Low Power Network Processor Design Using Clock Gating

***Abstract*— Network processors (NPs) have emerged as successful platforms to 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, most processing elements (PEs) in NPs are nearly idle and yet still consume dynamic power. This paper develops a low power technique to reduce the activities of PEs according to the varying traffic volume. We propose to monitor the average number of idle threads in a time window, and gate off the clock network of unused PEs when a subset of PEs is enough to handle the network traffic. To accommodate different applications and network parameters (i.e. packet size, arrival rate), the thresholds of turning on/off PEs will be dynamically tuned on-the-fly. We show that our technique brings significant reduction in power consumption (up to 30%) of NPs with no packet loss and little impact to the overall throughput.**

I. Introduction

In today's capital constrained environment, traditional focus on network growth has been shifted to service profitability. 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 processors for packet processing. New generation of 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, product's life time and lower cost.

A number of challenges for NP implementation are already evident, and power dissipation is one of them. Typical routers mount a few racks containing groups of line cards (e.g. 8, 16) each of which contains one or two NPs. Such routers are extremely dense in power dissipation (e.g. 500 Watts per line card), which causes high operating temperature. On the other hand, with the demand of increasing performance, NP's clock frequency is increasing and more PEs will be put on an NP. For example, Intel IXP2850 contains 16 Microengines operating at 1.6GHz with 19$\sim$ 25W power consumption [16], while it's predecessor IXP1200 contains 6 Microengines operating at 232MHz with 4.5W power consumption.

This paper develops a low power technique by exploiting the unbalanced network traffic load. It is known that routers experience different workload during different time of a day. From some of the NLANR [14] router traces, it can be observed that traffic volume varies, for example, from 5 to 50 Mbps in a 24-hour period with low rates at the nighttime. This implies that much less processing power is required at nighttime compared to the daytime, which leaves the NP underutilized. This phenomenon brings challenges and opportunities for low power NP design.

We propose a low power technique to save active power of NPs without sacrificing the throughput. Our approach is to use the clock gating technique on PEs when the packet processing requirement is low, and reopen 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 are more processing power than the amount required by the incoming packets. Therefore, we propose to use the number of idle threads to determine how many active PEs are necessary. If the NP can process packets with one fewer PEs, then we 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 happen. Our goal here is not to introduce *extra* packet loss due to clock gating the PEs, but to guarantee enough process capability at low power consumption.

We investigate the potential problems after turning off the PEs and give solutions to overcome them. We design techniques to avoid possible extra packet loss due to turning off of PEs. To accurately measure and test the effectiveness of our technique, we implemented our scheme in an NP simulator [7]. We added clock power modeling to the simulator, and also studied the proper timing to apply clock gating in NP. We measured the power savings and throughput using real world router traces from NLANR [14]. Our experiments show that up to 30% of power savings can be achieved when the traffic is non-saturated.

The rest of the paper is organized as follows. In section II we introduce the network processor model that will be used in our design. Section III provides an overview of the dynamic PE turning off methodology. Section IV discusses how to solve the problems introduced by turning off PEs In section V, we make it clear why clock gating technique is chosen to reduce power, and then give details on how we modeled clock power. We show the results of our clock gating technique in section VI and discuss related work in section VII. Finally, section VIII concludes this paper.

II. Network Processor Model

A network processor usually contains multiple 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 [7] to model the NP architecture. NePSim is based on Intel IXP1200 and includes a cycle-accurate architecture simulator and a power estimator. Although IXP1200 was built several years ago and has been advanced to IXP2400/2850 recently, it still represents a prevalent class of NP configuration in which the PEs perform similar functionality in parallel. All the configurations in NePSim are parameterizable, while the software development kits (SDK) distributed by vendors are not open-source, nor do they provide power estimations. Other tools such as the NP analytical models proposed by Franklin *et al.* [3] are not suitable for our testbed since we need to obtain accurate timing and power information in order to evaluate our design.

