.1	3731.3	71.4	41

Table 1: **Basic operation counts for the Specfp92 programs on the vector machine (Columns 2–4 are in millions).**

Parameters	Latency Scal (int/fp)	Latency Vect (int/fp)
read x-bar	–	2
write x-bar	–	2
add	1/2	1/2
mul	5/2	5/2
logic/shift	1/2	1/2
div	34/9	34/9
sqrt	34/9	34/9

Table 2: **Latency in cycles for the functional units in the architectures under study.**

3.1 Benchmarks

It is very important to make clear that this study focuses on highly vectorizable code. Our goal is to show that for this type of programs, the merge of ILP and DLP techniques leads to very high performance. It is not our claim that a DLP engine will provide speedups for non-regular code (programs such as `gcc` or `li`, from the Spec suite). Therefore, it is reasonable that, for our study, we select those programs that show an acceptable degree of vectorization.

We have chosen as our workload the Specfp92 benchmarks. We compiled all of them on the Convex machine and we selected the 7 programs that achieved at least 70% vectorization.

Table 1 presents some statistics for the selected programs. Columns two and three present the total number of instructions issued by the decode unit, broken down into scalar and vector instructions. Column four presents the number of operations performed by vector instructions. Each vector instruction can perform many operations (up to 128), hence the distinction between vector instructions and vector operations. The fifth column is the percentage of vectorization of each program. We define the percentage of vectorization as the ratio between the number of vector operations and the total number of operations performed by the program (i.e., column four divided by the sum of columns two and four). Finally column six presents the average vector length used by vector instructions, and is the ratio of vector operations and vector instructions (columns four and three, respectively).

3.2 Architectures

In order to have some common ground in which both types of architecture were similar, the first decision was to have similar functional units in both machines. We chose functional units close to the ones present in the R10000 and we use the R10000 latencies in all cases. *Table 2* summarizes all latencies present in the architecture.

ILP architecture

We have taken an approximate model of an R10000 processor as an example of a machine that exploits ILP. We have obtained the measurements for the ILP machine both from R10000 hardware counters during real executions and from execution-driven simulations using the SimpleScalar ToolSet [3]. In particular, we have used the out-of-order simulator which supports out-of-order issue and execution based on the Register Update Unit [18]. This scheme uses a reorder buffer to automatically rename registers and hold the results of pending instructions. Each cycle the reorder buffer retires completed instructions in program order to the architected register file. We have set up the simulator to support the R10000 number and type of funcional units, as well as their latencies (shown in *table 2*). The processor memory system consist of a load/store queue. Loads are dispatched to the memory system when the address of all previous stores are known. Stores are included in the queue if the store is speculative. Loads may be satisfied by either the memory system or a previous store still in the queue if they have the same address. The memory model consist of an L1 data cache and an L1 instruction cache. Both of them are non-blocking and have been configured with the size and replacement policy of the R10000 L1 caches (32 Kb). The main memory latency has been set to 40 cycles. The simulator performs speculative execution. It supports dynamic branch prediction with a branch target buffer with 2-bit saturating counters. The branch misprediction penalty is three cycles.

ILP+DLP architecture

The architecture exploiting both ILP and DLP is derived from a simplified version of the Convex C3400. A C3400 processor has a scalar and vector unit. The vector unit consists of two functional units (one is fully general purpose and the other only performs add-like operations) and one memory access unit. All these functional units are connected to a single vector register file which is organized in banks. It has 4 banks which hold 2 vector registers each. The vector registers hold 128 elements of 64 bits. Each bank has 2 read ports and 1 write port. The machine implements fully flexible chaining except for loads, which can not be chained to a computation. See [4] for further details.

