Conserving Network Processor Power Consumption By Exploiting Traffic Variability

YAN LUO
University of Massachusetts Lowell
JIA YU
University of California Riverside
JUN YANG
University of Pittsburgh
and
LAXMI N. BHUYAN
University of California Riverside

Network processors (NPs) have emerged as successful platforms for providing both high performance and flexibility in building powerful routers. Typical NPs incorporate multiprocessing and multithreading to achieve maximum parallel processing capabilities. We observed that under low incoming traffic rates, processing elements (PEs) in an NP are idle for most of the time but still consume dynamic power. This paper develops a low-power technique to reduce the activities of PEs in accordance with the varying traffic volume. We propose to monitor the average number of idle threads in a time window, and gate off the clock signals to unnecessary PEs when a subset of PEs is enough to handle the network traffic. We solve the difficulties arising from clock gating the PEs, such as redirecting network packets, determining the thresholds of turning on/off PEs, and avoiding unnecessary packet loss. Our technique brings significant reduction in power consumption of NPs with no packet loss and little impact on overall throughput.

Categories and Subject Descriptors: C.1.4 [**Processor Architectures**]: Network Processor

General Terms: Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Network processor, low power, clock gating, scheduling

ACM Reference Format:

Luo, Y., Yu, J., Yang, J., and Bhuyan, L. N. 2007. Conserving network processor power consumption by exploiting traffic variability. ACM Trans. Architec. Code Optim. 4, 1, Article 4 (March 2007), 26 pages. DOI = 10.1145/1216544.1216547 http://doi.acm.org/10.1145/1216544.1216547.

Authors' address: Yan Luo, Department of Electrical and Computer Engineering, One University Ave, University of Massachusetts Lowell, Lowell, MA 01854; email: yan_luo@uml.edu. Department of Electrical and Computer Engineering, Jia Yu and Laxmi N. Bhuyan, Department of Computer Science and Engineering, University of California Reverside, Reverside California 92507. Jun Yang, Dept. of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania 15260.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 1544-3566/2007/03-ART4 $5.00 DOI 10.1145/1216544.1216547 http://doi.acm.org/10.1145/1216544.1216547

1. INTRODUCTION

In today's capital-constrained environment, network routers now must support continually evolving requirements on aggregating a range of network protocols and traffic types. The need in providing both high performance and flexibility is the key to designing profitable routers. To meet such requirements, network processors (NPs) have emerged as a new class of programmable platform for packet processing. New generation NPs offer high performance through parallel-processing architecture, which incorporates multiple processing elements (PEs) configured as either independent or pipelined units. Being programmable, NPs support new applications with improved time to market and product lifetime, at lower cost.

A number of challenges for NP designs are already evident, and power dissipation is one of them. Typical routers mount a few racks containing groups of line cards (e.g., 8 and 16) each of which contains one or two NPs. Such routers are extremely dense in power dissipation (e.g., 375 W per line card [Cisco]), which causes high operating temperature. At the same time, NP's clock frequency and the number of on-chip PEs keep increasing to meet higher and higher performance requirement. For example, Intel IXP2850 contains 16 microengines operating at 1.6 GHz with 19∼25 W power consumption [Intel 2004], while its predecessor IXP1200 contains 6 microengines operating at 232 MHz with 4.5 W power consumption.

This paper develops a low-power technique by exploiting the varying network traffic load. Routers experience different workload during different time of a day. Figure 1 shows the incoming traffic variation (Mbps) in a router trace collected from the University of Leipzig's central Internet access router during a 24-hour monitoring time [National Laboratory for Applied Network Research]. It can be observed that traffic volume varies in a 24-hour period with low rates at night. The trend shown is common across a large number of Internet packet traces. Significant changes in the traffic bandwidth at the time scale of 12 hours were also observed from the analysis of Internet backbone traffic in Papagiannaki et al. [2003]. This traffic variation implies that much less processing power is required at night as opposed to daytime. In other words, the NP is underutilized at night. This phenomenon brings opportunities as well as challenges for low-power NP design.

To illustrate this, we measured the maximum number of PEs necessary to handle different traffic volumes and the power they consume in IXP1200 using an NP simulator [Luo et al. 2004]. Figure 2 shows the incoming traffic rate and the throughput, both in Mbps, of the NP using three and six PEs with their corresponding power consumption. Two traffic conditions, A and B, are depicted for low and high traffic, respectively. We can see that when the traffic arrival rate is low, using three PEs produces as much processing capability as six PEs, while consuming much less dynamic power. When the arrival rate is high, all the six PEs are necessary to deliver full processing power without packet loss.

In this paper, we propose a low-power technique to save the active power of NPs without sacrificing performance. Our approach is to use the clock-gating technique on PEs when the packet-processing requirement is low and reopen

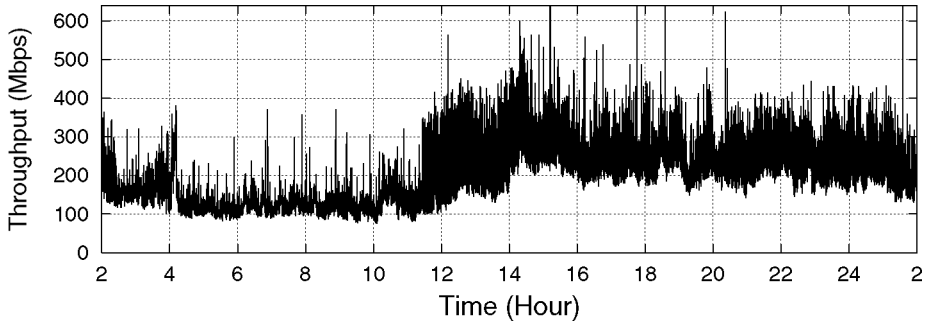

Fig. 1. A sample traffic variation of a central internet access router in 24 hours.

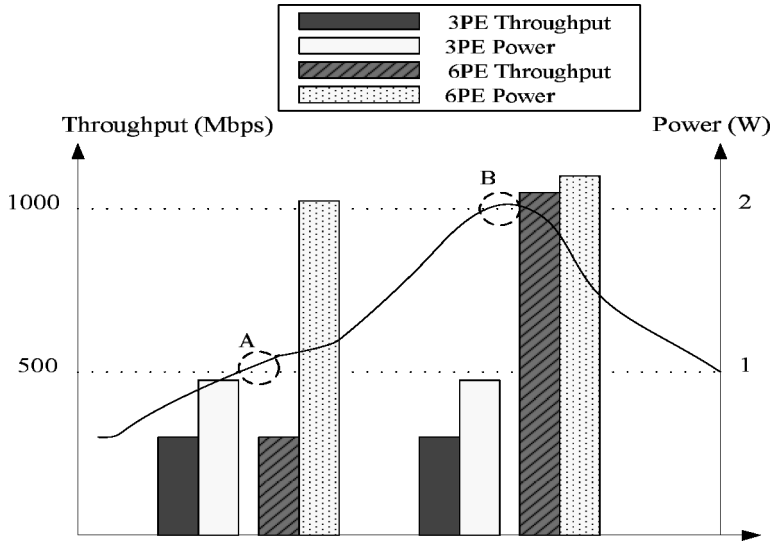

Fig. 2. A comparison between traffic arrival rate and NP processing capability and power consumption.

the clocks when the need is high. The motivation of using clock gating is to effectively "turn off" PEs, but not actually power them down completely, considering the high cost of powering them up. The decision of turning on/off PEs should be made dynamically according to the activity of PEs. A good indication is the number of idle threads that are present in the system. Threads in NP are sequences of code that run in parallel to receive, process, and transmit network packets. If some of them are idle, it means that there is more processing power than required by the incoming packets. Therefore, we propose to use the number of idle threads to determine when to turn off a PE. To determine when to turn on a PE, we observe the pressure arising from the packet incoming buffer. A full buffer indicates low processing capability from NP and packet drops may occur. Our goal here is not to introduce *extra* packet loss due to clock

gating the PEs, but to guarantee enough processing capability with low-power consumption.

We design hardware techniques for detecting idleness of threads, determining thresholds, enabling/disabling the PEs, and rescheduling of packets. We investigate the potential problems at each step and give solutions to overcome them. To accurately measure and test the effectiveness of our technique, we implement our scheme in an NP simulator [Luo et al. 2004]. We add the clock power modeling to the simulator, and also study the proper timing to apply clock gating in NP. We measure the power savings and throughput using real world router traces from National Laboratory for Applied Network Research [National Laboratory for Applied Network Research]. Our experiments show that significant power savings can be achieved when the traffic is nonsaturated.

In addition, the leakage power in nanometer devices increases dramatically because of reduction in threshold voltage (V_{TH}), channel length, and gate oxide thickness [Liao and He 2005]. Clock gating only saves dynamic power; idle MEs still consume a significant amount of leakage power. To tackle this problem, we use leakage power-reduction techniques—power gating—to further save the static power in idle MEs. This technique allows us to preserve the contents in the instruction memory while keeping the leakage low. Our experiments project that the technique can save 22% more power in addition to using clock gating alone.