We therefore decide to use NePSim as a testbed to experiment the applicability of clock gating PEs. *Our technique can be generalized to multi-PE-based NP architectures with differences in specific implementations only.*

The reference model of the network processor design follows IXP1200 and consists of a StrongARM processor, six multi-threaded PEs called microengines (MEs), memory interfaces, high-speed bus

interfaces. The StrongARM processor is used for management functions such as initializing MEs, running routing protocols, exception handling. The MEs can be programmed to perform any high-speed packet inspection, data manipulation, and data transfer. The memory modules outside IXP1200 are used mainly to store control data information and packets temporarily. The bus interface, IX bus unit, transfers data packets between MEs and network interfaces. The usage of each component is highly dependent on the application and workload.

III. Policies of Deactivating and Reactivating Processing Elements

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

A. Selecting the parameters

The idea of turning off PEs originates from the observation of the network traffic variation over time. Such variation is usually specified in terms of packet arrival rate in unit of megabits per second (Mbps). It is natural to use this information as a guide to making decisions. However, it is very difficult to set common thresholds on arrival rate because different applications support different line speeds.

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

Alternatively, a PE should be turned off(on) when the required workload for the entire NP is low(high) and fewer(more) number of PEs are enough to handle the workload. Such a status can be indicated by (1) the idle time of a PE during which the PE does no effective 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. A multi-threaded PE sleeps when all of its threads go to sleep. A thread is put into sleep mode when it needs to wait for certain events. Two major events are memory access and new packet arrival, as they are both time consuming. Thus, the idle time of a PE is a mixture of different events, not a direct suggestion of low workload even though the PE is not doing any useful work when idle. As a result, we will not use this parameter in guiding the decision of turning off the PEs.

The thread queue holds threads that are waiting for the arrival of new packets. We will explain in section IV.B how the thread queue is implemented. When a new packet arrives, the head of the queue wakes up to service the packet. 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 waiting 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 clock gate a PE.

The internal packet buffer is a place to hold temporarily the incoming packets before a thread fetch them for processing. In IXP1200, the RFIFO is a buffer of this kind [15]. 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 IV.C 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 of them are monitored on-chip. The queue length is compared with certain thresholds at fixed time intervals. If a pre-defined condition is satisfied, the PEs will be turned off or on. Next, we will discuss how to determine the thresholds for the parameters.

B. Determining the thresholds

As explained earlier, the length of the thread queue, l, indicates the number of free threads that are present in the NP due to the low packet arrival rate. When the system reaches a stable state, the number of active threads have enough processing power for the incoming packets, and l thus represents the excessive processing power of the NP. If the number of threads each PE supports is T, then when l is greater than T, it means that we could use one fewer PE to sustain the network traffic. However, l is a varying number since the packet arrival rate is varying due to the network conditions. Therefore, we need to estimate the averaged l during a period of observation time, P, 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 the value of 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 shutdown PEs. The smaller the th the more aggressive our scheme is. We initially set th as half of P, i.e., if over 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 agressiveness. After that, we allow th to be updated dynamically so that different applications and traffic condition can find their own appropriate thresholds. This is because the th is very application and network condition dependent. The updating algorithm is as follows. The th is increased (up to the value of P) if negative impacts, such as internal packet buffer full, have been observed. On the other hand, th is lowered if it has not introduced negative impact to the NP. This means that there still might be excessive processing power left on average. The advantage of allowing th to adjust itself is that we can let th converge for an application without the need to find a common value across all applications.

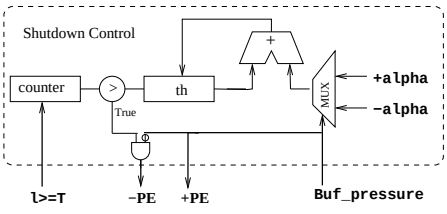

Fig. 1. Control logic for PE configuration transition.

