International Journal of Computer and Electrical Engineering, Vol. 2, No. 5, October, 2010
1793-8163

Exploiting Statically Identified ILP for Network Processor Applications

Byeong Kil Lee, *Member, IEEE*

***Abstract*—Along with sharply increasing bandwidth requirements, modern network applications and new protocols demand highly intelligent and sophisticated processing over the network. Since these workloads require processing capability beyond state-of-the-art microprocessors, parallel architectures are required in order to handle the packet data without slowing down line speed. Network processors with various parallel architectures are appearing in the market, however, a thorough investigation of the implications of static versus dynamic scheduling of this class of emerging workloads has not been done. In this paper, we characterize the performance and power dissipation of statically identified ILP architectures, and we also compare them to dynamically scheduled architectures for network processing. In dynamically scheduled architectures, the power consumption of the instruction window greatly increases with increasing issue widths. On the other hand, statically scheduled architectures show better performance, as well as tremendous advantages in power, since they do not have instruction window wakeup/select, reorder buffer, and other scheduling related hardware modules. With the large parallelism and the loop nature of network applications, our experimental analysis supports static scheduling as an appropriate strategy for network processor applications.**

***Index Terms*—Network processors, ILP, performance evaluation, workload characterization.**

I. Introduction

Modern network applications and protocols demand intelligent and sophisticated processing over the network, requiring non-trivial computation capabilities. The processing requirements within network interfaces and routers are becoming more complex. To keep up with current trends, programmable microprocessors called network processors (NP) are being introduced in network interfaces to handle these demands.

The network processor on the physical port of a router should be able to process the modern workloads without slowing down line speed. Assuming a stream of minimum-sized packets of 64 bytes, a router over 10Gbps (OC-192) link should handle 19.5 million packets per second. In this case, one packet time, a processing time for one packet data, is 51 nano-second. Given a single processor of 1 GHz clock frequency for the router, if we assume every instruction can execute within a cycle, this single processor can execute only 51 instructions per one packet time. The required number of instructions per packet for executing NP applications is in the range of 300~10,000 [4], and hence a conventional processor is not enough to handle these workloads. Highly parallel architectures are required to handle these workloads.

In order to get a high parallelism, recent research and commercial products for parallel implementation of network processors are using multithreaded or vector-type array processors. Melvin et. al. [12] utilize multiple multithreaded processing engines to get a high degree of thread-level parallelism (TLP) in an NP design that supports 256 simultaneous threads in eight processing engines. In this scheme, each thread has its own independent register file, while sharing functional resources and memory ports with other threads. ClearSpeed [13] introduces an MTAP (Multi-Threaded Array Processing) processor, which provides a scalable processing solution, based on an array of 10s to 1,000s of small processing elements. Each PE has its own local memory and I/O capability. Although these implementations can meet the demanded performance, they still have large amounts of hardware complexity, cost and power problems. Complex and special compiler infrastructure is needed to utilize these architectures.

NP workloads have large instruction-level parallelism (ILP) and data-level parallelism (DLP), since they have many loops in the algorithm [4]. However, the existing research and products have been mostly focused on the DLP and TLP. With the being changed trends, the ILP concept also should be considered in designing a network processor in order to get the required throughput. Generally, the architectural concept of ILP implementation extends to information embedded in the program pertaining to the available parallelism between the instructions [14][15]. The two most important types of ILP processors are Superscalar and VLIW.

Current popular media processors - TI's C6x and TriMedia's TM-1300 - rely on the simpler hardware of VLIW processors in order to minimize the cost and power of ILP implementation [3]. This is because multimedia and DSP applications include many loop operations in the algorithm and they are well suited for the static scheduling architecture. Network processor workloads are also loop-intensive, so we consider statically identified ILP implementation as an appropriate architecture for NP applications. As the demand of programmability of network processors is increased, compiler support for network processors also would take on a significant role for performance evaluation.

In this paper, we characterize the performance (throughout) and power dissipation of statically identified ILP implementation, and we also compare them to the dynamic

Manuscript received March 2, 2010.

B. Lee, is with the Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, Texas 78249 USA. Phone: +1 210 4585027; Fax: +1 210 4585947; e-mail: byeong.lee@utsa.edu.

optimization for network processor applications. In order to get a high throughput, more aggressive parallelism and multiprocessor architecture is greatly needed in designing a network processor, but the analytical characteristics of a single processor, with respect to the performance and power dissipation, are also necessary. It can be a solution to finding the appropriate architecture for a specific application workload.

An important question is whether network processor architectures should be statically scheduled or dynamically scheduled. A thorough investigation of the implications of static versus dynamic scheduling for this class of emerging workloads has not been done. The goals of our experiments are to find:

- Do NP workloads benefit from dynamically scheduled processors?
- Can static scheduling gain better performance than dynamic scheduling?
- Can naive (static) scheduling be sufficient? How important are sophisticated compiler techniques?
- What kind of power saving can be obtained from static scheduling (and dynamic scheduling)?

In our experiments, we do not advocate any particular statically scheduled architecture at this point. We are simply using the VLIW paradigm as a vehicle to investigate the feasibility of static scheduling for NP applications.

