A Novel Approach in Scheduling Of the Real- Time Tasks In Heterogeneous Multicore Processor with Fuzzy Logic Technique For Micro-grid Power Management

Lavanya Dhanesh[1], P. Murugesan[2]
[1]Sathyabama University, Chennai, India
[2]S.A. Engineering College, Chennai, India

Article Info

Article history:

Received Jul 10, 2017
Revised Aug 28, 2017
Accepted Jan 1, 2018

Keyword:

Deadline
Execution time
Fuzzy logic
Micro-grid
Multicore
Priority
Real –Time tasks
Scheduling

ABSTRACT

Scheduling of tasks based on real time requirement is a major issue in the heterogeneous multicore systems for micro-grid power management. Heterogeneous multicore processor schedules the serial tasks in the high performance core and parallel tasks are executed on the low performance cores. The aim of this paper is to implement a scheduling algorithm based on fuzzy logic for heterogeneous multicore processor for effective micro-grid application. Real – time tasks generally have different execution time and dead line. The main idea is to use two fuzzy logic based scheduling algorithm, first is to assign priority based on execution time and deadline of the task. Second , the task which has assigned higher priority get allotted for execution in high performance core and remaining tasks which are assigned low priority get allotted in low performance cores. The main objective of this scheduling algorithm is to increase the throughput and to improve CPU utilization there by reducing the overall power consumption of the micro-grid power management systems. Test cases with different task execution time and deadline were generated to evaluate the algorithms using MATLAB software.

Corresponding Author:

Lavanya Dhanesh,
Research Scholar, Sathyabama University,
Chennai, India.
Email: lavanyadhanesh2007@gmail.com

1. 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 Caviums OCTEON[1], Ciscos AON , and IBMs Blade Center[2,3] , 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.,Net FPGA) [4,5]. 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 [6,7].

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 [8,9,10]. The Performance Analysis of Preemptive Based Uniprocessor Scheduling was discussed for the real time task [11] . The Processor Speed Control for the Real-Time Systems for power reduction was analysed [12].But these scheduling schemes do not apply pipelining process and they suffer from significant throughput deterioration when executing periodic packet processing tasks. Many researchers presented the 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 [13,14]. 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. The 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 life-time. 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.

2. METHODOLOGIES

2.1 Latency And Throughput Aware Scheduling (LATA)

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 Net Bench and Packet Bench, 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.

2.2 Lata System Design

In the LATA’s system design , 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.

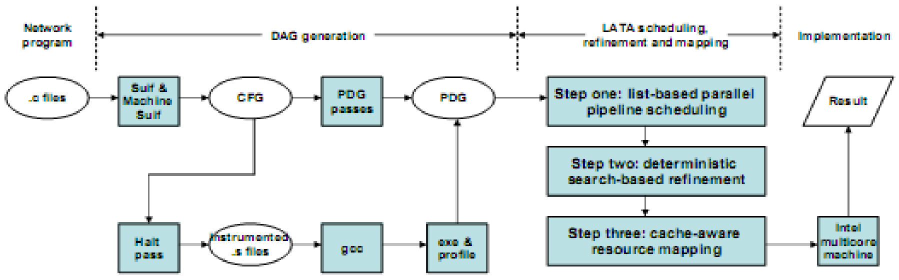

Network program
DAG generation
LATA scheduling, refinement and mapping
Implementation
.c files
Suif & Machine Suif
CFG
PDG passes
PDG
Step one: list-based parallel pipeline scheduling
Step two: deterministic search-based refinement
Step three: cache-aware resource mapping
Result
Halt pass
Instrumented .s files
gcc
exe & profile
Intel multicore machine

Figure 1. LATA system design flow chart.

2.3 Communication measurement

The communication time cannot be accurately measured 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 1 and 2. Comm avg 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.

$$\mathrm{Comm}_{\mathrm{avg}=}\ \mathrm{Comm}_{\mathrm{avg}} . \mathrm{Data\ Size} \tag{1}$$

$$\mathrm{Comm}_{\mathrm{avg}} = ((\mathrm{TL}_1 . \mathrm{Hit\ L}_1) + (\mathrm{TL}_2 . (1 - \mathrm{Hit\ L}_1)). \mathrm{HitL}_2 + \mathrm{T}_{\mathrm{mem}} . \big((1 - \mathrm{Hit\ L}_1). (1 - \mathrm{Hit\ L}_2)\big)\big) \tag{2}$$

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, where L is the scheduled latency.

2.4 Problem Statement

The latency can be defined as the schedule length of a program and throughput as the system throughput. The problem statement is: given the latency constraint L0, schedule a packet processing program in parallel pipeline core topology so as to maximize the throughput Th. The aim is to rearrange the tasks into the parallel pipeline task graph as shown in Figure 3, so that the total execution time T1 + T2 + T3 + T4 is minimized while maintaining the throughput as high as possible. 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, where L is the scheduled latency.

$$\mathrm{Maximize\ Th} = \frac{1}{\mathrm{T}_{\mathrm{max}}} (\mathrm{s.t.at\ L} \leq \mathrm{L}_0) \tag{3}$$

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 which is shown in Figure 2 .