We then develop a control logic, as illustrated in Figure 1, for choosing PE configurations using the thresholds. $counter$ stores the number of cycles when $l >= T$. We turn off a PE if $counter$ is greater than th. When buffers experience pressure, i.e., the buffer is full, we turn a PE back on. During every period P, th is increased by

$alpha$ if the internal packet buffer has been full once, or is decreased by $alpha$ otherwise.

IV. Deactivating the Processing Elements

A. Terminating threads gradually

To turn off a PE completely, the hardware needs to terminate all the threads first. However, the threads are 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 occupied spaces in the packet buffer (or memory) but nobody would move it out of NP, creating "leakage" in resources which would eventually be drained out.

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 IXP1200, 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 relinquish the pipeline resources. Therefore, implementing this part requires a flag bit per PE and an extra conditional branch and "kill" instruction in the program.

Since the PEs are multi-threaded processors, at anytime 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 a decision is made and the PE is truly turned off, we measured the time across all the benchmarks. We found turning off PEs needs up to tens of thousands of cycles which amounts to 0.0663∼0.240ms at a 232MHz clock rate. This time varies from different application since the receiving threads are responsible for processing the packets and different applications have different processing complexity.

B. Reschedule packets for orphan ports

In some NPs such as IXP1200, the receiving ports are statically allocated to the PEs that receive and process the packets from those ports. Hence, when a PE is turned off, the network ports from which the PE reads packets become "orphans", i.e., there will be no threads that receive and process packets from those ports. 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 [16] 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.

The dynamic mapping is accomplished by adding very simple hardware in the interface controller. We used a *thread queue* to queue up 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 Figure 2. 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 waken up 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. Also, the thread's request needs to be enqueued and dequeued which are both extra operations

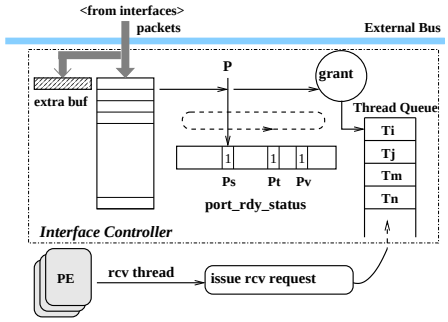

Fig. 2. Implementation of dynamic thread-port mapping.

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 232MHz 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. In addition, the controller will also take up some power which will be included in calculating the net power savings of the NP. Our experiments show that the dynamic mapping effectively solves the "orphan" port problem.

C. 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 lost 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, and when the buffer is full, new packets will be dropped. We have discussed earlier that when the buffer is full, we immediately wake up a PE to drain the packet buffer.

A clock-gated PE can be waken up very quickly in several cycles [6]. However, it still takes some more 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 ready to receive packets is about 50 cycle. The maximal packet throughput we observed in NePSim is about 1Gbps. Thus we need about 30 bytes (1Gbps*50cylces/232MHz) 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 Figure 2. Thus, the extra buffer space needed to avoid packet loss is very minimal.

D. Putting it all together

To summarize all the schemes discussed in the previous sections, 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 Figure 3). When the timer elapses or packet-buffer-full signal is asserted, the shutdown control logic (Figure 1) will decide whether a state transition is necessary (implemented using a finite state machine). Upon a decision, it will generate signals to the "PE on/off controller", telling it 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. 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 modern NPs tend to have large number of power-hungry PEs. We will investigate power saving techniques for dedicated hardware units in the near future.