The ILP+DLP architecture (see *fig. 1*) is derived from this baseline machine by adding out-of-order execution and register renaming, in a very similar way as the R10000 [21]. Instructions flow in-order through

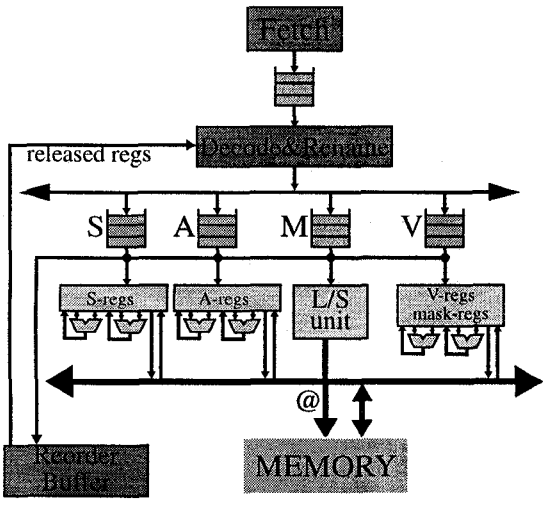

Figure 1: **The ILP+DLP architecture.**

Program	R10000 instrs.	R10000 cycles	R10000 IPC
swm256	11466	20895	0.5
hydro2d	5759	5518	1.0
arc2d	5035	7144	0.7
flo52	1291	970	1.3
nasa7	6897	16830	0.4
su2cor	5135	6348	0.8
tomcatv	1453	2428	0.6
bdna	2501	2501	1.0
avg.	–	–	0.8

Table 3: **Performance of the benchmarks when run on an R10000 processor. Second column is number of executed instructions (in millions). Third column is total execution cycles (in millions) and column IPC is the ratio of columns 2 and 3.**

the Fetch and Decode/Rename stages and then go to one of the four queues present in the architecture based on instruction type. At the rename stage, a mapping table translates each virtual register into physical register. There are 4 independent mapping tables, one for each type of registers: A, S, V and mask registers. When instructions are accepted into the decode stage, a slot in the reorder buffer is also allocated. Instructions enter and exit the reorder buffer in strict program order.

In the ILP+DLP machine each vector register has 1 dedicated read port and 1 dedicated write port. The original banking scheme of the register file can not be kept since it would induce a lot of port conflicts.

Table 2 presents the latencies of the various functional units present in the architecture. The memory system is modeled as follows. There is a single address bus shared by all types of memory transactions (scalar/vector and load/store), and physically separate data busses for sending and receiving data to/from main memory. Vector load instructions (and gather instructions) pay an initial latency and then receive one datum from memory per cycle. Vector store instructions do not result in observed latency because the processor sends the vector to memory and does not wait for the write operation to complete. We use a value of 50 cycles as the default memory latency.

All instruction queues are set at 16 slots. The machine has a 64 entry BTB, where each entry has a 2-bit saturating counter for predicting the outcome of branches. Also, an 8-deep return stack is used to predict call/return sequences. Both scalar register files (A and S) have 64 physical registers each. The mask register file has 8 physical registers.

3.3 The EIPC measure

To be able to compare the performance of the ILP machine and the ILP+DLP machine we define the following indicator of performance:

$$EIPC = \frac{total\ MIPS\ R10000\ instructions}{ILP{+}DLP\ cycles} \quad (1)$$

EIPC stands for "Equivalent IPC" where IPC indicates the number of instructions executed per cycle in the machine. To compute this measure of performance, we run the 7 programs on a MIPS R10000 processor. Using its hardware performance counters, we counted the total number of instructions executed (graduated) for each program. The result is shown in *table 3*. *Table 3* also shows the total number of cycles required to execute each program (in millions) and the resulting IPC (the ratio of columns 2 and 3).

The intuitive sense of the EIPC measure is simple: an EIPC of 10 indicates that a superscalar machine should sustain a performance of 10 instructions executed each cycle to match the performance of the ILP+DLP machine introduced in this paper. Note that "real" IPC's (obtained dividing R10000 instructions by R10000 cycles) are directly comparable to EIPC's. Both measures are giving an idea of the amount of parallelism extracted when executing the same task. Here, a task is a full program and EIPC allows a cycle-time independent comparison between two relatively different architectures.

4 Instruction Level Comparison of the ILP and ILP+DLP models

We start by comparing the ILP and ILP+DLP models looking at the number and types of instructions executed. While number of instructions is not directly a performance measure, it will allow us to show that much of the DLP success is based on its greater semantic contents.

4.1 Number of instructions

In a DLP processor, a single vector instruction can specify many operations (in our case, up to 128). Therefore, in order to specify all the computations required for a certain program, much less instructions are needed.

