Assertion-Based Power/Performance Analysis of Network Processor Architectures

Jia Yu, Wei Wu, Xi Chen, Harry Hsieh, Jun Yang
University of California, Riverside
{jiayu, wwu, xichen, harry, junyang}@cs.ucr.edu

Felice Balarin
Cadence Berkeley Laboratories
felice@cadence.com

Abstract

Network processors (NPUs) have emerged as successful platforms to provide both high performance and flexibility in building powerful routers. With the scaling of technology and higher requirements on performance and functionality, power dissipation is becoming one of the major design considerations in NPU development. In this paper, we present an assertion-based methodology for system-level power/performance analysis of network processor designs, which can help designers choose the right architecture features and low power techniques. We write power and performance assertions, based on Logic of Constraints. Trace checkers and simulation monitors are automatically generated to analyze the power and performance characteristics of the network processor model. Furthermore, we apply a low power technique, dynamic voltage scaling (DVS), to the network processor model, and explore their pros and cons with the assertion-based analysis technique. We demonstrate that the assertion-based methodology is useful and effective for system level power/performance analysis.

1 Introduction and Motivation

As Internet gets more and more complicated with the rise of new protocols and services, so does the cost of new equipments and equipment upgrades. Network processor (NPU) is a hardware platform that provides high performance and flexible programming capability, which allows it to address many market segments and a wide range of applications. As a result, the upgrade cost can be reduced and developing cycles for new protocols/data types can be shortened. Therefore, NPUs are poised to replace expensive and inflexible fixed-function silicon application-specific integrated circuits (ASICs).

A number of challenges for NPU implementation are already evident, and power dissipation is one of them. For example, in a typical router configuration, there may be one or two NPUs per line card. A group of line cards, e.g. 16 or 32, are generally placed within a single rack or cabinet. Thus, the aggregated heat dissipation becomes a big concern, given that each NPU typically consumes around 20 Watts and the operating temperature can reach as high as 70^oC [4]. On the other hand, with the demand of performance scaling, NPU's clock frequency is increasing and more computation engines will be put on NPUs. Figure 1 shows the power and performance characteristics of three Intel IXP family NPUs [2, 3, 4]. Note that the power dissipation increases as the complexity of an NPU increases. This trend poses a significant challenge for the NPU design.

Description	IXP1200	IXP2400	IXP2800
Performance(MIPS)	1200	4800	23000
Media Bandwidth(Gbps)	1	2.4	10
Frequency of ME(MHz)	232	600	1400
Number of MEs	6	8	16
Power(W)	4.5	10	14

Figure 1. Power and performance characteristics of Intel IXP NPUs

System level modeling with executable languages such as C/C++ or other modeling frameworks have been crucial in designing a large electronic system. One essential approach is to develop a cycle-level accurate simulator. To software developers, the simulator enables application software development and performance optimization long before the product becomes available in silicon. To NPU designers, the simulator facilitates conducting power and performance analysis and fine tuning architectural parameters.

Unfortunately, most cycle-level accurate simulators only report power and performance data for worst and/or average cases, which limits the capability of power and performance analysis. For example, the performance and power dissipation of an NPU are closely related to its workload, namely the incoming packet rate. The workload is usually unbalanced, which may cause extreme high power dissipation at one time versus another. On the other hand, the unbalanced workload provides opportunities for power-performance tuning. Therefore, the power and performance distribution can be an important complement to average/worst case data in the design analysis.

We believe that the assertion-based analysis methodology is very suitable for transaction-level or cycle-level power and performance analysis for NPU designs. From formally specified assertions, trace checkers or simulation monitors are automatically generated to validate and analyze simulation traces. Designers do not need to write separate reference models or

scripts to scan through the traces. So it is very efficient in design analysis for large systems with high complexity and functionality such as NPU designs. In order to analyze the quantitative power and performance characteristics of the NPU design, we use Logic of Constraints (LOC) [6] to specify assertions for the rate, latency, power and energy, and analyze their distributions.

In [8], basic functional and performance properties of an NPU design were verified and analyzed. Our contributions in this paper are in two aspects. The first is to demonstrate power analysis using the assertion-based methodology. This helps designers quickly find right architectures and configurations according to a set of particular power/energy requirements. The second contribution is to extend the use of assertions to analyze the power-performance distributions and trade-offs under various workloads, with low power techniques, such as the dynamic voltage scaling (DVS) [5].