The rest of the paper is organized as follows. In Section 2 we introduce the network processor model that will be used in our design. Section 3 provides an overview of the dynamic PE turning-off methodology. Section 4 discusses how to solve the problems introduced by turning off PEs. In Section 5, we make it clear why the clock-gating technique is chosen to reduce power and then give details on how we model clock power. We show the results of our clock-gating technique in Section 6. Section 7 will discuss an approach to saving leakage power in low traffic periods. Finally, Section 8 discusses related work. Section 9 concludes this paper.

2. NETWORK PROCESSOR MODEL

A network processor usually contains multiple processing cores, coprocessors, versatile memory interfaces, and high-speed network I/O interfaces, as shown in Fig. 3. The multiple processing cores are programmable key elements for packet processing. Coprocessors, such as hardware accelerators, are incorporated in many NPs to initialize the processor, log events or speed up a particular task such as 3DES. An NP usually interfaces with memory units of difference size and speed. For example, fast SRAM memory is used to hold control data structures and large SDRAM memory used to store packets payloads. An I/O bus unit controls packet receiving and transmitting through the network interfaces. When packets are received from the bus, they are stored in the SDRAM and then processed by the processing elements. During this time, both SDRAM and SRAM are accessed frequently. Once the processing is finished, the packets are then sent out through the I/O bus unit again.

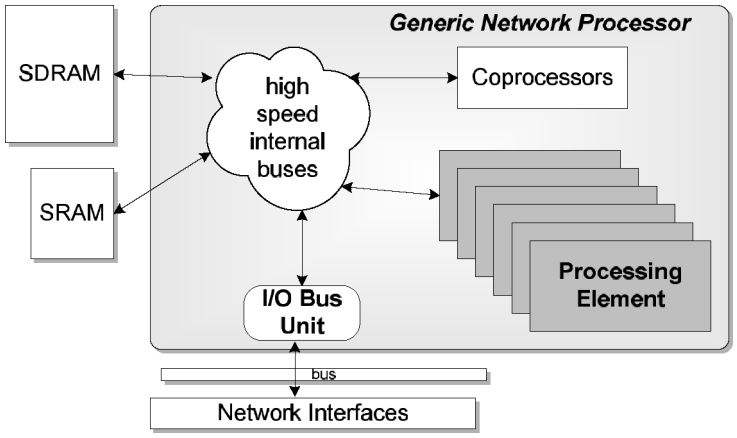

Fig. 3. Diagram of a generic network processor.

To design and evaluate our low-power NP techniques, we utilize our open-source NP simulator NePSim [Luo et al. 2004], because it allows us to modify components, such as easily turning on/off PEs at the architectural level. Though NePSim models IXP1200, an Intel NP that emerged several years ago, it still represents a typical class of NP configuration in which multiple PEs process packets in parallel. As we will explain in Section 4.4, our techniques can be naturally applied to more advanced NPs, such as Intel's IXP2400/2800 [Intel 2004], and AMCC's NP7510 [AMCC 2002], and because they frequently adopt parallel PEs for a single task to boost throughput, analogous to IXP1200.

2.1 Brief Background About Intel IXP1200

The architecture of the Intel IXP1200 resembles a generic NP, as depicted in Fig. 3. There are six processing elements in total, termed microengines or MEs. Each ME is a 32-bit RISC processor with a five-stage pipeline, which supports hardware contexts up to four threads. The pipeline performs coarse-grain multithreading among the threads, i.e., each thread is swapped out of the pipeline only on certain long-latency events, such as memory accesses; otherwise, it holds the pipeline resources for continuous execution. Other components in an ME are registers, ALU, and shifter, instruction memory (termed control store), and miscellaneous buffers for storing control information and commands. There is one SDRAM for holding packets and one SRAM for holding control data.

The IXP1200 supports a group of network ports. Each port can receive network packets at independent rates. When a packet appears on a network port, the IX bus (the I/O bus unit in Fig. 3) sets a bit associated with the port in a status register. Later on, an ME thread will check this bit and fetch the packet for processing. Upon completion, the thread enqueues this packet into the outgoing packet queue and informs another thread to transmit the packet to the destination port. Thus, there are two types of threads in IXP1200: "receiving" threads, which receive packets and process them and "transmit" threads, which simply send packets out. Among six MEs, study have shown that when the number of

receiving MEs versus transmitting MEs is 4:2, the IXP1200 can achieve peak throughput [Spalink et al. 2001].

We listed here only the necessary information that is relevant to our clock-gating technique for low power. More details can be found in the IXP1200 hardware reference manual [Intel 2000]. In the remainder of the paper we will use four benchmarks that have been successfully ported to the NePSim. The details of those benchmarks will be given in the section on experiment evaluation.

3. POLICIES OF DEACTIVATING AND REACTIVATING PROCESSING ELEMENTS

In this section, we discuss the policies of turning off and on PEs and the parameters and their thresholds that we use in the policies.

3.1 Selecting the Parameters

The idea of turning off PEs originates from the observation of the variation in network traffic over time. Such variation is usually specified in terms of packet arrival rate in Mbps. It is natural to use this information as a guide to making decisions. However, it is very difficult to set common thresholds on the arrival rate across different applications because they support different line speeds. Therefore, we do not directly use packet arrival rate as a PE-shutdown parameter.

When the NP is overloaded, incoming packets start to be dropped at the network interface, which indicates that current NP processing power is not enough and more PEs are needed. Thus, packet loss is a good indication of the saturation of an NP. However, although it can serve the purpose of waking up inactive PEs, this parameter implies that a packet has been lost, while one of the design goals of our scheme is to avoid extra packet losses that are introduced because of the reduced number of PEs for power savings.

Alternatively, a PE should be turned off(on) when the workload for the entire NP is low(high) and fewer(more) number of PEs are enough(needed) to handle the workload. Such a status can be indicated by (1) the idle time of a PE during which the PE does no useful work; (2) the length of the thread queue in which a thread waits for incoming packets; and (3) the fullness of an internal packet buffer where packets come in and wait to be processed.

The idle time of a PE is a measure of the PEs being put into the sleep mode, but it is not necessarily an indication of low workload. Many reasons can cause the idleness of the PEs. Two main reasons are long latency memory access and no incoming packets. We certainly cannot turn off PEs when they are waiting for the data from memory. Thus, the idle time of a PE is a mixture of different events, we will not use this parameter in guiding the decision of turning off the PEs.

The thread queue holds threads that are waiting to service the arriving packets. The lower the packet arrival rate, the longer the thread queue. In fact, the number of threads waiting in the queue is the number of excessive threads for the current traffic load. In addition, the length of the thread queue can be easily monitored with little hardware overhead and is not application-specific.

Thus, we will use it as the main parameter to determine when to turn off a PE.

The internal packet buffer is a place to temporarily hold the incoming packets before a thread fetches them for processing. In IXP1200, the RFIFO is a buffer of this kind [Intel 2000]. When the RFIFO starts to saturate, it implies that current PEs are almost inadequate, and more processing power is required. When all PEs are operating, it indicates that the incoming network traffic is too fast to be processed and, thus, packets start to be dropped. We do not address this, since it is the nature of a normal NP even without low-power techniques. When partial PEs are operating, a full RFIFO implies that more PEs should be brought up to clear off the buffer. Otherwise, packets will be dropped. Thus, we use the fullness of the internal buffers as an indicator to activate more PEs. We will explain in Section 4.3 how extra packet loss can be avoided when new packets arrive before an entry in the buffer is freed up.

In summary, the parameters we will use are the length of the thread queue and the fullness of the internal packet buffer. Both are monitored on-chip. The queue length is compared with certain thresholds at fixed time intervals. If a predefined condition is satisfied, the PEs will be turned off or on. Next, we will discuss how to determine the thresholds for the parameters.

3.2 Determining the Thresholds

Our basic logic in determining the thresholds is that if shutting down one PE can equally handle the incoming packets without any loss, then we should turn off one PE. For example, if, during a time interval, the packets are coming at the peak rate, all the PEs should be up. At the next time interval, if the arrival rate drops to below a certain threshold and we justify that with one fewer PE, the NP can still handle the current incoming packets. We then decide to turn off one of the active PEs. However, if during this time interval the fullness of interface buffers is observed, the NP should fall back to have more PEs up and running.

As explained earlier, the length of the thread queue, **l**, indicates the number of free threads that are waiting for incoming packets. If the number of threads each PE supports is **T**, then we could use one fewer PE, if $l > T$, to sustain the network traffic. However, l is a varying number since the packet arrival rate is varying because of the network conditions. We cannot simply use an average value of l. Instead, we monitor the percentage of $l > T$ during a period of observation time to decide the excessive processing power in the NP.

We monitor the thread queue length l for a period of P cycles. During this time, l may exceed T for a certain number of cycles, C. If C accounts for the majority of cycles in P, then we have high confidence of l being greater than T. Hence, we use a threshold **th** ($th < P$) in the unit of cycles that, when C is greater than th, a PE will be clock gated. Note that the value of th reflects how aggressively we turn off PEs. The smaller the th, the more aggressive our scheme is. We initially set th as half of P, i.e., if over one-half of the time there are more number of free threads than one PE contains, then we turn off one PE. This indicates a medium aggressiveness.