Modern NP applications can be functionally categorized into two types of operations: the data plane operations and the control plane operations [4]. While the data plane performs packet operations, the control plane handles flow management, signaling, congestion control and higher-level protocols. Although NPs have initially been targeted for data plane applications, they also play a major role in the control plane, particularly for emerging workloads. In fact, with the increased demand for complex processing, the boundaries between data plane and control plane have become blurred [1]. Along with this tendency, the recently released network processor benchmark, NpBench, includes control plane applications [4].

For our experiments, eight control plane and data plane applications, as well as three media applications, are used to evaluate the performance in these experiments. Experimenting with both static and dynamic approaches, we can see that the static scheduling architecture shows better performance (throughput) than the dynamic scheduling architecture in the NP domain. If we can choose proper compiler optimization options (e.g., different block formation) for each application in VLIW, the performance becomes 1.4x ~ 4.1x better than comparable superscalar architecture. In regards to the energy consumption, the static scheduling architecture shows 6.5x ~ 12x more energy efficiency than the compatible superscalar architecture with respect to the power per instruction.

With the characteristics of large parallelism [4] and loop intensive nature, our experimental analysis supports static scheduling as an appropriate paradigm for NP applications. Even though NP applications are quite different from multimedia and DSP applications in architectural aspects, the success of VLIW in multimedia areas could be applied to NP domains with better performance, lower hardware complexity and lower power dissipation.

The rest of the paper is organized as follows: Section 2 provides the network processor workload, including control plane and data plane, used in this paper. Section 3 describes our experimental framework. In section 4, we analyze the characteristics of dynamic scheduling architecture for the network processor. The performance comparison of static and dynamic scheduling architecture is discussed in Section 5. Finally, we present conclusion and future work in Section 6.

II. Network Processor Workloads

It is extremely important to identify appropriate benchmarks for efficient design and evaluation of any processor. In NP fields, three benchmark suites have been previously proposed: CommBench [9], NetBench [5] and NpBench [4]. Wolf et. al. presented eight selected workloads called CommBench [9] for traditional routers and active routers. CommBench has two groups of benchmarks, namely Header Processing Applications (HPA) and Payload Processing Applications (PPA). Memik et. al. proposed nine benchmarks called NetBench [5] for micro-level, IP-level and application-level benchmarks. The NpBench [4], proposed by Lee et. al, categorized NP workloads into control plane and data plane categories. Control plane workloads are just emerging and evolving in current network environments, and they perform congestion control, flow management, higher-level protocols and other control tasks [4]. The other benchmark suites focus on data plane while the NpBench focuses on control-plane. One of significant features in modern network workloads is differentiated service. In fact, large amounts of network bandwidth are consumed by multimedia contents and these multimedia contents are provided with different quality of service levels. In order to capture these newer workload scenarios, control plane workloads and payload processing should be included in NP benchmarks.

Past studies using CommBench [9] and NetBench [5] contrast network workloads with other benchmarks, such as SPEC [6] and mediabench [7], with respect to instruction set characteristics and memory behaviors. NpBench [4] shows the difference characteristics between control plane and data plane network workloads. Based on the three benchmark suites, the characteristics of network processors can be summarized as large number of memory accesses, poor data cache performance, large amounts of branch instruction (particularly in the control plane), and high level of data parallelism.

We choose eight representative NP applications from the above three benchmarks for our experiments, including control plane workloads and payload applications. We also include three media applications in order to compare the effectiveness of static scheduling. Table 1 summarizes selected applications.

TABLE 1. SELECTED WORKLOADS: EIGHT NP APPLICATIONS AND THREE MEDIA APPLICATIONS

Applications	Description
DRR	Deficit round robin scheduling
FRAG	Packet fragmentation application
REED	Reed-Solomon error correction scheme
WFQ	Weighted fair queuing
RED	Random early detection algorithm
SSLD	SSL(Secure sockets layer) dispatcher
MPLS	Multi-protocol layer switching
MTC	Media transcoding
MM	Matrix multiplication
ADPCM	Adaptive Differential pulse code modulation
FFT	Fast fourier transform

Contrary to other applications, network processor applications have two different types of data: packet header data and payload data. While some NP applications need to process only packet header information, others have to deal with payload data as well. Network processor applications handle the packet header data and payload data with the same kernel, which means they have many loops in the algorithm, resulting in high parallelism.

Generally, the parallelism and throughputs are restricted by branch operations and memory operations. These two operations typically consume a significant part of the total execution cycles. Therefore, we investigate the instruction frequency of branch operations and memory operations for the selected NP applications. Figure 1 shows a percentage of the total number of dynamic instructions for memory operations ('load' and 'store') and branch operations. This data is based on the executables compiled for Simplescalar environment. From this experiment, we observe that NP applications use more memory operations and branch operations than media applications. Large number of memory accesses indicates NP applications are a data-intensive application. Also, many comparison and conditional operations are required for several control plane workloads (e.g., QoS (Quality of Service) [11]), because they have to process each packet by its priority. In the next section, we provide a more detailed analysis of the performance factors.