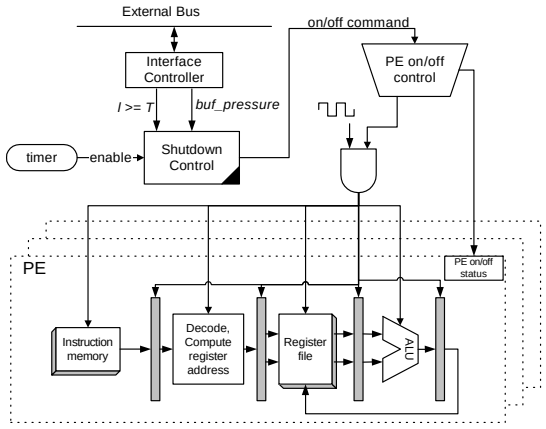

Fig. 3. Synopsis of dynamic clock gating of the PEs.

V. Clock gating

There are many circuit-level approaches proposed to save dynamic or static power. Examples are dynamic voltage scaling(DVS), power gating, clock gating etc. Luo et al. [7] proposed to lower clock frequency and supply voltage (DVS) when there is abundant PE's idle time. By partitioning the PEs into several domains operating at different supply voltages, both static and dynamic power saving are possible. However, the adjustment of voltage and clock frequency requires long latency (e.g. 10μs). This period of time is short for general embedded systems, but not for a NP which supports Gbps throughput. During this time, no useful work can be conducted by the PE, and many packets might be dropped. Similarly, the power gating technology, which has been implemented in the form of "sleep" modes in embedded systems, has notable latency considering huge capacitance on the power supply nodes in a unit. Compared with the above two techniques, clock gating is safe because it's simple to implement and requires a latency of only several cycles [6].

Clock gating essentially disables the clock to a circuit to save power by both preventing unnecessary activity in logic modules and by eliminating power dissipation on clock network. Clock gating can be applied in either *fine-grained* or *coarse-grained* manner as shown in Figure 4. *Fine-grained* allows us to reach miscellaneous small units in clock sinks and aggressively save their dynamic power even for a few cycles. *Coarse-grained* gating saves power from higher level of the clock tree by removing all clock switching from its down-stream units.

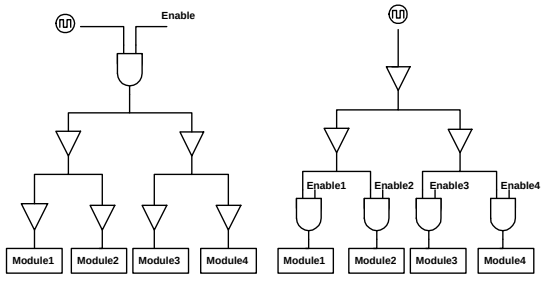

Fig. 4. Clock Gating Mechanisms: coarse grained vs. fine grained.

In this paper, we choose to use *coarse-grained* scheme since it is well suited for PEs that can be deactivated for substantially long period. It also requires less hardware overhead in terms of enabling gates, gating controllers and wires. Since we gate off the PEs for a long period of time, e.g. 1 million cycles, it involves less cycle-to-cycle current variation which could introduce large transient power on chip. Additionally, entering and exiting clock gating mode is very fast, taking only a few clock cycles. This capability allows NPs to be turned on quickly at any time it needs to process traffic spikes.

A. Clock power model

To measure the potential power savings of clock gating, we added the clock power modeling for various components in NePSim. We followed the clock models in [1], [2] and made modifications according to physical features of the IXP1200. The major sources of clock power we considered include:

- Clock distribution tree (wiring) – We implemented 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 [17] (126mm^2 in a 0.28-micron process). We assumed that the H-tree is located at the center of the PEs.
- Clock Generator (Phase-Locked Loop) – Clock generators are usually implemented as phase-locked loops (PLL). We used the power model described in [2] to estimate the power consumption of PLL component-by-component.
- Clock buffers – Clock buffers are inverter chain with increasing gate sizes and the ratio between each stage and number of stages are optimized for speed and minimal skews. We estimated the capacitance load of clock buffers using an analytical model as described in [2].
- Pipeline latches – We use Wattch [1] to estimate the power dissipated by latches in the 5-stage pipeline.
- SRAM array bitline precharge in memory structure – We assume the register files and the control store (where program is stored) use the classic 6-transistor cell and a single precharge transistor per bit line. We obtain the precharge transistor size information from existing cache model in NePSim (from Cacti) and calculate their gate capacitances.
- 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 whole 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 loading. The above clock load in PEs consumes 0.21W, and Figure 5 captures how the different components contribute to the clock power. Here PLL's power is not included in the pie chart, because we assume it's located out of PEs.