The value of *th* can be determined either statically or dynamically. In a *static th* scheme, we let *th* be a fixed value obtained from an off-line study. In a *dynamic th* scheme, we can let the hardware adjust *th*, based on the history information. For example, we can lower *th* by δ periodically until negative impact, such as internal packet buffer full, has been observed. Similarly, we can increment *th* gradually as long as no negative impact occurs. With a dynamic scheme, the *th* will vary with the traffic. If the traffic volume is low for a long period, *th* will drop after several successful shutdowns, achieving more energy savings. If the traffic volume is high, *th* value will rise as a result of the observation of full packet buffer. Therefore, the *dynamic th* scheme is very dependent on the arrival traffic. We will compare the two proposed schemes in Section 6.

4. DEACTIVATING THE PROCESSING ELEMENTS

4.1 Terminating Threads Safely

Before turning off a PE, the hardware should first terminate all its active threads. However, the threads might be working at different stages, either in the middle of processing a packet or just finished processing a packet. For the latter case, it is safe to kill the thread immediately. For the former case, the thread should finish processing the current packet and then terminate. Otherwise, the packet inside the NP occupies spaces in the packet buffer (or memory), but nobody would release it from the buffer, creating "leakage" in resources, which would be drained out eventually.

When a decision is made on turning off a PE, we set an "off" flag in that PE informing it to prepare for shutting down. In IXP instruction set, there is a "kill" instruction that a thread can use to terminate itself. For the threads that are responsible for receiving packets and processing them, they need to check the "off" flag right after finishing processing a packet. If the flag is set, it executes the "kill" instruction and pipeline resources are relinquished. Therefore, implementing this part requires a flag bit per PE and an extra conditional branch and "kill" instruction inserted in the program.

For the threads that are transmitting packets, i.e., their job is to move packets that have been processed by receiving threads to the destination port; it is relatively easier and faster to terminate them. The transmitting threads always check for some register bits that show whether there are packets to be sent out before transmitting the packets. Terminating such threads can be done by clearing off all the bits in the register. Soon the thread will get a cleared register and then execute a "kill" instruction to cease execution.

The number of receiving versus transmitting PEs when they are turned off gradually are: 4+2, 3+2, 2+2, 2+1, and 1+1. The procedure is illustrated in Fig. 4. The number of active receiving PEs (RCV) and transmitting PEs (XMIT) are plotted. In the extreme case, the NP should keep at least one receiving and one transmitting PE to stay awake.

The PEs are multithreaded processors and, at any time, there is only one thread that is executing in the pipeline. Therefore, terminating all the threads in a PE takes a while to complete. To see the duration between when a decision is

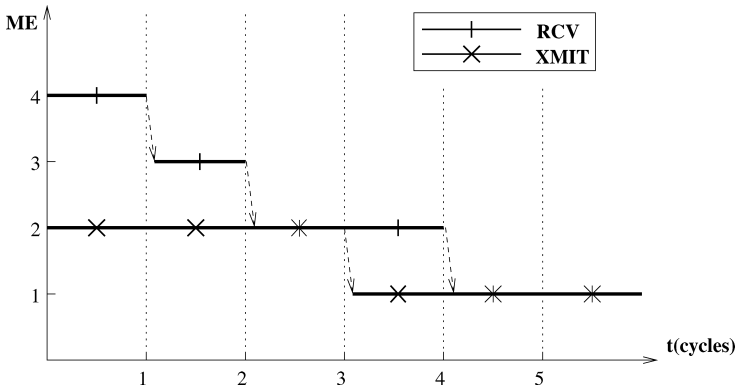

Fig. 4. The procedure of turning off PEs.

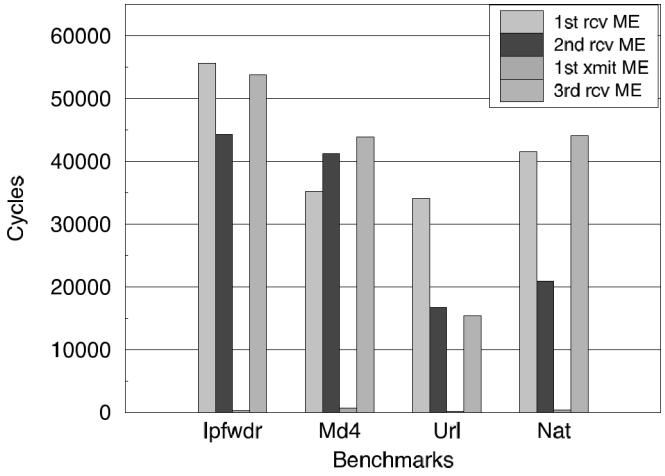

Fig. 5. Time spent to turn off PEs.

made and the when the PE is truly turned off, we measure the time across all the benchmarks in Fig. 5. These durations depend on the time taken for completing the current packets in the PE upon receiving the turn-off signal. Hence, they vary from application to application. This figure plots the number of NP cycles spent on turning off PEs gradually. As we can see, turning off processing PEs take much longer than the transmitting PE. Turning off receiving PEs needs up to tens of thousands of cycles, which amounts to 0.0663∼0.240 ms at a 232 MHz clock rate. This time varies for different applications since the receiving threads are responsible for processing packets and different applications have different processing complexity.

4.2 Reschedule Packets for Orphan Ports

In some NPs, such as IXP1200, the receiving ports are statically allocated to the receiving PEs. Hence, when a receiving PE is turned off, the ports associated

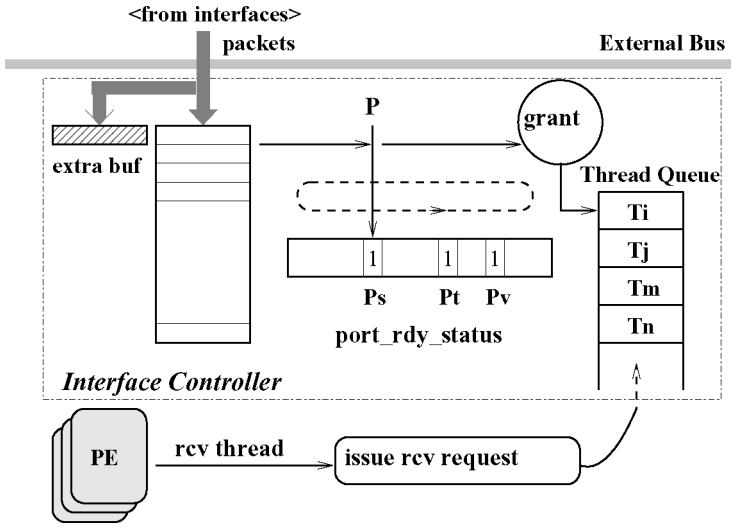

Fig. 6. Implementation of dynamic thread-port mapping.

with them become "orphans," i.e., no threads will pick up the packets from them. As a result, the packets coming from those ports would be dropped. To address this problem, we develop a dynamic mapping scheme, i.e., every thread can take packets from every port as long as there is an incoming packet. Such a dynamic mapping can be found in IXP2400/2800 [Intel 2004] as well, which demonstrates the readiness of applying our proposed technique in up-to-date NPs. The main advantage is to provide flexible scheduling of packets to threads, as explained next.

In static mapping, the IX bus unit continuously examines the ports and sets a port status bit register whenever there is a new packet. A thread keeps polling its own bit in the register until it is set, which indicates that a new packet is ready. After that, the thread starts to receive and process the packet. The main disadvantage of such a static mapping is that if the packets come into different ports at different rates, the ones to the busy ports may be dropped, even if there are threads that are waiting at their idle ports.

To balance the packet arrival rates and the thread-processing capability, it is better to break the one-to-one tie and let free threads pick up packets from any ports. The dynamic mapping is accomplished by adding a very simple hardware in the interface controller. A *thread queue* is used to store the threads that are requesting packets. We use a hardware scheduler to scan the existing bit register ("port_rdy_status") and assign a ready port to the thread at the head of the queue, as shown in Fig. 6. In this way, when a thread is ready to receive and process a new packet, its ID is queued and the thread is put to sleep. The scheduler scans through the "port_rdy_status" register and assigns the ready port number "P_s" to the queue header "T_i." Thread "T_i" is then awakened to read a packet from "P_s." The scheduler scans through the register in a round-robin fashion.

The dynamic scheduler added to the interface controller will take some extra time to perform mapping between the ports and the threads. This is because the scheduler needs to read and test the bit one by one to find the first bit that is set. The thread's request also needs to be enqueued and dequeued, which are both extra operations compared to the static scheme. Though reading and testing the status register bit can be done very quickly, we conservatively charge 1 clock cycle of the 232 MHz NP to every operation, i.e., if m bits are scanned, m cycles are charged. We also charge 1 cycle to thread enqueue and dequeue, respectively. Our experiments show that the dynamic mapping effectively solves the "orphan" port problem.

4.3 Avoid Extra Packet Loss

In general, packet loss may happen when the incoming traffic load exceeds the maximum processing capacity of the NP. We cannot avoid this kind of packet loss since it is the nature of the NP, even if all the PEs are running. However, when we employ clock-gating technique to turn off some PEs, packets may be dropped when they come in burst, but the NP has not responded to such a burst. Specifically, the packets will quickly fill up the internal packet buffer so that when the buffer is full, new packets will be dropped. To avoid packet drops resulting from clock-gated PEs, we immediately wake up one PE to drain the packet buffer. Such fast switching is critical to ensure that the activation delay of PEs does not pose significant impact on the overall throughput of packet-loss sensitive applications, such as voice-over-IP [Shim et al. 2003].