The rest of the paper is organized as follows. In the next section, we introduce the network processor model and its simulator NePSim. In Section 3, we discuss the approach of assertion-based power and performance analysis for the network processor designs. In Section 4, we demonstrate the usefulness and effectiveness of our approach using two analysis case studies, general power analysis and dynamic voltage scaling analysis. Section 5 concludes the paper.

2 Network Processor Model

A network processor design usually contains multiple RISC processing cores, dedicated hardware for common networking operations, high-speed memory interfaces, high-speed I/O interfaces and interfaces to general purpose processors. Here we use NePSim simulator [9] to model our NPU architecture. NePSim is based on Intel IXP1200 and includes a cycle-accurate architecture simulator and a power estimator. All the configurations in NePSim are parameterizable.

The reference model of our network process design, IXP1200, consists of a StrongARM core, six multi-threaded processing units called microengines (MEs), memory controllers, high-speed buses and packet buffers. The basic architecture of the processor is shown in Figure 2. The StrongARM core initializes the program store of the microengines and loads necessary data into memory before enabling the microengines. The off-chip SRAM (up to 8M) is typically used to store the forwarding table, while the SDRAM (up to 256M) is typically used to store IP packets. The usage of each component is highly dependent on the application and workload.

The instruction set of a microengine contains 33 categories of instructions. Memory access takes much longer time compared to other instructions in an NPU design. In each microengine, memory references are issued to a two-entry command FIFO. The commands are then sent to the command bus. Based on the priority of commands, the command bus arbiter selects one or more reference commands among the command FIFOs and move them to the corresponding memory controller. In the NePSim model, to fetch a block of data stored continuously in SRAM requires 7 cycles to fetch the first piece of 32-bit data, and then 2 cycles for each subsequent piece. Because SDRAM bus is shared by the StrongArm core, MEs and PCI, SDRAM access latency is even longer. Due to varying command queuing time, the latency for SDRAM reading/writing varies in the range of 10 to 60 cycles.

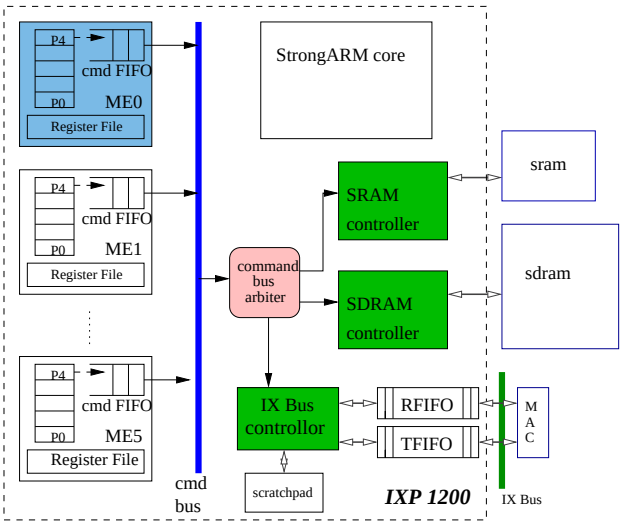

Figure 2. IXP1200 architecture.

3 Assertion-Based Analysis Methodology

Assertion-based trace checking and analysis are similar to the popular embedded assertion technique in hardware design, where simple comparison circuitry is inserted into HDL descriptions to help designers uncover bugs during simulation. The methodology begins with a formalism, Logic of Constraints (LOC) [6] [7], and generates stand-alone checkers, independent of any simulation language and platform. The detailed procedure of the methodology is shown in Figure 3. Furthermore, LOC is designed to specify quantitative performance and functional properties for analysis of transaction-level execution traces. To express the constraints effectively, LOC uses integer index variable to express properties that belong to infinite automata domain. The basic components of LOC are *event names*, *instances of events*, *annotation*, and a single *index variable i*. For example, a latency property (a dequeue event happens no later than 50 cycles after the corresponding enqueue) can be formally specified as an LOC formula: *cycle(deq[i])- cycle(enq[i])<=50*. The formula is satisfied if it holds for all event instances, i.e. for all values of *i*. The automatically generated checkers are used to analyze simulation trace files and report all the violations of the assertions.