For example, consider a loop moving 256 words of data from array A to array B. In a ILP machine, a typical loop would consist of about 5 instructions: a load, a store, an addition to increment the address pointer, a subtraction to decrement the loop counter

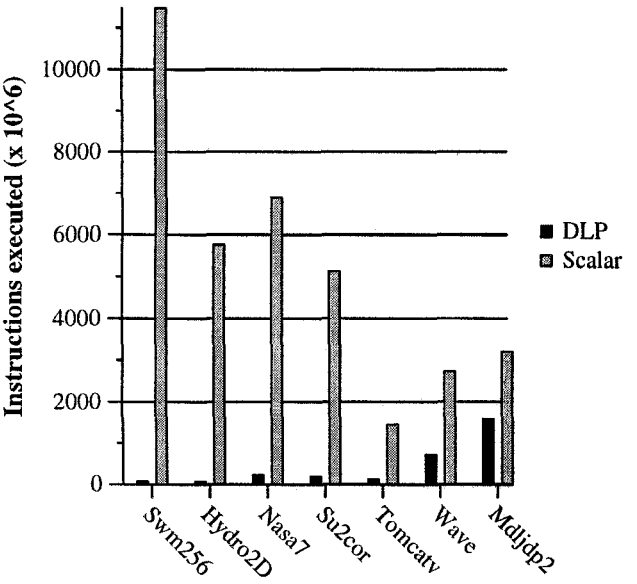

Figure 2: **A comparison of the number of instructions executed on the ILP machine (R10000) and a DLP machine (Convex C34).**

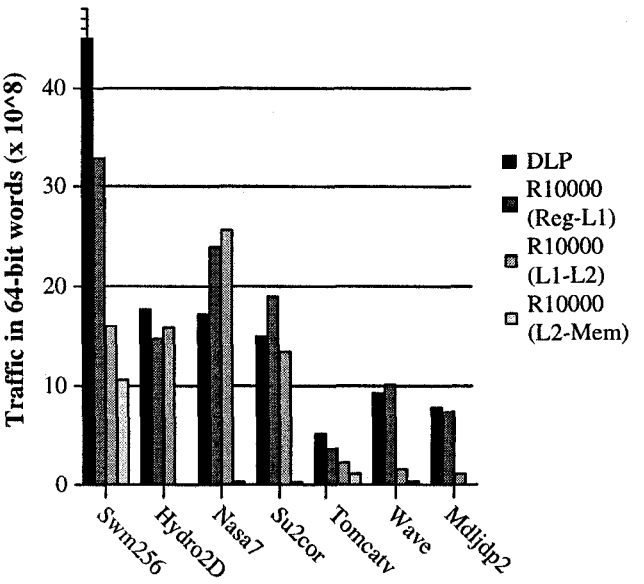

Figure 3: **Comparison of total memory traffic in the DLP and ILP machines.**

and a compare-and-branch instruction. To move 256 words, the loop would execute $256 \times 5 = 1280$ instructions. On the other hand, a DLP machine, would also have the same 5 instructions in the loop. But, the load and store would be vector instructions each responsible of moving 128 elements. Thus, the DLP version of the loop would require just two iterations and, in total, would have executed about 10 instructions to perform the same task.

Although this is a very simple example, it shows the instruction efficiency advantage of exploiting the DLP paradigm. Although severalcompiler optimizations (loop unrolling, for example) can be used to lower the overhead of the add, decrement and branch instructions in the ILP code, vector instructions are inherently more expressive. Having vector instructions allows a loop to do a task in less iterations. This implies less computations for address calculations and loop control. It also directly translates into less pressure in the fetch and decode units and less pressure on the I-cache (fewer instructions per loop).I

Figure 2 presents a comparison of the total number of instructions executed on the ILP machine (R10000) and a DLP machine (Convex C34) for each of our benchmark programs. In the R10000 case, we use the values of graduated instructions gathered using the hardware performance counters. In the C34 case, we use the traces provided by Dixie. As it can be seen, the differences are huge. Obviously, as vectorization degree decreases, this gap is diminished. It is interesting to note that the ratio of number of instructions can be larger than 128. These extra instructions corresponds to the overhead that the scalar machine has to pay due to a larger number of loop iterations.