A clock-gated PE can be awakened very quickly in several cycles [Li et al. 2003]. However, it still takes some cycles before an entry in the packet buffer can be cleared. This time includes some initialization of a thread upon execution and the time to put a thread into the thread queue. In the NP we modeled, this time is within 50 cycles. If a new packet arrives in this period, it cannot be captured and moved into the already saturated internal buffers.

Therefore, we need to use extra buffer space to hold the packets that arrive before a thread comes to fetch packets. The extra buffer space is calculated as follows. The delay before a thread is awakened to receive packets is about 50 cycles. The maximal packet throughput, we observed in NePSim, is about 1 Gbps. Thus we need about 30 bytes (1 Gbps $\times$ 50 cycles/232 MHz) extra buffer space. That is, there are, at most, 30 bytes coming into the NP during the initiation of a new PE. Since the IXP1200 fragments packets into 64-byte "mpackets," only one additional "mpacket" entry is needed to the packet buffer (RFIFO), as shown in Fig. 6. Thus, the extra buffer space needed to avoid packet loss is very minimal.

4.4 Putting It All Together

In summary, the NP keeps a counter, which is incremented when the thread queue length is greater than or equal to T, and is periodically reset by the "timer" (see Fig. 7). When the timer elapses or packet-buffer-full signal is asserted, the shutdown control logic will decide whether a state transition is necessary (implemented using a finite state machine). Upon a decision, it will

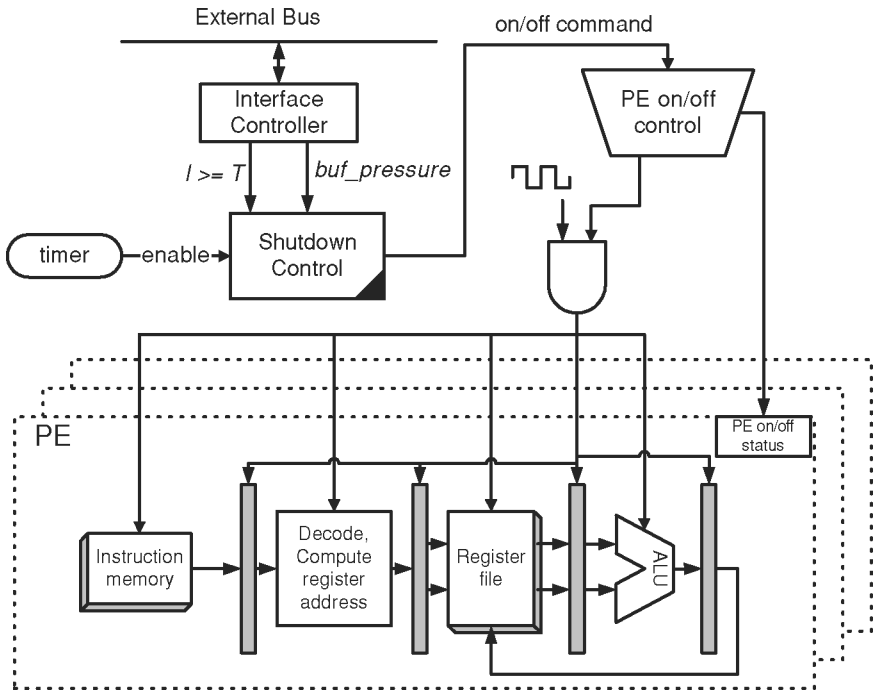

Fig. 7. Synopsis of dynamic clock gating of the PEs.

generate signals to the "PE on/off controller," informing what action to take on PEs. The controller then sets or unsets the terminating flag of the selected PE indicating whether or not it should prepare to stop. For simplicity, we choose the PE with the current lowest ID number to shutdown or activate. It then produces a clock enable/disable signal to the AND gate performing clock gating to the entire PE. Note that our shutdown technique is applied to PEs, not to other dedicated hardware units, such as accelerators and CAMs, because they are typically shared resources and are hardly free for shutdown.

Our proposed scheme is generally applicable to other network processors. However, the number of PEs available for shutdown depends on how they are organized for applications, i.e., the program/task allocation. Two general schemes are: *multiprocessing* and *context pipelining*. In *multiprocessing*, the application is replicated onto multiple PEs that execute in parallel to explore packet-level parallelism. In *context pipelining*, the program is divided into several stages that are assigned to different PEs to form a pipeline. Our scheme should be carefully applied to PEs where duplications exist. For example, if n PEs execute the same program, we have the opportunity to shut down $n - 1$ of them as long as the program semantics are not changed. If no duplication exists and every PE constitutes one stage in a context pipeline, then we have no opportunity to shutdown PEs. Advanced NPs, such as Intel's IXP2400/2800, AMCC's nP7510 [AMCC 2002], and Hifn's Rainier [Hifn], employ multiple PEs, among which duplications do exist. Such PE configurations are the trend in industry

to exploit the packet-level parallelism. Therefore, our scheme can be extended and used in other NPs.

5. CLOCK GATING

Many circuit-level techniques, such as voltage scaling and clock gating, were proposed to save dynamic power. Our previous work applied frequency and voltage scaling (DVS) to the NP design [Luo et al. 2004]. By partitioning the PEs into several domains operating at different supply voltages, both static and dynamic power savings are achieved. However, the adjustment of voltage and clock frequency requires long latency (e.g., 10 μs). This latency is acceptable for general-purpose processors, but not suitable for NPs that are plugged in mission-critical routers. During this time, no useful work can be conducted by the PEs and many packets might be dropped.

Compared with DVS, clock gating is safe because it is simple to implement and it only needs several cycles for switching on and off [Li et al. 2003]. The clock gating is implemented by granting a portion of the clock network using a special "enable" signal, namely, the normal clock signal AND'ed with the clock-enabling signal that is asserted by the clock-gating control logic. Once the decision is made, the target unit can be turned on or off in the next clock cycle. Clock gating saves dynamic power by both reducing power consumption in the clock-distribution network as well as switching activities in logic components. The clock accounts for a large portion of power consumption (20–50%) of the chip dynamic power, because clock signal has large capacitive load because of its high fan-out. By disabling the clock signals, we can effectively terminate all those activities in PEs.

5.1 Clock Power Model

To measure the potential power savings of clock gating, we add the clock-power modeling for various components to the NePSim. We follow the clock models in [Brooks et al. 2000; Duarte et al. 2002b] and make modifications according to physical features of the IXP1200. The major sources of clock power we consider include:

- Clock-distribution tree (wiring)—We implement a one-level H-tree, which is a common clock distribution topology. The wire lengths of the tree is obtained from the IXP1200 die photo [Halfhill 1999] (126 mm^2 in a 0.28 μm process).
- Clock buffers—Clock buffers are chained inverters with increasing gate sizes. The ratio between each stage and the number of stages are optimized for speed and minimal clock skews. We estimate the capacitance load of clock buffers using an analytical model as described in Duarte et al. [2002].
- Pipeline latches—The latches are a number flip-flops between pipeline stages. Their widths are estimated according to the inputs and outputs from each neighboring stage. The widths determine the number of flip-flops and the effective clock load capacitance.
- SRAM array bit-line precharge in memory structures—We assume the register files and the control store (where program is stored) use the classic

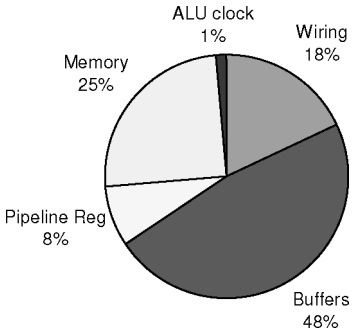

Fig. 8. Clock power breakdown in six PEs in NePSim.

six-transistor cell and a single precharge transistor per bit line. We obtain the precharge transistor size and the gate capacitance information from HSPICE.

- Clock gate capacitance in execution units—Execution units (e.g., ALU) are often implemented with dynamic logic blocks for high performance and less area. The clock signal drives the precharge gates so that the entire logic can be evaluated later. Hence, the precharge gate capacitances in dynamic logic modules are considered as clock load.

We use TSMC 0.25-μm technology parameters, which is consistent with NePSim, to estimate the power of the clock load. The clock load in PEs consumes 0.21 W; Fig. 8 captures how the different components contribute to the clock power within the PEs. Here the PLL's power is not included in the pie chart, because we assume it is located outside the PEs.

Besides the explicit clock power consumed in clock distribution network and precharge gates, there are dynamic logic modules that consume power because of the process of precharging/evaluating the storage nodes. With clock gating, we can eliminate the useless precharge stage during idle time so that power saving can be achieved.

- Execution units—We implemented the customized 32-bit ALU, which supports binary logic functions (i.e., AND, OR, NOT, XOR), addition, and subtraction in the Cadence toolset. The ALU latency is verified to be within 4.3 ns (1 clock cycle) through SpectreS simulations. The average power consumed by an ALU is 0.03 W.
- Wordline decoders—Modern caches use dynamic logic for the wordline decoding and driving. Hence, we treat the capacities of the wordline decoder and drivers as the clock load.