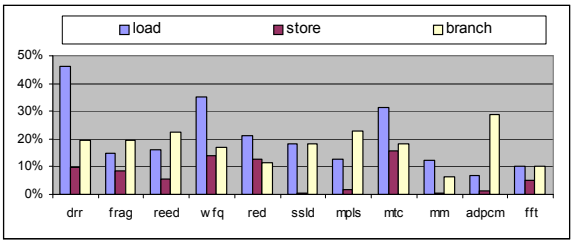


Figure 1. Memory and branch instruction mix in NP applications

III. EXPERIMENTAL FRAMEWORK

In order to study the effectiveness of static and dynamic scheduling for NP applications, we perform experiments on an out-of-order superscalar processor model and a VLIW architecture model. However, we do not advocate any particular statically scheduled architecture. We are simply using the VLIW paradigm as a vehicle to investigate the feasibility of static scheduling for NP applications. Performance and power consumption are used as metrics, hence power simulators are also needed. We utilize four different tools in this evaluation: We use the Simplescalar out-of-order [16], which is used for simulating the dynamic scheduled architecture and analyzing performance, and the Trimaran [17] tool for static scheduled architecture simulation and performance evaluation. We also use the tool Wattch [8] to estimate the power dissipation on superscalar architecture for each application and we use the tool PowerImpact [10] to estimate power consumption of VLIW architecture.

TABLE 2. ARCHITECTURAL CONFIGURATIONS FOR THE EXPERIMENTS OF SUPERSCALAR AND VLIW

Superscalar	Simulated by Simplescalar	
	4-issue	8-issue
Decode width	4	8
Issue width	4	8
Commit width	4	8
Integer ALU	4	8
Integer Multi/Div	4	8
FP ALU	1	1
FP Multi/Div	1	1
L1 I-cache	Size:32K, block size: 64, assoc:1	
L1 D-cache	Size:32K, block size:32, assoc:4	
L2 u-cache	Size:1,024K, block size: 64, assoc:4	
VLIW	Simulated by Trimaran-4121	
Integer	4	
Floating Point	1	
Memory	2	
Branch	1	
L1 I-cache	Size:32K, block size: 64, assoc:1	
L1 D-cache	Size:32K, block size:32, assoc:4	
L2 u-cache	Size:1,024K, block size: 64, assoc:4	

In the static scheduling simulation using Trimaran, we use three region formations for front-end compiler optimization, which is independent of the target processor, in order to see the effectiveness of aggressive compiler optimization techniques. These three region formations are: basicblock, hyperblock and superblock. Basicblock scheduling has a limited scope of exploiting ILP and each basicblock has 4-5 interdependent instructions on average. Hyperblock and superblock are a kind of extended basicblock for scheduling in which groups of basicblocks are scheduled as a single unit.

Wattch is based on the Simplescalar framework and PowerImpact is designed on the Impact tool [22]. Wattch is capable of breaking down the power consumption into the various units of the processor and hence we do detailed analysis of the power consumption of various processor units.

Several processor configurations are simulated on the different tools. For section 4, various superscalar configurations ranging from 4-issue to 64-issue are simulated. Simplescalar configurations for 4 and 8-issue superscalar are explained in Table 2, and the wider superscalar configurations use proportionally larger resources. For the comparison of superscalar and VLIW in section 5, we use 8-issue VLIW architecture, and its Trimaran configuration is shown in Table 2. In order to make compatible superscalar architectures with 8-issue VLIW, we use both 4-issue and

8-issue configurations (Table 2). The 4-issue superscalar architecture has smaller issue width than 8-issue VLIW, but they have compatible functional units. For example, while this VLIW has dedicated functional units for memory and branch operations, 4-issue superscalar handles memory and branch operations, as well as integer ALU operations in 4 integer ALUs. It should be noted that the integer units in VLIW configurations can do multiply and divide. The 8-issue superscalar architecture has same issue width as 8-issue VLIW, but it has more functional units. In our experiment, we assume the architectural configuration of 8-issue VLIW is more compatible with 4-issue superscalar, but it is somewhere in between 4-issue and 8-issue superscalar configurations.

IV. Analysis of Dynamically Scheduled Architectures for NP Applications

A. Effectiveness of Wide Issue

The most significant issue in designing network processors is maintaining the required throughput. Network processor workloads contain a large amount of instruction level parallelism. Hence they possibly will be able to exploit wider and wider issue widths. For each NP application, we applied various issue widths of superscalar architectures in order to see the performance (IPC) variations; the result is shown in Figure 2. We consider a 4-issue superscalar as a base configuration and we double the hardware resources according to each issue width. Most applications show better performance (IPC) with increasing issue widths, but some applications, such as FRAG, REED, SSLD, ADPCM and FFT, show early saturation at a small issue width. In order to get a high throughput, an aggressive increase of issue width can help to improve the performance, but the cost and complexity of the hardware might diminish the benefit.

B. Power consumption of wide issue superscalar

In order to understand the power consumption of wide superscalar processors, we perform experimentation using the Wattch framework. As shown in Figure 3, total energy consumption increases with increasing issue width. In dynamic scheduled architecture, large amounts of power are consumed in instruction window wakeup/select, reorder buffer, and other scheduling related hardware modules. We note that the total energy consumption of a 64-issue architecture is 10 times (on average) larger than that of 4-issue architecture. In particular, the power consumption of the instruction window greatly increases with increasing issue widths.

