Power Efficient Scheduling for Network on Chip Applications on Multicore Processor

R. Rubavani  
Department of Electronics and Communication Engineering,  
Vel Tech Dr. RR & Dr. SR Technical University, Avadi, Chennai-600062, Tamil Nadu, India.

S. Saranraj  
Department of Computer Science Engineering,  
Vel Tech Dr. RR & Dr. SR Technical University, Avadi, Chennai-600062, Tamil Nadu, India.

S. Saranya  
Department of Electronics and Communication Engineering,  
Arignar Anna Institute of Science & Technology, Sriperumpudur-602117, Tamil Nadu, India.

R. Ranjani Devi  
Department of Electronics and Communication Engineering,  
Arignar Anna Institute of Science and Technology, Sriperumpudur-602117, Tamil Nadu, India.

Abstract
Task scheduling on multicore systems for optimized power and energy consumption is essential as the trend of utilizing multiple processors is ever increasing. In the design and implementation of today’s multicore/many core chips, power and energy have become increasingly important concerns. In this project, we present priority-based CPU scheduling algorithms that use dynamic performance data to reduce the power consumption by over 20% with significant increase in performance. This algorithm operates at different fixed frequency and uses Linux CPU sets and cores. Many technique like dynamic frequency scaling, lower the core’s frequency during the execution of the task and hence lowering performance. This algorithm match processes to core better suited to execute those processes in an effort to lower the average completion time of all processes in an entire task, thus improving performance. This algorithms use dynamic process priorities as scheduling criteria for reducing power and energy consumption hence indirectly reduces cores temperatures and aids in eliminating the hot spots to ensure overall thermal safety. In this project, we propose an optimal ILP solution to task scheduling for different applications on a multicore system with power and energy constraints. Effectiveness of our techniques is shown in result on different benchmark.

Keywords: Task scheduling, thermal-aware, multicore

Introduction
The explosive growth of network bandwidth and Internet high traffic applications, such as web browsing, online searching, video streaming, and gaming, require orders of magnitude increase in packet processing throughput. The advent of commodity multicore platforms, such as Cavium OCTEON, Ciscos AON, and IBMs BladeCenter, has opened a new era of computing for network applications to take advantage of these low cost machines due to their superiority in performance, availability and programmability. More and more network packet processing systems have been developed on such platforms ranging from general-purpose processors (e.g., Intel’s Xeon and AMD’s Opteron) to network processors (e.g., Intel’s IXP platform) and programmable logic devices (e.g., NetFPGA). To exploit available parallelism for better throughput, network applications running on multicore platforms usually take one of the following three forms: Spatial parallelism, where multiple concurrent packets are processed in different processors independently. Typical examples can be found in work for TCP (Transmission Control Protocol) parallelism, scalable DPI (Deep Packet Inspection) design, flow level packet processing and parallel multimedia transcoding. Temporal parallelism (pipelining), where multiple processors are scheduled into a pipeline to overlap periodic executions from different threads. It has been widely adopted in network processors, including Shangri-La, auto-partitioning, statistical approach and Greedy. Hybrid parallelism integrates both spatial and temporal parallelism to benefit from the advantages of both sides. It forms a parallel pipeline core topology, where each stage contains multiple parallel cores, such as Random and Bipar. Traditional task scheduling schemes, such as list-based scheduling and clustering based scheduling, are capable of reducing program latency by exploiting fine grained task-level
parallelism. However, because they do not apply pipelining, they suffer from significant throughput deterioration when executing periodic packet processing tasks. Papers and have presented some research results on reducing protocol latency for high-speed gateways and telecommunication systems based on hybrid parallelism. Developing a packet processing system that considers both latency and throughput for multicore architectures is both interesting and challenging. Thus, we present a latency and throughput-aware scheduling scheme based on parallel-pipeline topology. Along with increased throughput and reduced latency, however, comes increased power consumption for network applications running on multicore architecture. Collectively, millions of servers in the global network consume a great deal of power. And chip manufactures continue to increase both the number of cores and their frequencies, substantially increasing both dynamic and static power consumption. Higher power consumption increases costs, both directly and indirectly. Energy itself is expected to become more expensive, especially if environmental impacts are factored into consumption. Higher power consumption also increases core temperature, which exponentially increases the cost of cooling and packaging. Higher temperatures also increase indirect and life-cycle costs due to reduced system performance, circuit reliability, and chip lifetime. Therefore, power management is a first-order design issue. As we propose the parallel-pipeline scheduling on task-level, we realize that there has been no existing work considering the power budget issues for it. Previous power-aware algorithms either have not considered latency, or have not explored the parallel pipeline topology for task scheduling. Since power gating cannot be directly applied to task scheduling, we resort to DVFS to integrate power-awareness into parallel pipeline scheduling.