Figure 9 demonstrates the power consumed by individual components inside six PEs, obtained through taking unlimited uniform traffic input. We observe that the clock load on average consumes 22% of total PEs' power, while the dynamic logic modules (ALU, decoders, etc.) consume 21% of total PEs' power. During clock gating, we gate off the circuitry in the clock network, as well as the activity in the dynamic logic modules. Taking these two factors, we find the total clock-related power can achieve 43% of total PEs' power.

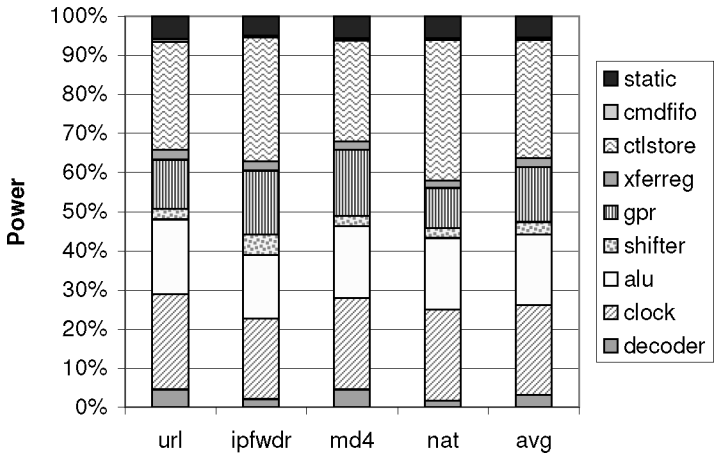

Fig. 9. Microengine power consumption breakdown.

For 0.25-μm technology, static power only constitutes a small portion ($\sim$2%) of the total power. As the technology size shrinks, static power will become a major contribution to the total power. Therefore, saving dynamic power alone is no longer enough. We will discuss how to reduce leakage power in nanometer NPs in Section 7.

6. EXPERIMENT EVALUATION

In this section, we first introduce our experiment environment. We then present the power/performance results after applying our technique on the NP.

6.1 The NePSim Tool

We use NePSim to produce our experiment results. NePSim is an open-source and fully parameterizable architecture-level simulator. The software development kits (SDK) distributed by Intel are not open-source and do not they power estimations. Other tools, such as the NP analytical models proposed by Franklin et al. [Franklin and Wolf 2003], are not suitable for our testbed, since we need to obtain accurate timing and power information in order to evaluate our design. The NePSim contains a cycle-accurate simulator for IXP1200, a power estimator, and a trace-based verification engine for testing and validation. The accuracy of the simulator is more than 95% of the IXP1200's real performance [Luo et al. 2004].

6.2 Benchmarks

There are four benchmarks ported in NePSim. They are *ipfwdr*, *url*, *nat*, and *md4*. The *ipfwdr* implements an IPv4 router, which forward IP packets between networks. The *url* is a content-aware routing program that routes packets based on their contained URL request. The *nat* is a network address translation program. The *md4* is a cryptography algorithm used in SSL or firewall to generate 128-bit digital signature on an arbitrary length message. We will use all these four benchmarks in our experiments.

The benchmark applications used in our study are representative applications in the network domain. *ipfwdr* and *nat* are two header-processing applications (HPA) and *md4* and *url* are two payload-processing applications (PPA). HPA and PPA are two major categories of network applications that are also included in well-known network benchmark suites, such as Commbench [Wolf and Franklin 2000], NetBench [Memik et al. 2001], etc. Most of the commercial software are closed-source and cannot be ported to NPs. The *ipfwdr* application comes with the Intel Software Development Kit. It is implemented in Microengine C language (a C variant specifically for Intel IXP network processors). *md4, nat*, and *url* are ported from public domain C code to Microengine C. We took a great effort in porting those programs, because there were no tools to convert C to Microengine C, and partitioning the data into different memory modules had to be done manually. Finally, these applications are compiled with Intel's Microengine C compiler to run on IXP network processors.

6.3 Input Traffic

We evaluate our design with real network packet traces (e.g., Leipzig-I) downloaded from NLANR [National Laboratory for Applied Network Research]. We experiment with several traces and present the results of Leipzig-I trace, because its link speed falls within the capacity of the IXP1200 NP system. Other traces with the same link speed generate similar results. The advantage of using real network traces is that they represent typical Internet traffic, in terms of packet size and arrival rate, seen by a router. However, because of the limited simulation speed of NePSim, it is too expensive to simulate entire traces of dozens of hours. Since NePSim simulates a multicore and multithreaded architecture, it usually takes more than 1 hour to simulate 1 s of real-world trace. We, therefore, sample a few seconds of real traffic with different arrival rates as individual inputs to the simulator. We scan the entire trace and divide it into 1-s slices. We calculate the average packet arrival rate, in units of Mbps, of individual slices. We choose four slices with the overall arrival rates of 90, 180, 360, and 480 Mbps, respectively that represent low to high traffic volume. In this way, we can obtain the typical traffic volume of a raw trace file and bound the simulation time in a reasonable range. We also adopt our statistical-sampling technique [Yu et al. 2005] to simulate a 24-hour network trace later, to show the potential power savings when the low-traffic volume during a day is fully exploited.

6.4 Power Overhead of the Control Logic

We extend the NePSim simulator with clock-power modeling (discussed in Section 5), so that we can measure the power savings of the clock gating. For the execution units, pipeline latches, and memory wordline decoders, the dynamic and static power is included, if it is not clock gated. If the circuit is clock gated in a cycle, zero dynamic power is added, while static power is still added.

We also take into account the power overhead associated with the additional control logics. We measure the power overhead of the counters, threshold registers, thread queue, and comparators with both Cadence and the Wattch model

using 0.25-μm technology. We find they consume negligible power, i.e., 0.00038 W for a 20-bit counter or register, 0.002 W for a comparator, and 0.0065 W for 24-entry thread queue. The extra buffer we add increases the buffer access energy by 3.4%, which is very small, considering the fact that the original receive buffer only contributes less than 2% of NP power [Luo et al. 2004]. In addition, the controller includes an adder and a finite-state machine (FSM), which makes the PE on/off decision. The FSM only has a handful of states. Both FSM and the adder are used just once in each time window, so their contribution to the overall power consumption is very small. We conservatively charge 2% of the total PE power as the overhead of the controller.

6.5 Parameter Values

Our power-saving scheme periodically checks if there is any opportunity to turn off a PE. We need to carefully choose the period length P in order to maximize the power saving. The smaller the P, the more shutdown opportunities we can exploit. However, a too small interval does not benefit, because, after a shutdown decision is made, it takes thousands of cycles for the PE to finish processing the current packet before gating its clock. Such latency can be as many as 60 K cycles (for *url*). We test several values and decide to use 1 M cycles as the shutdown period, since it can hide well the longest PE shutdown latency observed.

For the value of th, against which the thread queue length is compared, we test both the static and dynamic threshold. In static threshold scheme, the th is set to 500 K cycles (one-half of P) to represent a medium aggressiveness. In the dynamic threshold scheme, the initial th value is also set to 500 K cycles. It is then adjusted by 2% positively or negatively to adapt to dynamic traffic volume. To measure the quality of the static and dynamic thresholds, we compare them with the optimum case in terms of achievable total power savings. In the optimum case, an oracle control logic decides the minimum number of PEs required in every cycle. Thus, the optimum resource management gives the upper bound of energy saving for a given traffic load. We compare the three schemes under low traffic load (around 180 Mbps arrival rate). We observe that both the static threshold and dynamic threshold achieve near optimum power savings for the four benchmarks. The low traffic causes the PEs to be shut down very quickly in the first few periods, and the PEs remain sleeping in the later periods. Thus adjusting threshold as in a dynamic threshold does not make a noticeable difference under low traffic load. In observation of this phenomenon, we perform further experiments with medium traffic load (around 480 Mbps arrival rate). The dynamic threshold scheme shuts down more PEs than static threshold by 2%. The fluctuations in the traffic forces PEs to be turned back on several times. Hence, the dynamic threshold scheme provides useful feedback to the shutdown control logic. Thus it can exploit more shutdown opportunities than the static scheme. When we compare the two schemes with the optimum case, we observe that they are 10% worse in total power savings. This is because the optimum case, which changes the PE configuration instantaneously, is overly optimistic, because turning off PEs takes a long time. Our conclusion is that, overall, both

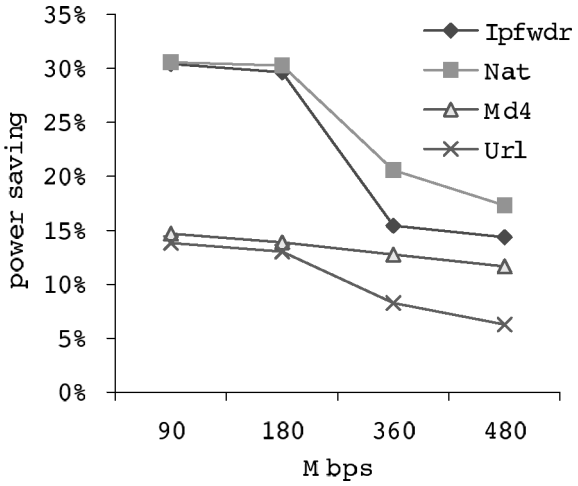

Fig. 10. Power saving versus packet arrival rate.

static threshold and dynamic threshold can effectively shut down the PEs under different traffic loads and dynamic threshold is better under the fluctuated medium traffic load. In the following section, we will present the results using a dynamic threshold. The metrics we evaluate are power consumption (in W), throughput (in Mbps), and PE utilization.