4.2 Memory Traffic

Figure 3 compares the total memory traffic generated by DLP and ILP machines. Here, we understand memory traffic as the total number of 64-bit words moved up and down through the memory hierarchy. In the DLP case, since there are no caches at all, all data transfers are between registers and main memory. In the ILP case, data transfers can occur at three different levels: registers to L1 cache, L1 to L2 cache and L2 cache to main memory. For the ILP machine, we are presenting the data gathered using the PowerChallenge R10000 hardware counters, which has an L1 of 32 Kb and an L2 of 2Mb. For each program, the first bar plots the total DLP traffic and the following bars plot traffic at the three levels of the memory hierarchy of the ILP machine.

Several things can be pointed out from *figure 3*. First, let's concentrate only in the R10000 memory traffic. A couple of programs (`wave` and `mdljdp2`) mostly fit in the L1 cache. This is deduced from the fact that the traffic between L1 and L2 and between L2 and main memory is very small when compared to the Register to L1 traffic. Programs `su2cor`, `hydro2d` and `nasa7`, fit inside the L2 cache but not inside L1. This can be seen because they move almost the same amount of data (and, sometimes, more) between registers and L1 and between L1 and L2. Moving more data between L1 and L2 than between registers and L1 indicates poor spatial locality and/or cache conflicts. Programs `swm256` and `tomcatv` show a relatively high fraction of L2 to Main memory traffic, although they seem to achieve a good reuse of the data present in L1.

Comparing the DLP bars against the first R10000 bar (register-L1) we see two different behaviors. In 4

programs, `swm256`, `hydro2d`, `tomcatv` and `mdljdp2`, the data movement specified by load/store instructions present in the program is larger in the DLP case than in the R10000 case. This was expected, since the original C34 machine only has 8 vector registers, forcing a lot of spill code that adds to the minimum traffic necessary to carry out the programs computations. It is interesting, though, than in the other 3 programs the DLP machine actually requests less words from memory than the R10000.

Even though the vector memory traffic might be greater in some cases, if we consider the three bars corresponding to the R10000, the picture changes. The height of the L1-L2 and L2-Mem bars gives an idea of the first and second level cache misses. Each cache miss has a certain cost in terms of cycles that can make the importance of these bars very high. Meanwhile, the vector traffic can be evenly distributed across long vector loads, that help compensate memory latency.

5 IPC comparison

We now turn to the performance of the two models under study from an IPC perspective. We will compare an ILP processor which is a close model of the R10000 to the out-of-order dynamic scheduling vector architecture described in section 3.2 (the ILP+DLP machine). We use trace driven simulation for the ILP+DLP machine and execution driven simulation for the superscalar machine. We will compare IPC to EIPC as defined in section 3.3.

We will start comparing a current, state-of-the-art 4-wide issue superscalar to the equivalent ILP+DLP machine. Then we will look into scalability issue to see how the superscalar machine improves when its fetch and decode capability is increased up to 16 instructions per cycle.

5.1 Configurations under study

Table 4 presents the different configurations we will be studying. A configuration is represented by a five-tuple of the form: (*type,issue,memory,RPC,MPC*). "*Type*" indicates whether it is an ILP machine (I) or it's an ILP+DLP machine (ID). "*Issue*" indicates the maximum number of instructions that can be fetched and issued per cycle. "*Memory*" can be either 'R' for a real memory system (40 cycles for the ILP machine and 50 cycles for the ILP+DLP machine) or 'P' for a perfect main memory system that delivers data in 1 cycle. "*RPC*" indicates the number of computed results per cycle for a certain configuration. "*MPC*" stands for memory-per-cycle and is equal to the number of words that can be read or written per cycle.

As it can be seen, the table is split in two main sections. First, configurations having a peak IPC of 4 appear. In the second half, configurations with a peak IPC of 16 are presented. The notation '2x4' in the vector units indicates that we have 2 independent functional units that are 4-way deep. This means that, on one of the functional units, on every cycle 4 independent operations from the same pair of vectors are launched. This is much simpler than actually having four independent functional units: when a single vector `add`, for example, is initiated, it will proceed at four results per cycle until all elements have been processed.

In the case of the memory port, the notation '1x4' indicates that, for stride-1 accesses, data is brought from the memory system in blocks of four. For stride-2 accesses, data arrives in blocks of 2 words and four all other strides (and for scalar references) data arrives one word per cycle. The implementation is such that we save many address pins over a configuration where 4 different ports where available. For stride-1 accesses, our system sends only every fourth address to the memory system, knowing that, in return, four words will be sent.