Besides the explicit clock power consumed in clock distribution network and precharge gates, there are dynamic logic modules that consume power due to 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 custom 32-bit ALU, which supports binary logic functions (i.e. AND, OR, NOT, XOR), addition and subtraction, in Cadence toolset. For the full adder, we used domino Manchester carry chain, because it's a standard design as used in Alpha 21064 microprocessor [18]. The ALU is composed of 3320 gates in total, including full adder, binary logic, muxes, and input/output drivers. Close

to minimum-size transistors are used except for the carry-generation circuitry. The large devices used here (up to 7.5 λ width) speed up the carry propagation, which is on the critical path. The ALU latency was verified to be within 4.3ns (1 clock cycle) through SpectreS simulations. The average power consumed by an ALU is 0.03W.

- Control Store wordline decoders – Modern caches use dynamic logic for wordline decoding and driving. Hence we treat the capacities of wordline decoder and drivers in control store (instruction memory) as the clock load. They are modeled in Cacti.

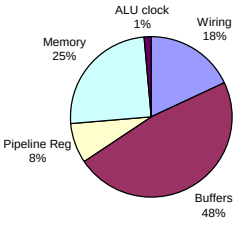

Fig. 5. Total clock effective capacitance breakdown for six PEs in NePSim

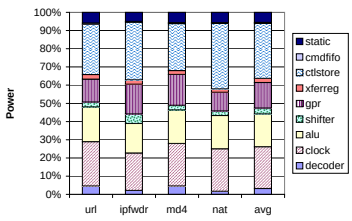

Fig. 6. Microengine power consumption breakdown

Figure 6 demonstrates the power consumed by individual components inside 6 PEs, obtained through the NePSim simulations with unlimited uniform traffic input. We separate the power of wordline decoders from the functional units. We observe that the clock load on average consumes 22% of total PEs' power, while the dynamic logic modules (i.e. ALU, decoders) consume 21% of total PEs' power. During clock gating, we gate off the circuitry in 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. Additionally, no new accesses to the remaining components of clock-gated PE will happen, so some of the functional units accesses which consume the rest 50% dynamic power can be saved.

VI. Experiment Evaluation

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

A. Benchmark applications and input traffic patterns

There are four benchmarks that are currently ported to NePSim. They are *ipfwdr*, *url*, *nat*, and *md4*. The *ipfwdr* implements an IPv4 router which forwards 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 design and experiments.

We evaluate our design with real network packet traces (e.g. Leipzig-I) from NLANR[14]. 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, due to the limited simulation speed of NePSim,it is too expensive to simulate the entire traces of dozens of hours. Since NePSim simulates a multi-core and multi-threaded architecture, it usually takes more than an hour to simulate one second of real world trace. We therefore sample a few seconds of real traffic with different arrival rates as individual inputs to the simulator.

B. Power overhead of the control logic

We extended the NePSim simulator with clock power modeling (discussed in section V), so that we can measure the power savings of clock gating. For the execution units, pipeline latches, 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 power is added.

We also take into account the power overhead associated with the additional control logic in Figure 1 and 3. We included the power overhead of the counters, threshold registers, thread queue and comparators measured both with Cadence and the Wattch model using 0.25μm technology, and found they consume negligible power, i.e. 0.00038 Watts for a 20-bit counter or register, 0.002 Watts for a comparator, 0.0065 Watts for 24 entry thread queue. The extra buffer we added increase 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 ([7]). Additionally, the controller includes a finite state machine (FSM) which decides the PE on/off decision and an adder. 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 total PE power as the overhead of the controller.