6.6 Experiment Results

We scan the downloaded traces and extract four segments with different packet arrival rates. We feed each segment of traces to the 16 input ports of NePSim.

6.6.1 *Power Savings.* Figure 10 shows the power saving of four benchmarks at different input traffic loads. The power savings are significant in all cases we tested. At the lowest traffic load, up to 30% of the power can be saved for *ipfwdr* and *nat*. *md4* and *url* saved about 15 and 14%, respectively. As the traffic load increases, the power saving decreases, because there are fewer power-saving opportunities that can be exploited. At the highest traffic load, power-reduction numbers are the lowest, but there are still 17, 15, 12 and 6% of the total power saved for *nat*, *ipfwdr*, *md4*, *url* , respectively. Among the four benchmarks, *nat* has the most power savings while *url* has the least. This is because the per-packet processing time of *nat* is the shortest, so, on average, the thread queue is the longest and the power-saving opportunities are the greatest. On the other hand, *url* has the longest processing time, resulting in the shortest thread queue and the least PE shutdown opportunities.

Note that the power saving in Fig. 10 is estimated for 1-s periods. To illustrate the energy savings in a much longer time span, such as full-day traffic with both high and low periods, we conducted simulations using the statistical input sampling method we developed for efficient simulations with high confidence and low error [Yu et al. 2005]. We use a 24-hour workload, shown in Fig. 1, for the experiment. The basic idea of the input sampling is to classify the long trace to

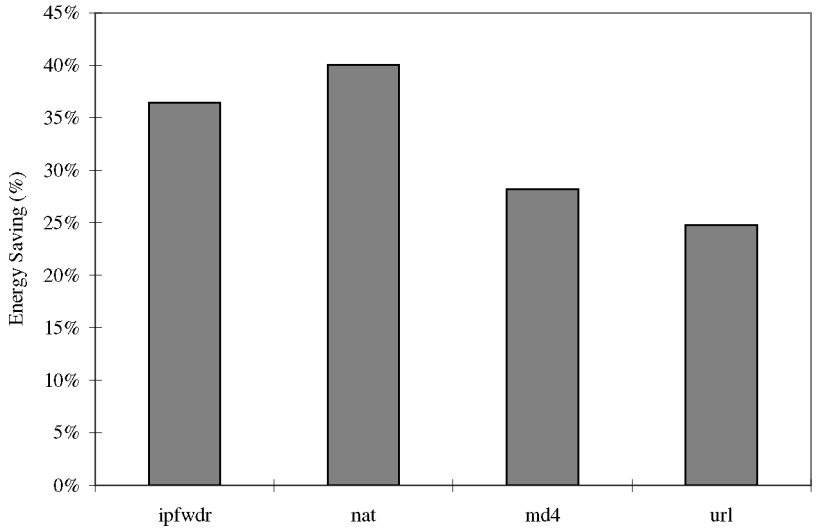

Fig. 11. Energy savings over a 24-hour period, which includes both high- and low-traffic volumes.

several groups, based on the arrival rates and average packet size, and sample randomly from each group for detailed simulation. The sample size should be estimated according to the variations of the performance metrics, i.e., arrival rates, packet size, power consumption, and the targeted accuracy. Typically, the larger the variation, the larger the sample size. The sampled traffic in this experiment is 250-s long for achieving ±3% simulation error with 95% confidence. We run detailed simulations using the sampled traffic and multiply the simulation results in each group with the weight of the corresponding group. The weight of each sampled period takes into account the sample size in each group and the percentage of that group in the entire population. The results are shown in Fig. 11. We observe that the four benchmarks have energy saving ranging from 23 to 40%, higher than that shown in Fig. 10. This is because most time slices in the 24-hour trace have less than 90 Mbps arrival throughput. The NP has plenty of opportunities to turn off 2–3 MEs. *ipfwdr* and *nat* have much higher energy saving than *md4* and *url*. This is consistent with the results in Fig. 10. *ipfwdr* and *nat* have shorter packet-processing time than *md4* and *url*. Thus, they have higher resource redundancy for saving power.

Figure 12 demonstrates how the number of PEs varies with packet arrival rates using our dynamic-threshold shutdown scheme. The benchmark shown here is *url* and other benchmarks have similar variations. The x axis shows time. The left/right y axis shows the arrival throughput/number of active PEs. The bottom two graphs zoom into two periods—4 AM to 6 AM and 14 PM to 16 PM—which represent the light and heavy workload, respectively. We observe that with light traffic load between 4 AM and 6 AM, using three PEs can service the arriving packets most of the time. Whereas during the busy time between 14 PM and 16 PM, four to six PEs are required to service the packets. Note that the data for the PE number (lower portion of the graph) are not distributed evenly, because we used stratified sampling technique and carried only detailed

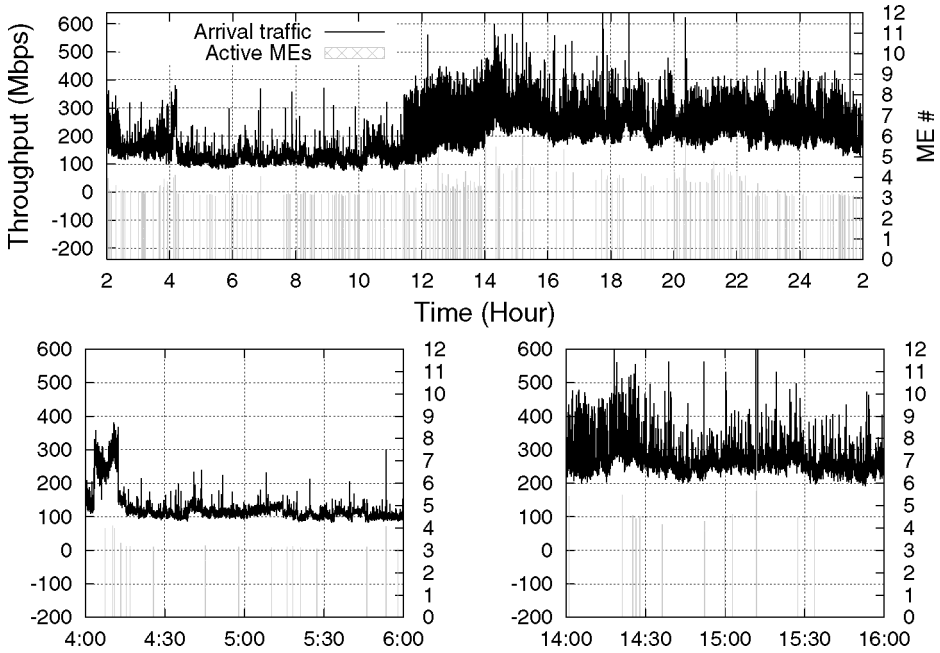

Fig. 12. Number of PEs varies with arrival rate over a 24-hour period.

simulation at the sampled periods. Our experiment shows that the number of PEs can vary promptly in response to the traffic fluctuation. There are plenty of opportunities to shut down PEs during nonsaturated traffic load.

6.6.2 *Throughput Changes.* Our clock gating scheme has very little impact on the system throughput, as shown in Fig. 13. Deactivating PEs reduced throughput by at most 4% (*url* with high traffic load). When less number of PEs are active, the packets tend to stay longer in the internal buffer before they are processed and drained out of the NP system. As a result, the system throughput decreases. Note that lower throughput does not imply packet losses; there is no packet loss with the help of the extra one-entry buffer. In addition, if the traffic load continues to increase toward the NP system capacity, the internal buffer will become saturated and clock gating to PEs will not be applied. Thus, there will be no reduction of throughput in such situations.

6.6.3 *Increasing PE Utilization.* From a different perspective, our low-power technique exploits low utilization of NP under low network traffic. By turning off PEs we effectively improve PE utilization while saving power significantly. We plot the utilization of the active PEs in Fig. 14 under different traffic loads. Each part of the figure compares the utilization of the active PEs with and without clock gating (base case). Figure 14 shows that in all the traffic load we tested, shutting down PEs, improved the utilization of the active ones by up to 20% (*url* under 180 Mbps).