It is important to note that, in all cases, the ILP+DLP machine is limited to fetching and decoding 4 instructions per cycle and that in the 16-wide configurations we also reduce the total number of results per cycle of the ILP+DLP machine.

5.2 Issue 4

Figure 4 presents the comparison between the first four configurations, all of which have a peak performance of 4 RPC and can transfer at most 1 memory word per cycle. The first thing to note is that, in all but one case, the performance of ILP+DLP is larger than that of ILP by factors that go from 1.5 up to 1.8. While IPC for the ILP machine hardly exceeds 1 in any case, the ILP+DLP machine is for most programs well over 2.3. When comparing the bars with real and perfect memory, we see that, while the ILP machine is very sensible to main memory latency (when increasing latency from 1 to 40 cycles, IPC drops by factors between 1.1 and 1.8, except in `wave`) the ILP+DLP machine experiences almost no difference between a 1 cycle and a 50 cycle main memory latency. Similar results have already been reported in [10].

Note that the ILP+DLP machine is very close to its peak performance. Although the nominal peak performance is 4, if we look back to *table 1* we can see that for the majority of the time at most three operations can be running concurrently: two vector functional units and the memory port. Even though the scalar units could work in parallel with the other 3 units, our analysis show that (a) scalar and vector sections tend to be disjoint and (b) the fraction of scalar code is too small to make a significant difference. Thus, the actual peak is around 3 instructions per cycle. Five programs reach more than 80% of this peak.

The overall conclusion is that the DLP model allows a typical superscalar machine to much better exploit the available parallelism in a program, providing an EIPC that is much closer to the theoretical peak.

5.3 Issue 16

We take our baseline ILP machine and increase its fetch and issue width up to 16 instructions per cycle and provide enough resources to substantially increase IPC. In the ILP+DLP machine, on the other hand, we keep using the same 4-wide issue out-of-order engine, but we provide the machine with more functional units.

Note that there is a big difference in the effort required to add these extra resources to both architectures. In the ILP+DLP machine the extra functional

Name	Fetch/ Issue	Branch Pred.	Reorder Buffer	Functional Units				L2-to-Mem Latency	RPC	MPC
				int	fp	vect	ld/st			
(I,4,R,4,1)	4	BTB-512	32	2	2	–	1	40	4	1
(I,4,P,4,1)	4	BTB-512	32	2	2	–	1	1	4	1
(ID,4,R,4,1)	4	BTB-64	32	1	1	2	1	50	4	1
(ID,4,P,4,1)	4	BTB-64	32	1	1	2	1	1	4	1
(I,16,R,16,4)	16	BTB-2048	128	8	8	–	4	40	16	4
(I,16,P,16,4)	16	BTB-2048	128	8	8	–	4	1	16	4
(ID,4,R,12,4)	4	BTB-64	32	2	2	2x4	1x4	50	12	4
(ID,4,P,12,4)	4	BTB-64	32	2	2	2x4	1x4	1	12	4

Table 4: **Machine configurations under study: widths 4 and 16 for the ILP machine (I) and ILP+DLP machine (ID).**