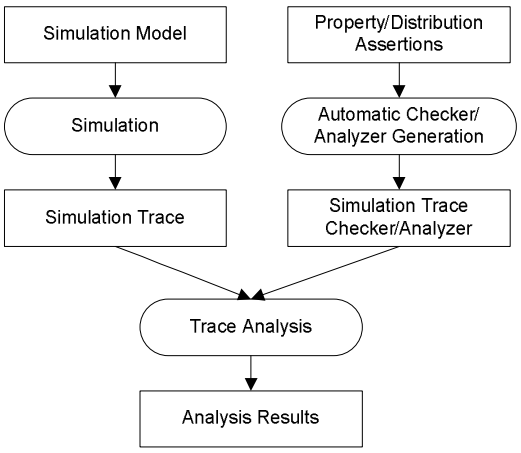

Figure 3. Assertion-based analysis methodology.

To automate quantitative distribution analysis that is common in power and performance analysis, we extend the assertion analysis methodology by introducing 3 simple operators $\bowtie$, $\triangleleft$ and $\triangleright$. To analyze the distribution of some quantity over certain ranges, we can use a formula *quantity* $\bowtie \{min, max, step\}$ to automatically generate a corresponding analyzer. An *analysis period* is represented as a triple $\{min, max, step\}$, where *min* and *max* are lower and upper bounds, and the interval between these two values is divided into bins of width *step*. For example, given the formula

$$(eng(forward[i+100]) - eng(forward[i]))/ \\ (\bowtie \{40, 80, 5\}) \ , \quad (1)$$

the assertion analyzer evaluates the left hand side with *i* being 0, 1, 2, ..., and etc., and report the percentage of *i* whose formula values fall within the ranges of $(-\infty, 40]$, $(40, 45]$, ..., $(75, 80]$, $(80, +\infty)$. If we replace the operator $\bowtie$ with $\triangleleft$ or $\triangleright$, the ranges become $(-\infty, 40]$, $(-\infty, 45]$, ..., $(-\infty, 75]$, $(-\infty, 80]$ or $[40, +\infty)$, $[45, +\infty)$, ..., $[75, +\infty)$, $[80, +\infty)$ respectively.

The distribution analysis still relies on assertion-formulas to generate traces during simulation. It's essentially an extension to do multiple assertion analysis together. It improves the efficiency of the calculation, and saves the execution time. Our methodology is independent on the accuracy of simulator. We choose NePSim because it provides detail power estimator for network processor, and easily changes configurations. If the power model is more accurate, we can always get better results with the same methodology. In the next section, it will be shown that the LOC based quantitative distribution analyzer is a very useful and efficient tool in power and performance analysis for NPU designs.

4 Power/Performance Analysis for NPU

In this section, we present our assertion-based power and performance analysis for NPU architectures through a set of case studies. We first introduce our experimental environment, and then perform simple Min/Max power analysis. We proceed to analyze the power/performance trade-offs with or without dynamic voltage scaling (DVS). We also compare the NPU base model with the power-optimized model that uses less microengines.

4.1 Experimental settings

In our experiments, the cycle-accurate simulator NePSim is used to simulate the network processor models with or without DVS applied and generate traces for power and performance analysis.

4.1.1 Benchmarks

We choose four representative networking applications to explore different architectural features of our NPU model, i.e. *ipfwdr, url, nat* and *md4*. The application *ipfwdr* is an IP forwarding software provided in Intel's SDK. The routing table is stored in the SRAM and the output port information is stored in the SDRAM. The program *url* routes packets based on URL requests. It checks the payload of packets frequently, so it needs a large number of SRAM and SDRAM accesses. In *nat* (network address translation), each packet only needs an access to SRAM for looking up the IP forwarding table. The *md4* provides a 128-bit digital signature algorithm. It moves data packets from SDRAM to SRAM and accesses SRAM multiple times for computation. So it is both memory and computation intensive.

Memory accesses, specially SDRAM accesses, have long latencies. They lead to long idle times for MEs, which in turn causes lower power and throughput. Computation intensive benchmark applications, which have much less memory accesses, will tend to show higher power consumption due to the high power usage of ALUs.

4.1.2 Simulation traces

The simulator provides the assertion analyzer with necessary data traces. The traces contain a set of architectural execution events that occur frequently during simulation and a set of power/performance related values (annotations). We use two types of events in our current power/performance analysis. A *pipeline* event occurs when an instruction enters the ME pipeline. and a *forward* event occurs when an IP packet is forwarded. The events are prefixed to differentiate different MEs or configurations. For example, *m2_pipeline* represents a pipeline event from ME2.