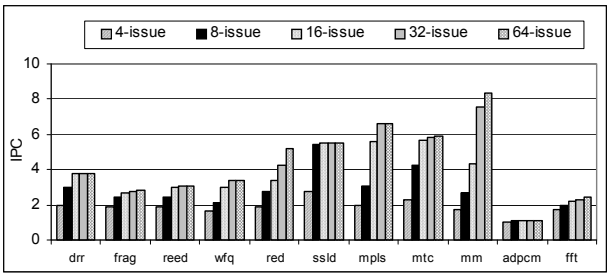
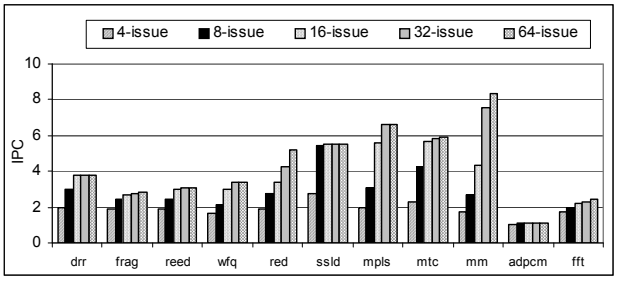


Figure 2. Performance impact of wide issue Superscalar architectures in NP applications



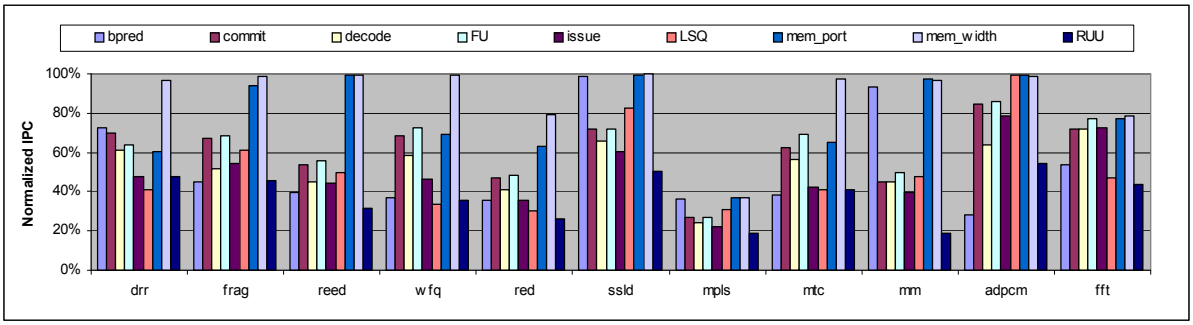
(Y axis: a normalized energy compared to the total energy of 4-issue superscalar architecture)

Figure 3. Energy consumption of wide issue Superscalar architectures for NP applications

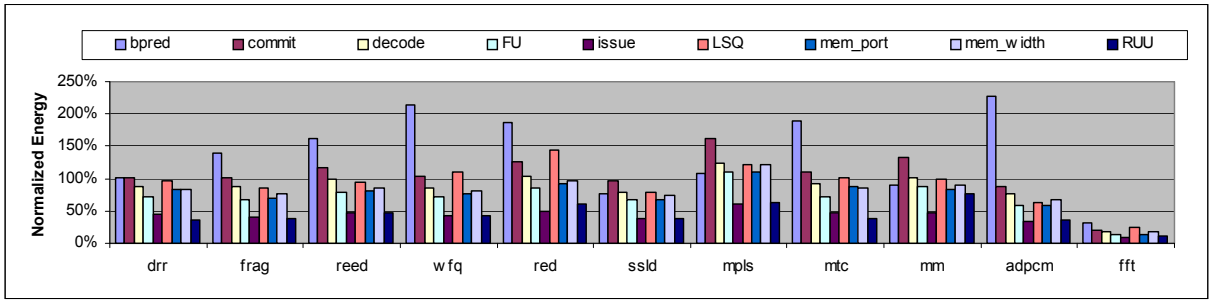
C. Sensitivity Analysis

In order to investigate which hardware element is most influential to the throughput on dynamic scheduled architectures, we perform a sensitivity analysis for NP applications. In this experiment, we use nine restricted hardware elements, including branch prediction, commit width, decode width, the number of functional units, issue width, load/store queue size, the number of memory ports, memory bus width and the register update unit.

Figure 4 presents the results of the sensitivity analysis for NP applications. For each NP application, the impact of restricting the resource is studied. In this analysis, we consider the performance of a 64-issue machine as the baseline performance. In each experiment, a single constraint is intentionally inserted into the baseline performance model. From the results of the experiment, we can determine the degree of impact, which indicates how the constraint affects the overall performance during dynamic execution. The percentage value of each bar represents a normalized performance metric, which is the relative performance compared to the assumed baseline performance (100%).



(Y axis: IPC normalized with respect to 64-issue architecture)
(a) Loss of IPC when resources are restricted



(Y axis: Energy changes compared to baseline architecture)
(b) Changes (increase/decrease) in total energy consumption when resources are restricted
Figure 4. Sensitivity analysis with respect to the resource constraints in NP applications

For this sensitivity analysis, nine constraints, which are independent of each other, are applied. The 'bpred' bar shows the effect of branch misprediction compared to perfect prediction. The 'commit', 'decode' and 'issue' bar show the impact of the limited size of each resource. The 'FU' bar illustrates the impact of restricted functional units. The 'LSQ' and 'RUU' bar show the effects of limited load/store queues and register update units, respectively. The 'mem_port' bar provides the sensitivity of the limited number of memory system ports available to the CPU, and the 'mem_width' bar represents the sensitivity of limited memory access bus width. We use each hardware element of the 4-issue superscalar as the corresponding constraint for the baseline performance model.From this analysis, we see that the restriction of memory width has little impact on the overall performance in most NP applications, except for RED and MPLS. Branch misprediction has largely affected all NP applications, except for SSLD. This observation shows that branches are quite unpredictable in NP applications. As shown in Figure 4 (a), MPLS is the application that was most affected by all of constraints. The common bottlenecks across all NP applications are 'LSQ' and 'RUU', with RED and MPLS having the largest impact. Also, the 'commit', 'decode' and 'issue' width are medium-level bottlenecks in the overall performance for all NP applications.