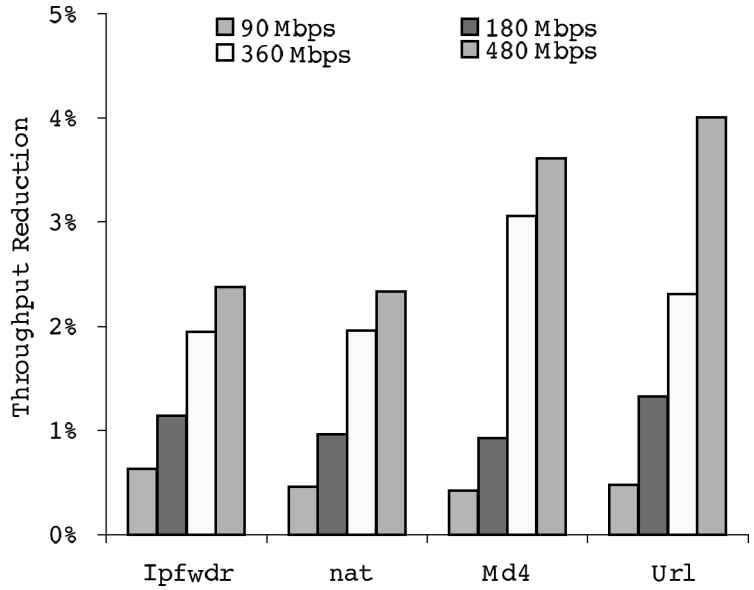

Fig. 13. Throughput reduction versus packet arrival rate.

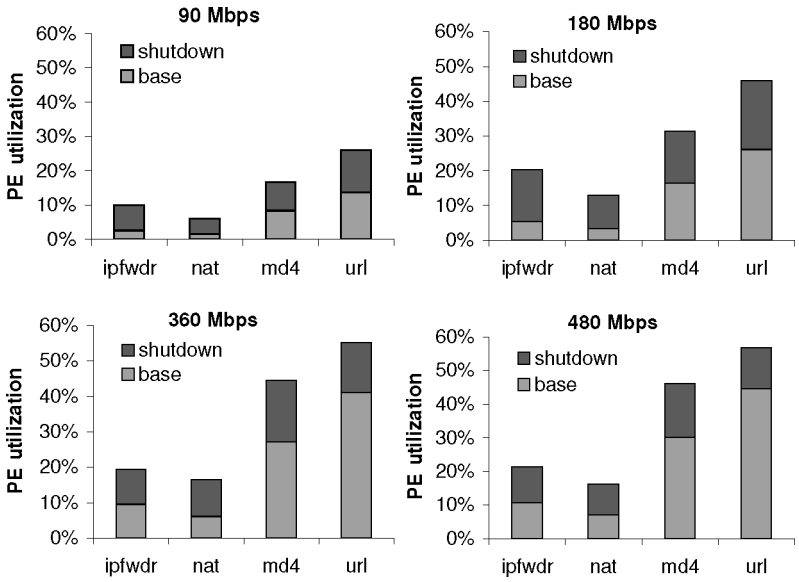

Fig. 14. ME utilization.

7. LEAKAGE POWER SAVING

The clock-gating technique we used here saves the dynamic power of the PEs only. The clock-gated components still consume certain leakage power. The IXP1200 we model currently uses 280-nm technology, for which leakage is not a significant problem. However, future generations of NPs may incorporate smaller feature sizes for higher clock frequency and transistor density. This will bring the leakage power consumption to a significant portion, as projected by the International Technology Roadmap for Semiconductors [Semiconductor Industry Association 2001]. Therefore, we need to use leakage-control techniques, such as power gating [Liao and He 2005; Duarte et al. 2002; Powell et al. 2000] in conjunction with clock gating.

We mainly consider the multithreshold CMOS (MTCMOS) and virtual power/ground rails-clamp (VRC) techniques as they are believed to be most effective in leakage reduction [Liao and He 2005]. In MTCMOS, a sleep transistor with high-V_t is inserted between the low-V_t circuits and GND. The sleep transistor can be turned off to reduce leakage power. However, this technique is stateless, meaning that any logic state is lost because of the leakage saving. Data retention in instruction memory of the NP is critical to our design as it takes tens of thousands of cycles to restore the instructions into a PE's instruction memory for wakeup. This delay would bring significant packet loss, which is undesirable in high-performance routers.

Therefore, we prefer a data-preserving leakage-saving technique, such as the VRC. In VRC, a diode is inserted in parallel with the sleep transistor to maintain the necessary voltage level for keeping the logic states in the circuits. VRC, on the other hand, introduces more transition energy and has a lower leakage-reduction ratio than MTCMOS. We, therefore, propose to use VRC in PE's instruction memory and MTCMOS in the other components. Note that we do not need to preserve the contexts of the threads, i.e., register file content. This is because when we shut down PEs, we make sure all the threads finish processing the current packets. When the PEs are awakened, the threads resume execution for new packets with new context.

We present a quantitative study of the leakage savings employing both the MTCMOS sleep transistor and the VRC. We study the ideal power gating and present the upper bound of the power saving. We assume turning off the sleep transistor encounters negligible overhead and the sleep transistors can effectively reduce the leakage power to zero. In practice, there still exists reduced static leakage power when power gated, but this reduced amount of leakage power is very small [Liao and He 2005]. We run experiments in NePSim using the 100-nm technology size. All the parameters are scaled from 250 to 100 nm as in Brooks et al. [2000]. The Technology Roadmap for Semiconductors (ITRS) [Semiconductor Industry Association 2001] predicts that in the next several processor generations, leakage power will approach 50% of total power dissipation. We, therefore, charge dynamic power and 50% of the maximum total power as leakage power for each component, if PEs are turned on. When a PE is turned off, we charge zero dynamic power and zero static power for that PE. Figure 15 compares the power reductions with and without a leakage-saving

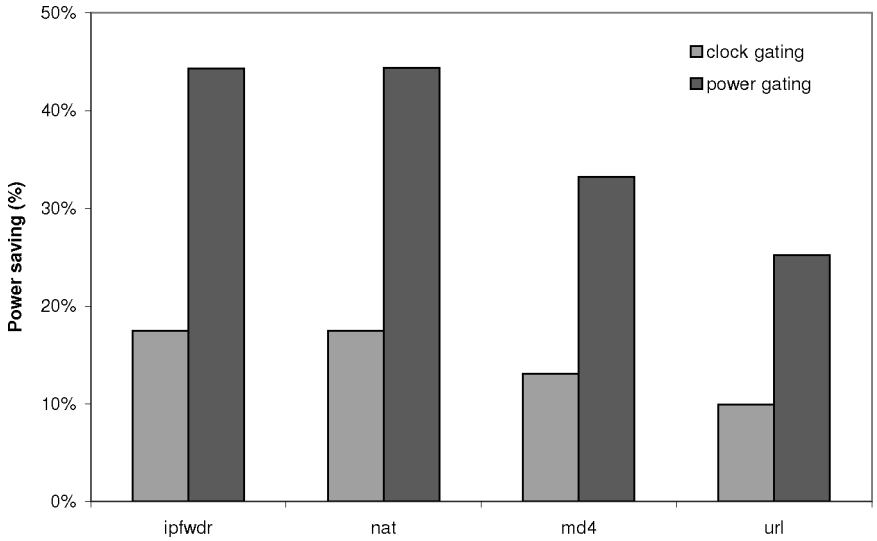

Fig. 15. Comparison of power saving using clock and power gating (leakage saving) with 100-nm process technology.

technique ("power gating") using a medium traffic load (around 360 Mbps). We observe that with future technology sizes, saving leakage power lead to a more significant energy savings than saving dynamic power alone. This additional amount of leakage energy saving constitutes 15 to 27% of total power. This result is consistent with the previous study on the SPEC2K benchmarks in VLIW processors [Liao and He 2005].

8. RELATED WORK

Power dissipation is one of the primary concerns for NP design. Over the past few years, several power-reduction techniques have been proposed for contemporary NPs. Franklin and Wolf developed an analytic performance-power model for typical NPs. They explored the design space of NPs and showed performance–power tradeoffs for different core and memory configurations [Franklin and Wolf 2003]. However, no specific low-power techniques were investigated with the proposed model. Other works focused on lowering the power in individual components of an NP. Kaxiras et al. proposed an IPStash memory architecture as a TCAM (used in packet classification and routing) replacement, which significantly reduces the memory set associativity and, thus, power [Kaxiras and Keramindas 2003]. Mallik and Memik [2004] investigated the optimal operation frequency of the data caches, where reliability is compromised for reduced energy and increased performance. This is justified by observing that errors in NPs can be fixed by higher levels of network protocol stack. Memik and Mangione-Smith [2002] proposed a data filtering engine (DFE) that processes data with low locality before it is placed on the system bus. The DFE is an execution core with additional features to control the passing of the memory data to the bus. By offloading

the code that contains low temporal locality load instructions to DFE, less bus and L2 cache accesses can be achieved. Thus the overall power can be reduced.

The work that are most related to our scheme are the system-level designs, which target the whole processor power reduction. Srivastava et al. [1996] proposed predictive shut down on portable devices. They predicted the next idle time, based on the knowledge from history, and then shut the processor down by shutting off the clock or power supply, if the predicted length of idle time justifies the cost. Chheda et al.[2004] proposed compiler-driven static IPC estimation schemes to adaptively adjust voltage and speed, as well as the instruction fetch rate. This approach is not suitable for NPs as IPC does not imply the relation between processing power and input traffic. Thread migration and adaptive resource allocation are common approaches to improving power efficiency for NPs or general CMPs. Kokku et al. [2004] presented an analytical model for allocating appropriate number of processors to each service on NPs. Their work made several assumptions to derive the estimate for the benefits in an ideal adaption scheme, while our work solves practical problems encountered in power management. Therefore, our work is not comparable to Kokku et al. [2004]. EPI throttling [Annavaram et al. 2005] exploits several approaches, i.e., voltage scaling and thread migration, to maximize performance given a fixed-power envelope on CMPs. Our approach is different from EPI throttling because our goal is to reduce power consumption by exploiting traffic variations rather than maximizing performance for a given power budget. The system level schemes discussed so far are flexible in the sense that the resource adjustments can be performed at fine granularity. However, they also require assistance from the operating system or the compiler, in addition to the adaption at the hardware level.