C. Experiment results

We experimented with several traces and present the results of Leipzig-I trace because its link speed falls in the capacity of the IXP1200 NP system. Using other traces have similar results. We scan the trace and extract four segments with different packet arrival rates. For each segment of traces, We feed it to the 16 input ports of NePSim. In this way we formed four traces with the overall arrival rates of, from low to high, $\sim$90Mbps, $\sim$180Mbps, $\sim$360Mbps and $\sim$480Mbps respectively. We tested different length of shutdown period P ranging from 125K to 8M cycles and found that it affects little to the results. We thus choose 1M cycles as the shutdown period since it can hide well the longest PE shutdown latency observed (60K cycles for url). The initial th is set to 500K cycles (half of 1M cycles) to represent medium aggressiveness. The threshold adjustment amount $alpha$ is set to 2% of P. The metrics we evaluate are power consumption (in Watts), throughput (in Mbps), and PE utilization.

Figure 7 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 amount decreases because less power saving opportunity can be exploited. At the highest traffic load, power reduction numbers are the lowest, but still there are 17%, 15%, 12%, 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. This implies that more power saving opportunities can be exploited by our scheme. On the other hand, url has the longest processing time, resulting in the shortest thread queue and the least PE shutdown opportunity.

Our clock gating scheme has very little impact on the system throughput as shown in Figure 8. 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 towards 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. This phenomenon is not shown in the graph.

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 Figure 9 under different traffic load. Each sub-figure compares the utilization of the active PEs with and without clock gating (base case). Figure 9 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 360Mbps).

Note that the power saving data we present here is in one second period. The overall power saving on a daily basis is tremendous considering that low traffic period contributes to a large portion of a day.

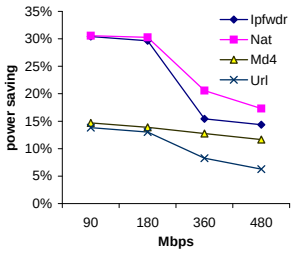

Fig. 7. power saving vs packet arrival rate.

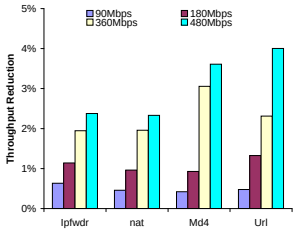

Fig. 8. Throughput reduction vs packet arrival rate.

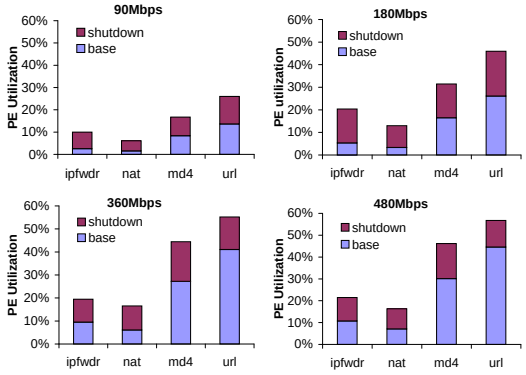

Fig. 9. ME utilization.

VII. Related Work

Recently, power reduction techniques for NPs have appeared at various levels. Luo *et al.* presented how to apply dynamic voltage scaling (DVS) to reduce NP's power. Kaxiras *et al.* proposed IPStash memory architecture to act as a TCAM (used in packet classification and routing) replacement, which significantly reduces the memory set associativity and thus power [5]. Franklin and Wolf developed an analytic performance-power model for typical NPs. They explored the design space of NPs and showed performance-power impact of different systems [3]. Mallik and Memik investigated the relation between transient errors and lowering the voltage for the cache memories of an NP to save power [8]. Lastly, Memik and Mangione-Smith proposed a data filtering engine (DFE) that processes data with low locality before it is placed on the system bus [9]. Significant power saving is achieved for the system bus.