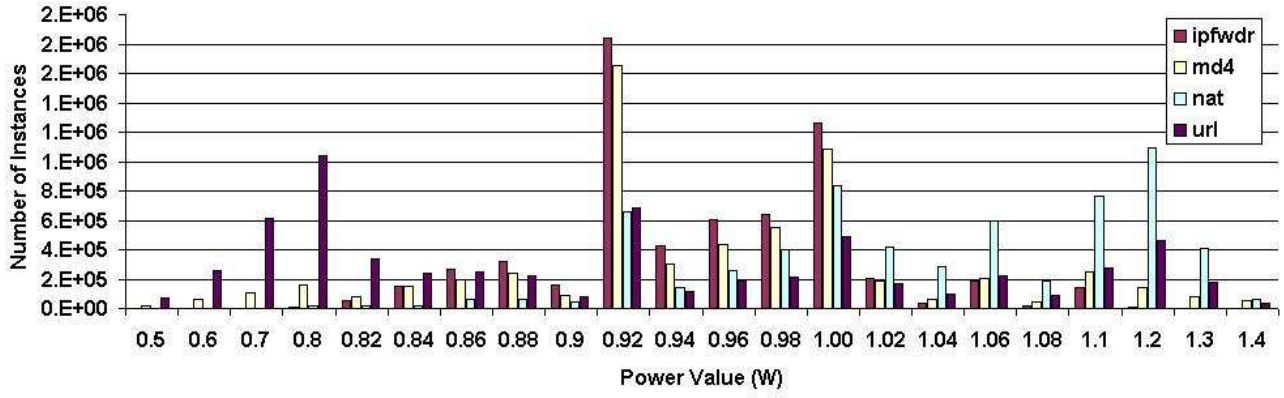

Figure 4. Power distribution graph for 4 benchmarks

Each event is associated with a set of annotations, *cycle*, *time* [1], *eng* and *bit*. We use *cycle* to annotate the number of core clock cycles, *time* to represent the simulated time, *eng* to represent the cumulative energy consumed from the beginning to the occurrence time of the event and *bit* for the total bits of data that has been processed. Figure 5 shows a snapshot of a trace file generated by the NePSim simulator.

cycle	time(us)	eng	bit	event
		.....		
365	1.573	0.773932	5120	m2_pipeline
365	1.573	0.768133	5120	m3_pipeline
368	1.586	0.794108	5632	forward
368	1.586	0.784506	5632	m5_pipeline
369	1.590	0.809369	6144	forward
		.....		

Figure 5. NePSim simulation trace

4.2 Min/Max power analysis

We use our assertion-based analyzer to check the maximum power and power distribution for the NPU model. We simulate our 4 benchmarks, each of which is executed for 8×10^6 cycles with an unlimited packet arriving rate.

Long period of high power consumption can increase the temperature to the extend of damaging the chips themselves. Therefore, we check a property for the maximum power consumption in the six microengines: "the power consumption within every 5 instructions pipelined should be smaller than a threshold value *a*". The property can be specified with an LOC formula:

$$(eng(pipeline[i+5]) - eng(pipeline[i]))/ \\ (time(pipeline[i+5]) - time(pipeline[i])) \leq a \quad (2)$$

The number of 5 is the window size we used to observe the power. The window is sliding, so all instances will be checked. It doesn't change the results if the window size is 10 or 100. The checker executes in less than 1 minute of CPU time. The threshold value *a* in the formula (2) is changed gradually, and we get the maximum and minimum power consumption in 5-pipeline-event time windows (Figure 6). The characteristics of different benchmark result in different min/max power. *nat* has highest maximum and minimum values. This is because it has no SDRAM accesses, so there is no long latency for memory access and the MEs are kept busy running.

	ipfwdr	md4	nat	url
MAX	1.45	1.7	1.7	1.65
MIN	0.6	0.3	0.6	0.3

Figure 6. Power values for 4 benchmarks

Besides checking whether the NPU consumes power within a safe range, we are also interested in how the power values are distributed. We want to know whether it stays close to the average value, or spreads over a wide range. The formula (2) is extended for distribution analysis as follows:

$$(eng(pipeline[i+5]) - eng(pipeline[i]))/ \\ (time(pipeline[i+5]) - time(pipeline[i])) \\ \bowtie \{0.40, 1.40, 0.01\} \quad (3)$$

Figure 4 shows the power distributions for the 4 benchmarks generated from the assertion analyzer [2]. We can see that

[1] The simulated time is important for measurements with DVS enabled, where the NPU clock frequency is changing and we cannot directly get time value from cycle number.

[2] For clearer presentation, infrequent ranges are merged in the graph.