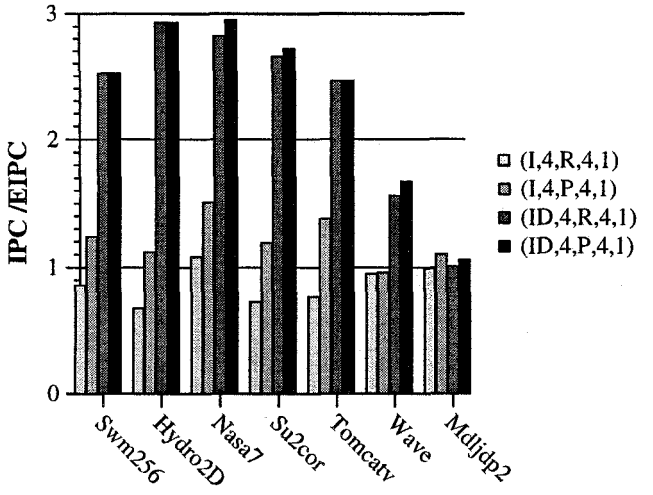

Figure 4: **EIPC comparison with issue 4.**

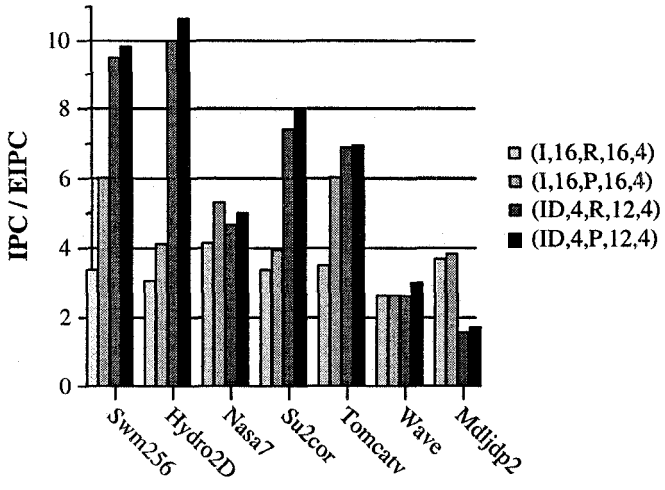

Figure 5: **EIPC comparison with issue 16.**

units are added by partitioning the vector data path and register file in 4 sections. Each section is completely independent of all others and, yet, they work synchronously under control of the same instruction. In each section we have 1/4 the total register file and 2 functional units. Thus, from the point of view of control, the extra resources do not require any special attention. On the other hand, in the ILP machine, increasing the number of functional units has forced us to add a 16-wide fetch engine and to implement a 128-entry reorder buffer. Moreover, the number of ports into the register files has grown enormously either putting in jeopardy the cycle time or introducing some need for duplicate register files. Finally, the L1 cache in the ILP machine has to be 4-ported, while the ILP+DLP machine still retains its simple scheme where only 1 address is sent over the memory port per cycle.

Figure 5 presents the IPC values for all the high end configurations. As we saw in the previous section, if we compare "real" configurations, the ILP+DLP machine outperforms the ILP machine in most cases. In four programs, `swm256`, `hydro2d`, `su2cor` and `tomcatv`, the speedup of the ILP+DLP machine over the ILP machine is in the range 2.0–3.45. For programs `nasa7` and `wave`, speedups are more moderate but still significant: 1.2 and 1.13, resp. Looking at the bars with real memory, we see that the ILP machine is typically below an IPC of 4, while the ILP+DLP machine exceeds an EIPC of 6 in four cases.

If a perfect memory system is considered for the superscalar machine, we can see that performance increases significantly. In one case, `nasa7`, the ILP machine outperforms the ILP+DLP machine with perfect memory, but only by a very small margin (less than 6%). In two cases, `swm256` and `tomcatv`, IPC for the ILP machine reaches 6.

6 Summary

This paper has presented data comparing the performance of an architecture exploiting instruction level parallelism (ILP) and an architecture exploiting both ILP and data level parallelism (DLP) on a set of highly vectorizable codes.

We have seen that, at the instruction level, the DLP paradigm offers substantial savings in terms of instructions executed and pressure on the fetch engine. Our

data shows that a vectorized program can be execute using 128 times less instructions than a purely scalar program. We have looked at the memory behavior of the ILP and DLP machines. Due to the relatively scarce vector registers, in several cases the DLP machine required overall more loads and stores than the ILP machine. Nonetheless, when the cache behavior model is factored in, that is, when we consider the fact that a word request can turn into a N-word line request, we have seen that the DLP machine better manages its memory hierarchy.

In the second part of this paper, we have performed an IPC comparison of the two architectures under study. For roughly equivalent machines able to produce 4 results per cycle and move 1 memory word per cycle, we have seen that the ILP+DLP machine outperformed the ILP machine by factors of 1.5–1.8. If we increase the available hardware resources by increasing issue width from 4 to 16 and by allowing up to 16 results per cycle and 4 memory words transferred per cycle, we see that the ILP+DLP machine can make much better use of the extra resources. The speedup of the ILP+DLP machine over the ILP machine was in the 2.0–3.45 range in most cases. Moreover, while the peak achieved IPC for the ILP machine is 4, the ILP+DLP machine exceeded an EIPC of 10.