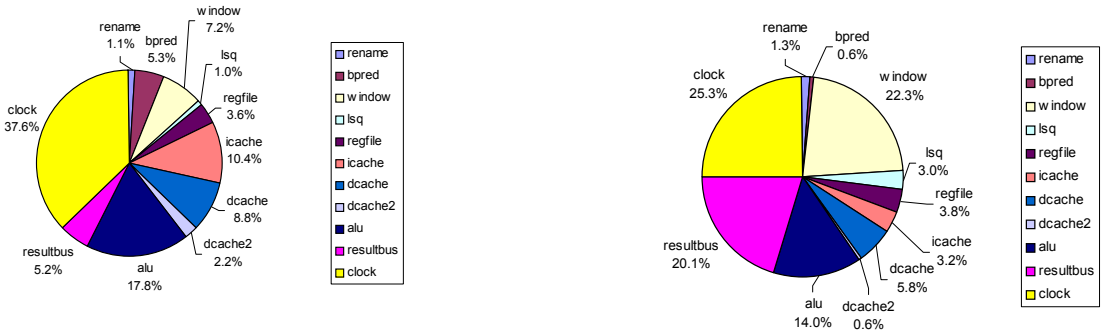
Figure 4 (b) shows the sensitivity analysis of the total energy with respect to the resource constraints. It is interesting to note that branch misprediction leads to a large amount of additional energy dissipation, which is due to the increase in cycles due to the misprediction. The 'commit' has a similar effect as the 'branch' in some applications. We see that better performance (and hence fewer cycles) mean less energy consumption. All other constraints, except for the above two, show the proportional impact of the reduced resources in the power dissipation.

Table 3 shows the impact of inserted constraints on detailed resource elements in one representative application. We choose WFQ for this experiment because WFQ shows a typical characteristic among the selected NP applications. When the 'bpred' is given as a constraint, the power dissipation of all resources (except for 'LSQ') is increased in order to execute additional instructions, which compensate for misprediction penalties. The most affected resource is the register file in the WFQ experiment. The register file consumes twelve times more energy by restricting 'commit' width. This is because there are large amounts of access to the register file, which is due to the narrow commit width. The 'issue' and 'RUU' make the largest impact on the power dissipation across all applications, which implies that large amounts of power is demanded for the related resources (e.g., instruction window) in large issue-width architecture. As shown in Figure 5, we see that the dynamic energy consumption of the instruction window is increased from 7.2% to 22.3% with larger issue architecture among the selected NP applications. We assume that aggressive clock-gating is employed and, therefore, power is scaled linearly with port or unit usage. It is assumed that unused units dissipate 10% of their maximum power.

TABLE 3. IMPACT OF RESOURCE CONSTRAINTS IN ENERGY DISTRIBUTION (WFQ)

WFQ		**Energy Dissipation in each Hardware Elements**											***total***
		rename	**bpred**	**window**	**LSQ**	**regfile**	**icache**	**dcache**	**dcache2**	**alu**	**resultbus**	**clock**	
Inserted Resource Constraints	**bpred**	3.28	7.11	2.63	0.63	2.07	2.36	1.70	2.71	3.01	2.73	2.98	*2.14*
	commit	0.90	1.46	1.00	0.33	12.23	1.48	0.71	1.47	1.43	1.00	1.46	*1.04*
	decode	0.25	1.72	1.00	0.35	0.94	1.76	1.03	1.73	1.68	1.00	1.21	*0.86*
	FU	1.25	1.38	1.01	0.32	1.22	1.07	0.91	1.38	0.38	0.76	0.65	*0.73*
	issue	1.57	2.16	0.14	0.46	0.03	1.57	1.24	2.17	2.13	0.31	1.55	*0.43*
	LSQ	2.18	2.96	1.12	0.38	2.20	1.82	1.59	2.97	2.91	1.32	1.62	*1.09*
	mem_port	1.41	1.44	1.00	0.04	1.86	0.98	0.32	1.50	1.36	1.01	1.21	*0.78*
	mem_width	1.10	1.01	1.00	0.27	0.95	0.83	0.71	1.01	1.01	1.00	1.13	*0.81*
	RUU	0.92	2.83	0.09	0.28	2.32	2.25	1.50	2.84	2.80	0.34	1.47	*0.43*

Above rates are calculated as *energy of restricted resource case divided by energy of maximum performance model*



(a) Power Breakdown of baseline architecture (4-issue) (b) Power Breakdown of 16 issue architecture

Figure 5. Power distribution in dynamic execution of NP applications

V. Comparison of Statically and Dynamically Scheduled Architectures for NP Applications

In this section, we compare the performance of statically scheduled architectures to dynamically scheduled architectures while executing NP applications. We also experiment with media applications in order to investigate the effectiveness of applying static architecture for NP applications. Since the performance of network processors can be measured by an actual throughput, we use the total execution cycles between the two architectures, rather than IPC. In some cases, IPC can be misleading about actual performance criterion because some aggressive compiler optimizations increase the total number of dynamic instructions. However, the increased number of instructions can be executed within smaller cycles, since the optimization techniques introduce much higher parallelism.