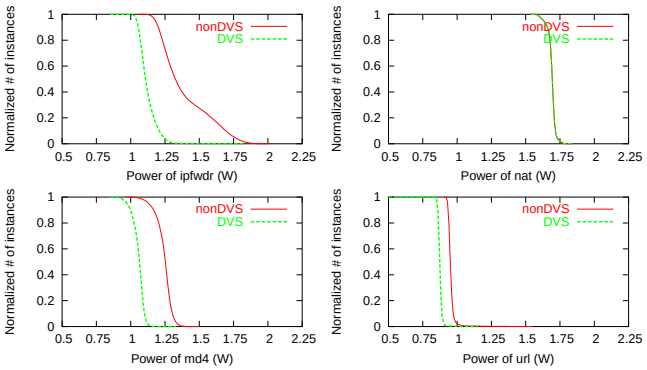

Figure 7. Energy distributions for NPU

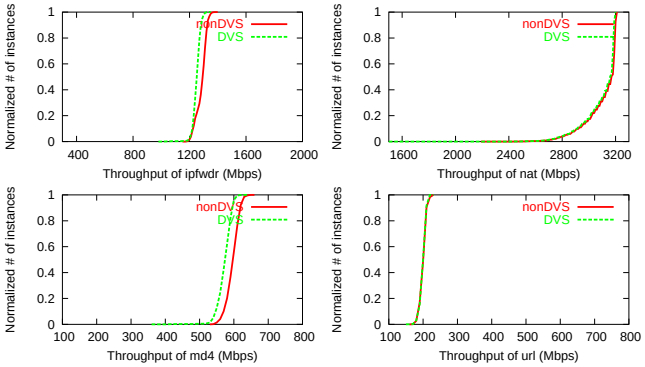

Figure 8. Throughput distributions for NPU

all the benchmarks show a high percentage of power values between 1.00W to 0.90W. The benchmarks *ipfwdr* and *md4* have 28% and 26% of total formula instances (i.e. i's) with power between 0.90W and 0.92W. Another frequent range is between 0.98W and 1.00W, which is caused by some frequently used instruction patterns, e.g. common computation operations. We can also see that the NPU is working around $\pm$10% of the average power for around 70% of the total simulation time. The minimum and maximum power consumptions rarely appear. This is a favorable situation to the chips since they will not become too hot by running in high power for short spurts.

4.3 Power and performance analysis for DVS

Dynamic voltage scaling [5] has been employed widely for microprocessors, resulting in significant lower power and energy. DVS exploits the variance of processor utilization, reducing voltage and frequency (VF in short) when the processor has low activity and increasing VF when the peak performance is required. Dynamic power consumption is proportional to $C \cdot Vdd^2 \cdot \alpha \cdot f$, so reducing voltage ($Vdd$) and frequency ($f$) can significantly reduce power consumption.

Here, we use the ME idle time as a control parameter for DVS. When the idle time is longer or shorter than 10% of each monitored period, we scale down or up the VF by one step, until a lower or upper bound is hit. The bounds of VF, similar to those used in Intel XScale [1], are from 400MHz to 600MHz and 1.1V to 1.3V. We set the frequency step to 50Mhz and compute the voltage as in XScale. In order to match higher NPU frequency, we scaled the speed of SDRAM, SRAM and ixbus to 1.3 times of IXP1200 configuration.

DVS reduces the power, but it may adversely affect the performance. The clock cycle becomes longer if *Vdd* is decreased, so the NPU takes longer time and possibly more energy to get the same work done. This motivates us to check both power consumption and performance of the NPU with or without DVS enabled. For power consumption, we can use the following LOC formula to analyze the traces from both configurations:

$$(eng(forward[i+100]) - eng(forward[i]))/ \\ (time(forward[i+100]) - time(forward[i])) \\ \triangleright \{0.5, 2.25, 0.01\} \ . \quad (4)$$

The left hand side of the formula calculates the average power consumption for each 100 packets forwarded. We can construct a similar formula for DVS-enabled NPU by replacing *forward* in the formula (4) with *dvs_forward*.

DVS is expected to save power, so if the DVS-enabled processor consumes more power, then the DVS policy applied must have made wrong decisions. With the assertion analyzer, we simulate several different configurations for DVS policy and pick the best one.

To compare the performances of the processors with and without DVS enabled, we study the forwarding rates (Mbps) of the two configurations. We calculate the rates during the period of forwarding every 100 packets. The distribution analysis formula is listed as followed.

$$(bit(forward(i+100)) - bit(forward(i))/10^6)/ \\ (time(forward(i+100)) - time(forward(i))) \\ \triangleleft \{100, 3300, 10\} \ , \quad (5)$$

where *bit* is used to annotate the total number of bits that have been forwarded. With DVS applied, we know the performance may degrade. So we need to combine the performance results with the power/energy savings to decide whether DVS really helps.