We believe that the results for the ILP+DLP machine are good enough to consider worth it adding a vector pipeline to current superscalar microprocessors. The tasks assigned to this vector pipeline would be the traditional vectorizable floating point applications plus the ever-growing number of computationally and bandwidth intensive media tasks: 3D rendering, MPEG processing, DSP functions, encryption, etc. We conjecture that an important consequence of adding the vector pipeline would be that the superscalar core need not to be as aggressive as it is nowadays. The vector part would take care of highly vectorizable programs, while the scalar part would focus on simple (but fast) execution of the other codes. This design strategy would allow a very fast clock, based on the simplicity of the scalar core and on the very regular and easily pipelineable nature of the vector core.

References

[1] A. Agarwal. Performance Tradeoffs in Multithreaded Processors. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):525–539, September 1992.

[2] D. Anderson, F. J. Sparacio, and F. M. Tomasulo. The IBM System/360 model 91: Machine philosophy and instruction handling. *IBM Journal of Research and Development*, 11:8–24, January 1967.

[3] D. Burger, T. Austin, and S. Bennett. Evaluating Future Microprocessors: the SimpleScalar Tool Set. Technical Report CS-TR-96-1308, Computer Sciences Department. University of Wisconsin-Madison., 1996.

[4] Convex Press, Richardson, Texas, U.S.A. *CONVEX Architecture Reference Manual (C Series)*, sixth edition, April 1992.

[5] Z. Cvetanovic and D. Bhandarkar. Performance characterization of the Alpha 21164 microprocessor using TP and SPEC workloads. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture*, pages 270–280, San Jose, California, February 3–7, 1996. IEEE Computer Society TCCA.

[6] K. Ebcioglu, R. Groves, K.-C. Kim, G. Silberman, and I. Ziv. VLIW compilation techniques in a superscalar environment. *SIGPLAN Notices*, 29(6):36–48, June 1994. *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation.*

[7] R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, and S. Liu. Evaluation of multithreaded uniprocessors for commercial application environments. In *ISCA*, pages 203–212. ACM Press, May 1996.

[8] R. Espasa and M. Valero. Decoupled vector architectures. In *HPCA-2*, pages 281–290. IEEE Computer Society Press, Feb 1996.

[9] R. Espasa and M. Valero. Multithreaded vector architectures. In *HPCA-3*, pages 237–249. IEEE Computer Society Press, Feb 1997.

[10] R. Espasa, M. Valero, and J. E. Smith. Out-of-order Vector Architectures. Technical Report UPC-DAC-1996-52, Univ. Politècnica de Catalunya–Barcelona, November 1996.

[11] A. Iwaya and T. Watanabe. The parallel processing feature of the NEC SX-3 supercomputer system. *Intl. Journal of High Speed Computing*, 3(3&4):187–197, 1991.

[12] L. Kurian, P. T. Hulina, and L. D. Coraor. Memory Latency Effects in Decoupled Architectures. *IEEE Transactions on Computers*, 43(10):1129–1139, October 1994.

[13] W. Oed. Cray Y-MP C90: System features and early benchmark results. *Parallel Computing*, 18(8):947–954, August 1992.

[14] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *24rd Annual International Symposium on Computer Architecture*, Denver , Colorado, June 2–4, 1997.

[15] M. Peiron, M. Valero, E. Ayguadé, and T. Lang. Vector multiprocessors with arbitrated memory access. In *22nd Annual International Symposium on Computer Architecture*, pages 243–252, Santa Margherita Ligure, Italy, June 22–24, 1995.

[16] R. M. Russell. The CRAY-1 computer system. *Communications of the ACM*, 21(1):63–72, January 1978.

[17] J. E. Smith, S. Weiss, and N. Y. Pang. A Simulation Study of Decoupled Architecture Computers. *IEEE Transactions on Computers*, C-35(8):692–702, August 1986.

[18] G. S. Sohi. Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers. *IEEE Transactions on Computers*, 39(3):349–359, March 1990.

[19] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *ISCA*, pages 191–202. ACM Press, May 1996.

[20] A. Wolfe and J. Shen. A variable instruction stream extension to the VLIW architecture. In *ASPLOS-IV*, pages 2–14, Santa Clara, CA, April 1991.

[21] K. C. Yager. The Mips R10000 Superscalar Microprocessor. *IEEE Micro*, pages 28–40, April 1996.