A. Performance Evaluation of Static Scheduling in NP Applications

Table 4 shows IPCs (Instruction per Cycle), total number of instructions and total execution cycles, when we applied NP applications and media applications to VLIW and superscalar architectures. Even though the two architectures use different ISAs and several different dynamic and static options, it is meaningful that for NP applications, VLIW architecture shows better performance than compatible superscalar architecture, with respect to the throughput.

In the static scheduling experiments, we see that some applications, such as WFQ and RED, need fewer cycles when executing the application with superblock optimization, while MPLS, SSLD and MTC have better results with hyperblock. Compared to the media applications, some NP applications show a large amount of benefit when we properly select a region formation for the static optimization in VLIW. As shown in Figure 6, the VLIW approach with the selected optimization option for each NP application, shows 1.4x ~ 4.1x (2.2x on average) better performance (throughput) than compatible superscalar (4-issue) architecture. Compared to 8-issue superscalar architecture, this VLIW approach shows better performance across all NP applications except for DRR and MTC. In Figure 6, the Y-axis shows a normalized speedup compared to the total execution cycle of a 4-issue superscalar.

Contrary to media applications, most NP applications show poor performance in 'VLIW-sched (basicblock)' than the superscalar model. Therefore, NP applications need aggressive optimization techniques in compilation. Our experimental results illustrate the effectiveness of aggressive optimization techniques in NP applications in Figure 7. Since hyperblock and superblock optimizations are used to reduce the impact of conditional operations with the penalty of code inflation, our experimental results also prove that NP applications benefit from optimized region formation techniques.

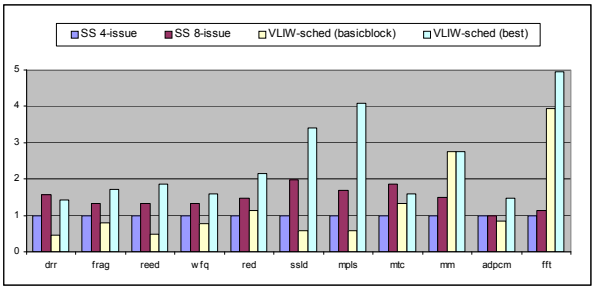
TABLE 4. Performance Evaluation for Selected Applications on VLIW and Superscalar

Benchmarks/Architectures			IPC	Total icount	Total cycles
DRR	Trimaran-4121	basicblock	1.17	299,861,954	255,950,125
		hyperblock	2.81	246,863,079	87,865,261
		superblock	**2.96**	**242,616,913**	**81,910,851**
		unscheduled	0.60	299,898,002	503,331,771
	simplescalar-4-issue		1.93	226419754	117280854
	simplescalar-8-issue		3.02	226142865	74842814
FRAG	Trimaran-4121	basicblock	1.57	46,701,519	29,800,918
		hyperblock	2.71	46,415,391	17,113,605
		superblock	**3.43**	**47,315,391**	**13,813,605**
		unscheduled	0.79	53,901,959	68,003,383
	simplescalar-4-issue		1.85	43999721	23763033
	simplescalar-8-issue		2.48	43933362	17750366
REED	Trimaran-4121	basicblock	1.19	791,093,230	664,971,864
		hyperblock	**4.35**	**764,712,720**	**175,726,369**
		superblock	3.64	1,085,431,602	297,961,729
		unscheduled	0.75	791,134,802	1,054,072,135
	simplescalar-4-issue		1.87	608158542	325170741
	simplescalar-8-issue		2.47	608092879	246202319
WFQ	Trimaran-4121	basicblock	1.08	21,840,219	20,205,926
		hyperblock	2.55	30,549,203	11,965,495
		superblock	**3.23**	**31,366,939**	**9,716,299**
		unscheduled	0.60	21,840,719	36,340,381
	simplescalar-4-issue		1.63	25457216	15596675
	simplescalar-8-issue		2.16	25457216	11783795
RED	Trimaran-4121	basicblock	1.68	1,829,663	1,089,889
		hyperblock	3.07	2,453,847	800,315
		superblock	**3.99**	**2,313,844**	**580,397**
		unscheduled	0.50	1,846,334	3,712,112
	simplescalar-4-issue		1.88	2333156	1242498
	simplescalar-8-issue		2.76	2333156	846699
SSLD	Trimaran-4121	basicblock	1.29	284,158,753	220,379,348
		hyperblock	**4.09**	**153,435,617**	**37,486,792**
		superblock	3.46	262,993,720	76,036,522
		unscheduled	0.69	284,159,291	410,309,274
	simplescalar-4-issue		2.74	349362970	127469305
	simplescalar-8-issue		5.43	349362970	64331782
MPLS	Trimaran-4121	basicblock	1.25	95,635,387	76,368,823
		hyperblock	**4.87**	**52,796,555**	**10,848,065**
		superblock	4.03	88,055,501	21,837,118
		unscheduled	0.77	95,834,327	125,008,745
	simplescalar-4-issue		2.25	99636228	44325791
	simplescalar-8-issue		3.80	99636228	26233316
MTC	Trimaran-4121	basicblock	1.7	423,951,206	249,137,276
		hyperblock	**1.98**	**411,585,030**	**207,468,467**