Literature Review
Min Bao, Alexandru Andrei, Petru Eles, and Zebo Peng
Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In Proc. of DATE ’10, 2010 proposes a highly scalable multi-threaded DPI system (L7-filter) for multicore servers. It explores potential connection level parallelism in pattern matching and develops an affinity based scheduler to enhance the scalability of multithreading. Similarly to proposes and implements two new DPI packet scheduling algorithms. One is designed to maximize work balance and the other cache affinity. Their observation confirms that scheduling packets for cache affinity is more important than balancing the workload. Second, proposes an auto-partitioning C compiler to automatically partition a sequential packet processing application into coordinated pipelined parallel subtasks. Their transformation technique ensures that packet processing tasks are balanced among pipeline stages and that data transmission between pipeline stages is minimized. Third, in, they partition network applications into different stages with the consideration of limited instruction memory of the processing elements (PEs).

Frank Bellosa, Simon Kellner, Martin Waitz, and Andreas Weissel. Event-driven energy accounting for dynamic thermal management. In Proc. of COLP ’03, 2003 proposes a global scheduling algorithm for real-time applications to minimize the peak temperature by deriving an ideally preferred speed for each core. In this category, some schemes do no rely on any thermal models proposes proactive power migration to reduce spatial and temporal temperature difference by redistributing the heat generating locations. They consider a predefined migration frequency and evaluate two migration techniques: Cyclic Multiplexing and Global Coolest Replace. In, they present a convex optimization based method that proactively controls the temperature of the cores by applying DVFS depending on current workload and current maximum temperature on the chip. They claim to minimize the power consumption, satisfy application performance constraints and guarantee that core temperatures are below a user defined threshold at all times. In, they propose a low-cost temperature management strategy to reduce the adverse effects of hot spots and temperature variations.

Methodologies
Latency and Throughput Aware Scheduling
In this chapter, we propose LATA, a latency and Throughput-Aware packet processing system for multicore architectures. It adopts hybrid parallelism with parallel pipeline core topology in fine-grained task level to achieve low latency and high throughput. We accomplish the above goal through the following three steps. First, we design a list-based pipeline scheduling algorithm from the task graph. Second, we apply a deterministic search-based refinement process to reduce latency and improve throughput through local adjustment. Third, we devise a cache-aware resource mapping scheme to generate a practical mapping onto a real machine.

To the best of our knowledge, LATA is the first of its kind to consider both latency and throughput in packet processing systems. We implement LATA on an Intel machine with two Quad-Core Xeon E5335 processors and conduct extensive experiments to show its better performance over other systems such as Parallel, Greedy, Random and Bipar. Based on six real packet processing applications chosen from NetBench and PacketBench, LATA exhibits an average of 36. 5% reduction of latency across all applications without substantially degrading the throughput. It shows a maximum of 62. 2% reduction of latency for URL application over Random with comparable throughput performance.

LATA System Design
Figure (a) shows LATA’s system design flow chart. Given an network application, We first generate its corresponding task graph with both computation and communication information. Then, we proceed in a three-step procedure to schedule and map the task graph according to our novel design. Last, we deploy the program onto a real multicore machine to obtain its performance result.
Communication measurement

We cannot accurately calculate the communication time between two cores in a multicore architecture unless we know the exact location of the cores. In LATA design, we use the average communication cost based on data cache access time, as given in Equations 3.1 and 3.2. Commavg means the average communication cost to transfer a unit data set, which can be approximated by system memory latencies (L1, L2 and main memory access time) and program data cache performances (L1 and L2 cache hit rate). Data Size refers to the transferred data set size between two communicating tasks.