Many circuit-level low-power techniques are also related to our work. To save dynamic power, many approaches were proposed to reduce the component's switching activities, voltage, and capacitance. Our previous work [Luo et al. 2004] presented how to apply dynamic voltage scaling (DVS) to reduce NP's power. Compared to clock gating, DVS has longer delays for adjusting the voltage and clock frequency. To save leakage power, techniques proposed include dual V_T, stacked transistors, body bias, and sleep transistors. In dual V_T technique [Tschanz et al. 2002], low-V_T devices are used in the critical path of a design, while high-V_T devices are used to reduce the leakage in the noncritical parts of the design. The stacked transistors [Ye et al. 1998] reduce leakage through transistor stacks to maximize the number of transistors that are "off" during the idle mode. Body bias [Thompson et al. 1997] entails dynamical changes in the body bias to reduce the leakage current. Using sleep transistors, which are often referred as power gating or MTCMOS [Liao and He 2005], is the most effective technique to reduce chip-level leakage. A recent study [Tschanz et al. 2003] shows that pMOS sleep transistor achieves 15% power reduction and dynamic body bias has 8% total power reduction on a 4-GHz, 1.3-V, 130-nm 32-bit integer execution core. Our scheme thus uses power gating to save leakage power. Other techniques might be also used in conjunction with power gating.

Numerous other works focus on boosting NP performance, i.e., packet processing throughput. To name a few, Hasan et al. [2003] proposed a series of techniques to improve packet memory throughput, and, hence, the packet throughput. Sherwood et al. [2003] proposed a pipelined memory design that emphasizes worst-case throughput over latency, and coexplored architectural trade-offs. Spalink et al. [2001] experimented using IXP1200 to build a robust inexpensive router that forward minimum-sized packets at a high throughput.

9. CONCLUSION

We investigated that under nonsaturated incoming traffic rates, power consumption of an NP can be saved by putting its processing elements into a low-power mode with no impact on packet loss ratio and little impact on overall system throughput. Our technique can be easily employed in various scenarios, since it adapts to traffic variation without imposing too much overhead at runtime. With current NP technology, significant dynamic power can be saved through efficient clock-gating PE components. In future generations of NP where technology size enters nanometer scale, leakage power becomes dominant and its reduction can be achieved through state-preserving power-gating techniques, in addition to clock gating. The effectiveness of our proposed power-reduction schemes have been demonstrated quantitatively in this paper.

REFERENCES

AMCC. 2002. Amcc np7510 product manual. Applied Micro Circuits Corp. Sunnyvale, CA.

ANNAVARAM, M., GROCHOWSKI, E., AND SHEN, J. 2005. Mitigating amdahl's law through epi throttling. *Proceedings of International Symposium on Computer Architecture*.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of International Symposium on Computer Architecture*. 83–94.

CHHEDA, S., UNSAL, O., KOREN, I., KRISHNA, C., AND MORITZ, C. 2004. Combining compiler and runtime ipc predictions to reduce energy in next generation architectures. In *Proceedings of the 1st Conference on Computing Frontiers*. 240–254.

CISCO. Cisco crs-1 carrier routing system 8-slot line card chassis system description.

DUARTE, D., TSAI, Y., VIJAYKRISHNAN, N., AND IRWIN, M. 2002a. Evaluating run-time techniques for leakage power reduction. In *Proceedings of International Conference on VLSI Design*.

DUARTE, D. E., VIJAYKRISHNAN, N., AND IRWIN, M. J. 2002b. A clock power model to evaluate impact of architectural and technology optimizations. *IEEE Transactions on VLSI Systems 10*, 6, 844–855.

FRANKLIN, M. AND WOLF, T. 2003. Power considerations in network processor design. In *Workshop on Network Processors in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*. 10–22.

HALFHILL, T. 1999. Intel network processor targets routers. *Microprocessor Report 13*, 12 (Sept).

HASAN, J., CHANDRA, S., AND VIJAYKUMAR, T. N. 2003. Efficient use of memory bandwidth to improve network processor throughput. In *Proceedings of International Symposium on Computer Architecture*. 300–313.

HIFN. Hifn 5np4g network processor data sheet. Hifn Corporation, Los Gatos, CA.

INTEL. 2000. Ixp1200 network processor family hardware reference manual. Intel Corporation, Santa Clara, California.

INTEL. 2004. Intel ixp2xxx product line of network processors. Intel Corporation, Santa Clara, California.

KAXIRAS, S. AND KERAMIDAS, G. 2003. Ipstash: a power-efficient memory architecture for ip-lookup. *Proceedings of International Symposium on Microarchitecture*. 361.

KOKKU, R., RICHE, T., KUNZE, A., MUDIGONDA, J., JASON, J., AND VIN, H. 2004. A case for run time adaption in packet processing systems. *ACM SIGCOMM Computer Communication Review 34*, 1.

LI, H., BHUNIA, S., CHEN, Y., VIJAYKUMAR, T., AND ROY, K. 2003. Deterministic clock gating for microprocessor power reduction. In *Proceedings of International Symposium on High Performance Computer Architecture*. 113.

LIAO, W. AND HE, L. 2005. Microarchitecture-level leakage reduction with data retention. *IEEE Transactions on Very Large Scale Integration Systems 13*, 11.

LUO, Y., YANG, J., BHUYAN, L., AND ZHAO, L. 2004. Nepsim: A network processor simulator with power evaluation framework. *IEEE Micro*.

MALLIK, A. AND MEMIK, G. 2004. A case for clumsy packet processors. *Proceedings of International Symposium on Microarchitecture*.

MEMIK, G., MANGIONE-SMITH, W. H., AND HU, W. 2001. Netbench: A benchmarking suite for network processors. In *Proceedings of International Conference on Computer Aided Design*. 39.

MEMIK, G. AND MANGIONE-SMITH, W. H. 2002. Increasing power efficiency of multi-core network processors through data filtering. *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 108–116.

NATIONAL LABORATORY FOR APPLIED NETWORK RESEARCH. The nlanr measurement and network analysis. National Laboratory for Applied Network Research (http://www.nlanr.net/).

PAPAGIANNAKI, D., TAFT, N., ZHANG, Z., AND DIOT, C. 2003. Long-term forecasting of internet backbone traffic: Observations and initial models. In *Proceedings of IEEE INFOCOM*.

POWELL, M. D., YANG, S., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. N. 2000. Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of International Symposium on Low Power Electronics and Design*. 90–95.

SEMICONDUCTOR INDUSTRY ASSOCIATION. 2001. International technology roadmap for semiconductors. Semiconductor Industry Association, San Jose, CA.

SHERWOOD, T., VARGHESE, G., AND CALDER, B. 2003. A pipelined memory architecture for high throughput network processors. In *Proceedings of International Symposium on Computer Architecture*. 288–299.

SHIM, C., XIE, L., ZHANG, B., AND SLOANE, C. 2003. How delay and packet loss impact voice quality in voip. Qovia Inc., Fredereck, MD.

SPALINK, T., KARLIN, S., PETERSON, L., AND GOTTLIEB, Y. 2001. Building a robust software-based router using network processors. In *Proceedings of International Symposium on Operating Systems Principles (SOSP)*. 216–229.

SRIVASTAVA, M., CHANDRAKASAN, A., AND BRODERSEN, R. 1996. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 4*, 1.

THOMPSON, S., YOUNG, I., GREASON, J., AND BOHR, M. 1997. Dual threshold voltages and substrate bias: keys to high performance, low-power, 0.1 μm logic designs. *Symposium on VLSI Circuits Digest of Techical Papers*, 69–70.

TSCHANZ, J., YE, Y., WEI, L., GOVINDARAJULU, V., BORKAR, N., BURNS, B., KARNIK, T., BORKAR, S., AND DE, V. 2002. Design optimizations of a high-performance microprocessor using combinations of dual-vt allocation and transistor sizing. *Symposium on VLSI Circuits Digest of Techical Papers*, 218–219.

TSCHANZ, J., NARENDRA, S., YE, Y., BLOECHEL, B., BORKAR, S., AND DE, V. 2003. Dynamic sleep transistor and body bias for active leakage power control of microprocessors. *IEEE Journal of Solid State Circuits 38*, 11.

WOLF, T. AND FRANKLIN, M. A. 2000. CommBench - a telecommunications benchmark for network processors. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Austin, TX, 154–162.

YE, Y., BORKAR, S., AND DE, V. 1998. A new technique for standby leakage reduction in high-performance circuits. *Symposium on VLSI Circuits Digest of Techical Papers*, 40–41.

YU, J., YANG, J., CHEN, S., LUO, Y., AND BHUYAN, L. 2005. Enhancing network processor simulation speed with statistical input sampling. In *International Conference on High Performance Embedded Architecture and Compilers*.

Received August 2005; revised March 2006; accepted July 2006