		superblock	1.99	413,819,818	207,553,368
		unscheduled	0.50	498,816,382	995,714,753
	simplescalar-4-issue		2.27	751971876	331122951
	simplescalar-8-issue		4.23	751971876	177661862
MM	Trimaran-4121	**basicblock**	**2.88**	**4,978,201**	**1,731,232**
		hyperblock	3.24	5,705,563	1,762,825
		superblock	3.23	5,724,763	1,770,025
		unscheduled	0.39	4,914,059	12,548,506
	simplescalar-4-issue		1.76	8424020	4785860
	simplescalar-8-issue		2.65	8424020	3175022
ADPCM	Trimaran-4121	basicblock	1.38	10,259,845	7,425,995
		hyperblock	2.52	11,812,295	4,682,874
		superblock	**2.70**	**11,480,874**	**4,253,279**
		unscheduled	0.84	10,261,049	12,181,785
	simplescalar-4-issue		1.06	6689305	6310430
	simplescalar-8-issue		1.12	6689305	5950291
FFT	Trimaran-4121	basicblock	1.19	32,932,654	27,607,940
		hyperblock	1.61	40,862,761	25,417,119
		superblock	**1.76**	**38,486,065**	**21,901,057**
		unscheduled	0.47	35,292,124	75,664,857
	simplescalar-4-issue		1.75	190076606	108461179
	simplescalar-8-issue		2.01	190076606	94720080

B. Comparison of Different Region Formation Techniques

In VLIW architecture, the compiler plays a major role in finding parallelism, decreasing dependencies among instructions and exploiting other optimization techniques in static mode. For more aggressive optimization, several types of region formations - basicblock, hyperblock and superblock - have been used in the compilation stage. Table 5 shows the static code size of each region formation in VLIW optimization. Hyperblock and superblock optimization has a much larger code size than basicblock optimization, as shown in Table 5, since these optimizations use several algorithms, such as tail duplication, node splitting and loop peeling, to exploit larger parallelism. These algorithms make the code size larger during the optimization process.



(Y axis: a normalized speedup compared to the total execution cycles of a 4-issue superscalar)

Figure 6. Performance comparison of the bench-marks on VLIW and Superscalar architecture

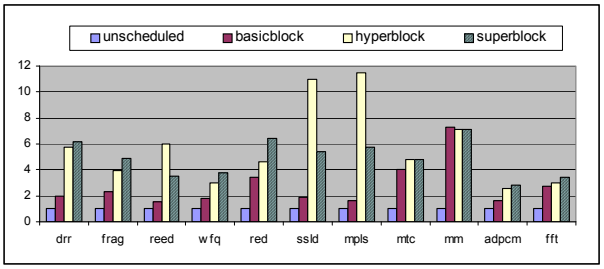
TABLE 5. THE STATIC CODE SIZE OF DIFFERENT REGION FORMATION TECHNIQUES

Benchmarks	Region formation for optimization		
	Basicblock	Hyperblock	Superblock
DRR	204	1,170	1,073
FRAG	89	152	152
REED	148	915	847
WFQ	353	2,258	1,875
RED	358	858	1,332
SSLD	280	1,551	1,778
MPLS	263	926	976
MTC	988	3,160	3,221
MM	166	621	573
ADPCM	172	469	491
FFT	580	2,074	2,268

The most important exploiting of parallelism can be done by employing instruction scheduling which is for assigning instructions into fixed functional units in VLIW architecture. Figure 7 shows the performance comparison between the scheduled and the unscheduled VLIW experiments. For a more intuitive comparison, we use a normalized speedup to total execution cycles of the unscheduled VLIW. From this experiment, we see that the impact of the optimized basicblock (e.g., hyperblock or superblock) in the NP applications is significantly larger than that of the media applications during the instruction scheduling.

C. Power Effectiveness of Static Scheduling in NP Applications

We compare the power consumption of static and dynamic scheduled architectures. As shown in Figure 8, the VLIW approach with the selected optimization option for each NP application, shows 6.5x ~ 12x (9.1x on average) more energy efficiency than the compatible superscalar (4-issue) architecture with respect to the power per instruction (PPI). Even with basicblock optimization of the VLIW architecture, the results show an average of 5.6x more efficiency than the compatible superscalar architecture model in the NP application.



(Y axis: a normalized speedup compared to the total execution cycles of a unscheduled VLIW)

Figure 7. Performance comparison between unscheduled and scheduled VLIW architecture

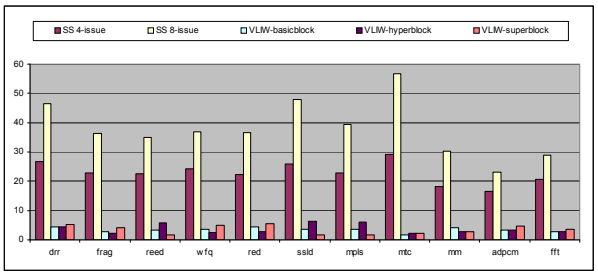


Figure 8. Power per Instruction (PPI) of the benchmarks on VLIW and Superscalar architecture

Since emerging NP workloads require high throughput, parallel architectures are required in order to handle the packet data without slowing down line speed. In order to get a high throughput, the parallel implementation for the network processors always comes with the increase of power dissipation, which cannot be ignored in the clustered processors or multiple processors. However, we should use large issue and clustered architectures to get desired throughput.