\[
Comm = Comm_{avg} \times Data\,Size \quad (3.1)
\]

\[
Comm_{avg} = T_{L1} \times H_{hit_{L1}} + T_{L2} \times (1 - H_{hit_{L1}}) \times H_{hit_{L2}} + T_{MEM} \times (1 - H_{hit_{L1}}) \times (1 - H_{hit_{L2}}) \quad (3.2)
\]

As we know, the throughput can be calculated by the inverse of the longest stage time 1/Tmax in pipelining. Thus, we form our objective function in Equation 3.3, where L is the scheduled latency.

\[
\text{Maximize } T_A = \frac{1}{T_{\text{max}}} \quad \text{(s.t. } L \leq L_0) \quad (3.3)
\]

Proposed Latency Reduction

Latency can be reduced by reducing either computation time or communication time. Because computation dominates the overall execution time for most packet processing applications running on multicore architectures, we prioritize computation reduction in designing LATA. Hence, LATA first applies latency hiding to reduce computation time.

Then, CCP elimination and CCP reduction are used to reduce communication time. Computation reduction: We defined a critical node as the node in a pipeline stage which dominates the computation time. Then, Latency hiding can be defined as a technique that places a critical node from one stage to one of its adjacent stages without violating dependencies, so that its computation time is shadowed by the other critical node in the new stage. Backward hiding (BaH) refers to placing a critical node into its precedent stage. Forward hiding (FoH) refers to placing a critical node into its following stage.

Performance evaluation

Figures (c) and (d) show the latency and throughput for six applications by LATA, Parallel and List. We observe that Parallel suffers from high latency due to its sequential execution of tasks. Compared with Parallel, LATA reduces the latency by an average of 34.2%.
Particularly, for URL, LATA achieves the maximal latency reduction of 62.2%. In addition, LATA’s throughput is close to that of Parallel in spite of the 75% latency constraint. This is because LATA is capable of optimizing its parallel pipeline core topology to produce good throughput. With respect to List, which is designed to produce the lowest latency, LATA actually matches its latency performance in most cases by aggressively exploiting task-level parallelism. Furthermore, LATA outperforms List in throughput by an average of 41.0% and a maximum of 56.7% for Route.

Power-Aware Parallel-Pipeline Scheduling Algorithm

We introduce the novel parallel-pipeline scheduling on task-level for network applications that can attain high throughput under given latency constraints. In this chapter, we address the power budget issue for this scheduling paradigm for network packet processing. We aim at optimizing both throughput and latency under given power budget by appropriately applying per-core DVFS. We propose a three-step solution to achieve our goal.

A three-step recursive algorithm:

STEP 1: In the first step, we reduce the power without compromising throughput or latency by keeping the pipeline stage time $T_i$, $i = 1, 2... S$ unchanged. We define a critical node as the node in a pipeline stage that dominates the computation time. Therefore, the computation time of a critical node is equal to the pipeline stage time ($t_i = T_i$). For each stage $S_i$, we increase the computation time of non-critical nodes in that stage to the length of $T_i$. Since all stage times remain the same, the throughput and the latency will also keep unchanged during this step.

STEP 2: In the second step, we reduce the power with throughput unchanged and minimal latency increase. This is achieved by keeping the longest stage time $T_{max}$ unchanged while we increase the stage time of other stages. We denote the stage with $T_{max}$ as the bottleneck stage in the pipeline. Thus, all other stages are non-bottleneck stages. We define $\Delta T$ as the shortest time period by which we can increase the latency. To minimize the latency increase, we iteratively increase the latency by $\Delta T$ until the power budget is satisfied or all the stages reach $T_{max}$. If the former comes true, the algorithm returns and the resulting scheduling guarantees the minimal latency increase, which will be proved shortly. Otherwise, if the latter comes true, we proceed to step three.

STEP 3: In the third step, we reduce the power by minimizing both the throughput and the latency performance loss. Remember that after step two, every stage has the same stage time $T_{max}$. Following the same rule of choosing a candidate stage in step two, we optimally choose a stage to further increase its stage time by $\Delta T$. Since the original $T_{max}$ is increased, the throughput is compromised accordingly.
Consider that task $T$ consists of $C$ clock cycles on processor $P$, which runs at voltage $V$ and frequency $f$. We assume that $C$ does not change with different $V$ and $f$. For a given voltage $V$, processor $P$ has an average power consumption $Pow$. It is known that processor power consumption is dominated by dynamic power dissipation given by:

$$Pow = Ka \cdot f \cdot V^2,$$

Where $Ka$ is a task / processor dependent factor determined by the switched capacitance.