Numerous other work focus on boosting NP performance, i.e. packet processing throughput. Hasan *et al.* proposed a series of techniques to improved packet memory throughput, hence the packet throughput [4]. Sherwood *et al.* proposed a pipelined memory design that emphasizes worst-case throughput over latency, and co-explore architectural tradeoffs [11]. Spalink *et al.* experienced using IXP1200 to build a robust inexpensive router that forwards minimum-sized packets at a high throughput [12].

VIII. Conclusion

We investigated mechanisms to lower the power consumption of NPs under non-saturated incoming traffic rates in routers. We studied thresholds parameters that can be used to turn on/off PEs. We presented scheduling policies to address the problems that occur to PE shutdown. In addition, we described the clock power and clock-gating technique. Our experiments show a significant reduction in power consumption of an NP taking network traffic traces of real-world routers. In the near future we will investigate power saving techniques on other components used by NPs.

References

[1] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *the 27th Annual International Symposium on Computer Architecture*, pp. 83-94, 2000.

[2] D. E. Duarte, N. Vijaykrishnan, M. J. Irwin, "A Clock Power Model to Evaluate Impact of Architectural and Technology Optimizations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 844-855, Vol. 10, Iss. 6, Dec. 2002.

[3] M. Franklin and Tilman Wolf, "Power Considerations in Network Processor Design," *Workshop on Network Processors – NP2, in conjunction with HPCA9*, pp. 10–22, 2003.

[4] J. Hasan, S. Chandra, and T. N. Vijaykumar, "Efficient Use of Memory Bandwidth to Improve Network Processor Throughput," *The 30th International Symposium on Computer Architecture (ISCA)*, pp. 288–299, 2003.

[5] S. Kaxiras and G. Keramindas, "IPStash: a Power-Efficient Memory Architecture for IP-lookup," *The 36th International Symposium on Microarchitecture (MICRO)*, pp. 361, 2003.

[6] H. Li, S. Bhunia, Y. Chen, T.N. Vijaykumar, K. Roy, "Deterministic Clock Gating for Microprocessor Power Reduction," Proceedings of the The Ninth International Symposium on High-Performance Computer Architecture (HPCA'03), pp 113.

[7] Y. Luo, J. Yang, L. Bhuyan, L. Zhao, "NePSim: A Network Processor Simulator with Power Evaluation Framework," *IEEE Micro Special Issue on Network Processors for Future High-End Systems and Applications, Sept/Oct 2004.*

[8] A. Mallik and G. Memik, "A Case for Clumsy Packet Processors," *to appear in the 37th International Symposium on Microarchitecture (MICRO)*, Portland / OR, Dec. 2004

[9] G. Memik and W. H. Mangione-Smith, "Improving Power Efficiency of Multi-Core Network Processors Through Data Filtering," *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 108–116, 2002.

[10] M. D. Powell, S. Yang, B. Falsafi , K. Roy, T. N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 90-95, 2000.

[11] T. Sherwood, G. Varghese, and B. Calder, "A pipelined memory architecture for high throughput network processors," *The 30th Annual International Symposium on Computer Architecture (ISCA)*, pp. 288–299, 2003.

[12] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building a Robust Software-Based Router Using Network Processors," *Symposium on Operating Systems Principles (SOSP)*, pp. 216–229, 2001.

[13] EZchip Technology, *http://www.ezchip.com/*

[14] The NLANR Measurement and Network Analysis, *http://www.nlanr.net/*

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

[16] Intel IXP2XXX Product Line of Network Processors, http://www.intel.com/design/network/products/npfamily/ixp2xxx.htm

[17] T.R. Halfhill, Intel Network Processor Targets Routers, Microprocessor Report, Volume 13, NUMBER 12, September 13, 1999

[18] Wayne Wolf, Modern VLSI Design, System-On-Chip Design, Prentice Hall, 3rd edition.