VI. Conclusion and Future Work

In this paper, we characterize the performance and power dissipation of statically identified ILP architectures and compare them to dynamically optimized architectures for network processing. In dynamically scheduled architectures, power consumption of the instruction window greatly increases with increasing issue widths. On the other hand, statically scheduled architectures show better performance as well as tremendous advantages in power, since it does not have instruction window wakeup/select logic, reorder buffer, and other scheduling related hardware modules. With the large parallelism and the loop nature of network applications, our experimental analysis supports static scheduling as an appropriate strategy for network processor applications.

NP applications have large parallelism, and they have many loops in the algorithm. Even though NP applications are quite different from multimedia and DSP applications in architectural aspects, the success of VLIW in multimedia areas could be applied to NP domains. While media applications can get enough throughputs with basic block optimization, NP applications need more aggressive optimization techniques in compilation.

A lot of research for VLIW architectures focuses on Clustered VLIW [18], VLIW code compression [19] and Value prediction module [21] in order to get better performance. Using these skills, the performance of NP applications could be improved with smaller code size (by compression scheme) and more aggressive parallelism (by clustered architecture). For the future work, we will attempt to analyze the architectural differences between multimedia applications and NP applications in order to find a clue for improving the performance. Also, we will consider the clustered VLIW architecture for the network processors to get a high parallelism.

References

[1] A. Nemirovsky, “Towards Characterizing Network Proc-essors: Needs and Challenges,” Xstream logic, white paper
[2] J. Williams, “Architectures for Network Processing,” IEEE International Symposium on VLSI Technology, Systems, and Applications, 2001
[3] J. Fritts and W. Wolf, “Evaluation of Static and Dynamic Scheduling for Media Processors,” 2nd Workshop on Media Processors and DSPs (held in conjunction with MICRO-33), Dec. 2000
[4] B. Lee and L. John, “NpBench: A Benchmark Suite for Control Plane and Data Plane Applications for Network Processors,” In Proceedings of the International Conference on Computer Design (ICCD'03), Oct. 2003
[5] G. Memik, W. Mangione-smith and W. Hu, “NetBench: A Benchmarking Suite for Network Processors,” ICCAD 2001.
[6] SPEC, http://www.specbench.org/
[7] C. Lee, M. Potkonjak and W. H. Mangione-Smith, “MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems,” International Symposium on Microarchitecture, 1997
[8] D. Brooks, V. Tiwari and M. Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,” ISCA 2000
[9] T. Wolf and M. Franklin, “CommBench - A Telecommuni-cations Benchmark for Network Processors,” International Symposium on Performance Analysis of Systems and Software, Apr. 2000.
[10] W. Liao and L. He, “Power Modeling and Reduction of VLIW Processors“, in Workshop on Compilers and Operating Systems for Low Power, 2001
[11] X. Xiao and L. M. Ni, “Internet QoS: A Big Picture,” IEEE Network, Mar./Apr. 1999
[12] Melvin, S., Nemirovsky, M., Musoll, E., Huynh, J., Milito, R., Urdaneta, H., and Saraf, K., “A Massively Multi-threaded Packet Processor,” Workshop on Network Processors - NP2, Held in conjunction with HPCA-9, Feb. 2003.
[13] ClearSpeed MTAP processor, http://www.clearspeed.com
[14] Schlansker M., Rau B. R., Mahlke S., Kathail V., Johnson R., Anik S., Abraham S. G., “Achieving High Levels of Instruction-Level Parallelism With Reduced Hardware Complexity,” Technical report, HPL-96-120, Hewlett Packard Laboratories, Feb. 1996.
[15] M. Johnson, Superscalar Microprocessor Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
[16] SimpleScalar LLC, http://www.simplescalar.com
[17] Trimaran toolset, http://www.trimaran.org
[18] A. Terechko, E. L. Thenaff, M. Garg, J. van Eijndhoven, “Inter-Cluster Communication Models for Clustered VLIW Processors,” The Ninth International Symposium on High-Performance Computer Architecture (HPCA'03)
[19] N. Ishiura and M. Yamaguchi, “Instruction Code Compression for Application Specific VLIW Processors Based on Automatic Field Partitioning,” Proceedings of the Workshop on Synthesis and System Integration of Mixed Technologies, 1998
[20] TMS320C62X/C64X/C67X Reference Manual
[21] T. Nakra, R. Gupta, and M.L. Soffa, “Value Prediction in VLIW Machines,” ACM/IEEE 26th International Symposium on Computer Architecture, May 1999.
[22] Pohua P. Chang, Scott A. Mahlke, William Y. Chen, Nancy J. Water, and Wen-mei W. Hwu, “IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors,” Proceedings of the 18th Annual International Symposium on Computer Architecture, May 1991

Byeong Kil Lee received the Ph.D. degree in computer engineering from the University of Texas at Austin, Austin, in 2005. He is currently an assistant professor in the Electrical and Computer Engineering Department, University of Texas at San Antonio, San Antonio, where he joined the faculty in 2009. He was a senior design engineer in Texas Instruments, Inc., Austin from 2004 to 2009. Also, he was a senior research staff in the Agency for Defense Development (ADD), Korea. His current research interests include computer architecture, application-specific embedded systems (network processor, multimedia processor), low power mobile processors, workload characterization of emerging applications, parallel computing and parallel architecture design, power-aware design, and power estimation.