The energy consumed by executing task $T$ on processor $P$ is computed as:

$$E = C \cdot Powf,$$

We can rewrite it as:

$$E = C \cdot Ef, \quad V = C \cdot Ka \cdot V^2,$$

where $Ef$, $V$ is the average cycle energy. From this we can see that lowering the voltage would yield a drastic decrease in energy consumption. The frequency $f$ is almost linearly related to the voltage:

$$f = Kb \cdot (V - VT)^2,$$

Where $VT$ is the threshold voltage and $Kb$ is a constant. For a sufficiently small threshold voltage, the frequency is approximated to $Kb$.

However, this algorithm is able to guarantee a minimal performance loss in this scenario. The only difference is that we need to substitute $Ti$ with $Tmax$ in the equation. The proof of optimality is in line with that in step two, where the minimal time period increment guarantees that when we satisfy the power budget constraint, the performance loss is minimal.

**Algorithm for power aware parallel pipeline scheduling algorithm**

1. if $POW \leq POW_{budget}$ then return
2. for each stage $Si$ do /* Step 1 */
3. for each parallel task $nj$ do
4. $tj \leftarrow Ti$
5. if $POW \leq POW_{budget}$ then return
6. while $Tmax$ unchanged do /* Step 2 */
7. for each stage $Si$ except $Tmax$ stage do
8. calculate $\Delta Pi$ according to Equation 9
9. choose stage $Si$ with $\max\{\Delta Pi\}$
10. for each parallel task $nj$ in $Si$ do
11. $tj \leftarrow tj + \Delta T$
12. update $POW$
13. if $POW \leq POW_{budget}$ then return
14. while $POW > POW_{budget}$ do /* Step 3 */
15. for each stage $Si$ do
16. calculate $\Delta Pi$ according to Equation 9
17. choose stage $Si$ with $\max\{\Delta Pi\}$
18. for each parallel task $nj$ in $Si$ do
19. $t \leftarrow t + \Delta T$
20. update $POW$ and $Tmax$
21. if $POW \leq POW_{budget}$ then return else goto 6

**Experimental Results**

In this section we present power and thermal-aware scheduling results for different benchmarks from the embedded system synthesis benchmarks suite or generated using the MATLAB tool. The ILP solutions were generated using OPENMP. The simulations were performed on a dual core system of 3 GHZ processor. As the power information of the processing elements in these benchmarks is not available, approximation values based on the internal structure of each core is used. Based on these approximated values, the power consumption of each core is adjusted by replacing by an estimated number of gates in the module.
Figure (i): Task scheduling

Figure (j): Task Allocation

Figure (k): Task Priority

Figure (l): Task Queue Lengths

Figure (m): Task Identification
Conclusion
The multicore architecture has prevailed in every aspect of computing platforms including desktops, servers, and embedded systems. The superiority of multicore systems comes from high performance, low cost, and good programmability. Accordingly, in the domain of network applications, we see a shift from single-core system, to multicore servers to accommodate high traffic volume with computationally intensive applications. Along with increased throughput, however, comes significantly increased power consumption. Higher power consumption also increases core temperature, which exponentially increases the cost of cooling and packaging, as well incurs indirect and life-cycle costs due to reduced system performance, circuit reliability, and chip lifetime. Therefore, it is critical to run such applications in a power efficient manner, which naturally gives rise to research on “intelligent” scheduling. In this report, we focused on power-efficient scheduling for network applications on multicore architecture. Our goal is to improve the performance of network applications not only in throughput, but also in latency, power, energy, and temperature.

Recommendations
First, this project only focused on single application, but concurrent execution of multiple applications should be studied in future. It will be challenging to extend the single application techniques to multiple applications with minimal changes. Second, while we are concerned with metrics such as throughput, power and temperature, we have not yet considered quality of service (QoS) for different packet streams. QoS should be considered when developing scheduling schemes for network applications.

References

Figure (n): Task Execution Time