With our current settings for the DVS policy [9], we run the simulation 8×10^6 cycles for each benchmark with and without DVS enabled. The packets are arriving with uniform intervals of 16×10^{-6} second, which is approximately 5Gbps. The simulation time is around 3 minutes for each, and the assertion checking time takes less than 10 seconds.

The results for power and performance comparisons are shown in Figure 7 and Figure 8 respectively. Each graph

shows the percentage of instances (of checking) that violates a particular maximum power or minimum throughput requirement. In certain range of power or throughput, the corresponding slope of the curve is very large. It means a large number of instances belong to this range. All curves in Figure 7 are very steep. This is consistant with the result we observed in the previous section, that 70% of the instances are around the average power. From Figure 7, we can see that *ipfwdr* shows the most power savings with DVS applied. When we run ipfwdr on the regular NPU, the max power threshold could be as high as 1.8 W (the point where the number of violated instances turns to 0), while in the NPU model with DVS, the max power threshold changes to 1.3 W, which is 72% of the original 1.8W value. This result is due to the fact that *ipfwdr* has abundant time under low VF operations. Similarly, for *md4* and *url*, the power thresholds can be decreased to 89% and 90% by using DVS. The benchmark *nat* shows no change for the power consumption after applying DVS. This is because there is no chance for *nat* to scale down VF. The DVS policy triggers lowering the VF only if enough microengine idle time is detected, but *nat* keeps 6 microengines busy all the time without the need of accessing the long-latency SDRAM. Therefore, no power saving is observed for *nat*.

On the other hand, we observe small performance degradation for most cases from the results shown in Figure 8. There is no performance degradation for *url*. But for *md4* and *ipfwdr*, there are approximately 2% to 3% performance drops. Overall, three of the four benchmarks (except *nat*) show at least 10% of power saving and only 2% to 3% of performance loss. We can see DVS is a very effective low power technique in the NPU design for most applications.

The accuracy of these power and performance measurements mainly relies on the network processor power estimator. However the assertion-based distribution analysis methodology we are utilizing is a general and useful tool for all the simulation models.

5 Conclusions

In this paper, we presented an assertion-based power and performance analysis methodology for network processor architectures. We utilized LOC assertions to specify and analyze the quantitative performance and power properties such as energy, power and throughput. These assertions were efficiently analyzed with automatically generated trace checkers and distribution analyzers on the simulation traces. Specifically, we analyzed the minimum and maximum power distributions for different benchmark applications and the power-performance trade-offs with and without DVS techniques. We showed tha the optimal configuration of an NPU model usually depends on multiple design factors such as the execution characteristics of the application, IP traffic workload and and power or performance requirements. The assertion-based methodology was shown to be an efficient tool to help a designer analyze power and performance characteristics of a design and choose an optimal configuration for it.

References

[1] http://developer.intel.com/design/intelxscale, Intel XScale microarchitecture, 2004.

[2] http://developer.intel.com/design/network/ixa.html, Intel Corporation, IXP1200 Network Processor Family Hardware Reference Manual, 2001.

[3] http://www.intel.com/design/network/products/npfamily/ixp2400.htm, Intel IXP2400 Network Processor, 2004.

[4] http://www.intel.com/design/network/products/npfamily/ixp2800.htm, Intel IXP2800 Network Processor, 2004.

[5] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *International Symposium on Low Power Electronics and Design*, pages 9–14, 2000.

[6] X. Chen, H. Hsieh, F. Balarin, and Y. Watanabe. Automatic trace analysis for logic of constraints. In *Proceedings of the* 40^{th} *Design Automation Conference*, June 2003.

[7] X. Chen, H. Hsieh, F. Balarin, and Y. Watanabe. Verifying LOC based functional and performance constraints. In *Proceedings of International Workshop on High Level Design Validation and Test*, Nov. 2003.

[8] X. Chen, Y. Luo, H. Hsieh, L.Bhuyan, and F. Balarin. Utilizing formal assertions for system design of network processors. In *Proceedings of Design Automation and Test in Europe*, Feb. 2004.

[9] Y. Luo, J. Yang, L. Bhuyan, and L. Zhao. Nepsim: A network processor simulator with power evaluation framework. *IEEE MICRO, special issue on network processors*, Sept. 2004.