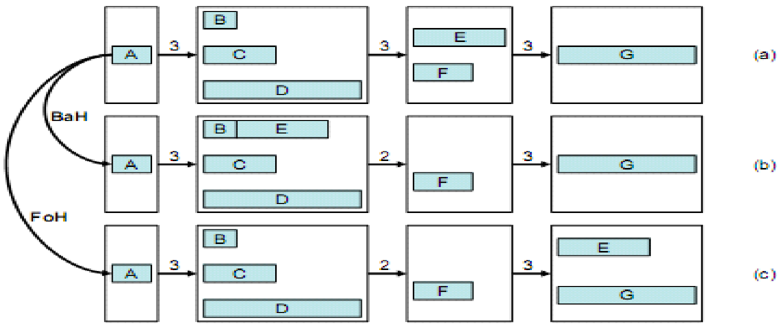
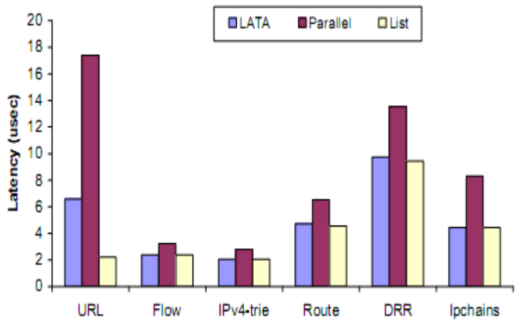


Figure 2. Latency hiding on node E.

4 PERFORMANCE EVALUATION

The latency and throughput for six applications by LATA, Parallel and List are shown in the Figures 3 and 4. 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%.



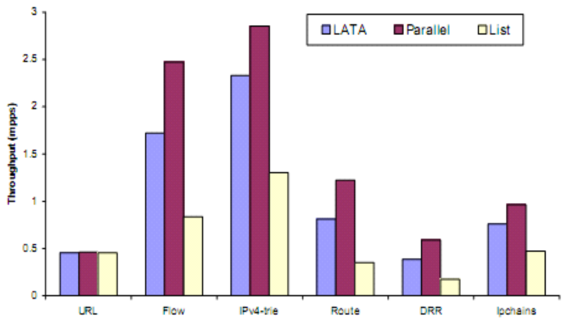


Figure 3. Latency of six applications by LATA, Parallel and List

Figure 4. Throughput of six applications by LATA, Parallel and List

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.

5. 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.

5.1 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 Ti, 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 (ti = Ti). For each stage Si, we increase the computation time of non-critical nodes in that stage to the length of Ti. Since all stage times remain the same, the throughput and the latency will also keep unchanged during this step which is depicted in the Figure 5.

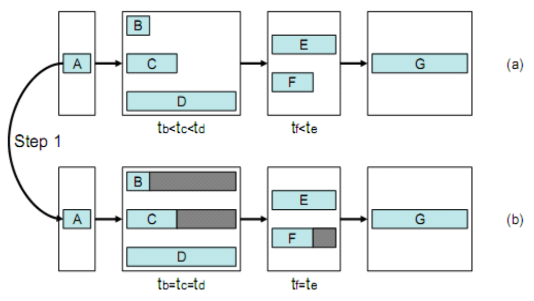


Figure 5. Illustration of the first step of the algorithm

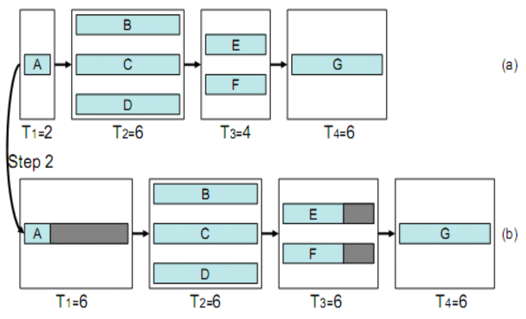


Figure 6. Illustration of the second step of the algorithm

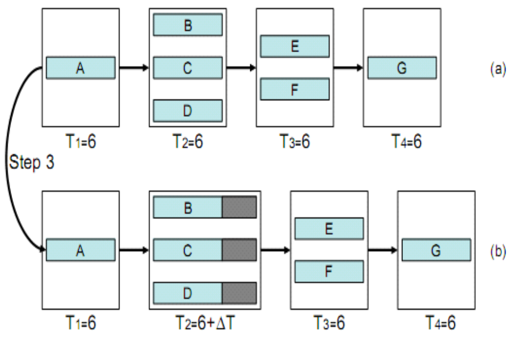


Figure 7. Illustration of the third step of the algorithm

STEP 2: In the second step, we reduce the power with throughput unchanged and minimal latency increase which is shown in Figure 6 . This is achieved by keeping the longest stage time Tmax unchanged while we increase the stage time of other stages. We denote the stage with Tmax as the bottleneck stage in the pipeline. Thus, all other stages are non-bottleneck stages. We define ΔT as the shortest time period by which we can increase the latency. To minimize the latency increase, we iteratively increase the latency by ΔT until the power budget is satisfied or all the stages reach Tmax. 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. After step two, every stage has the same stage time Tmax. Following the same rule of choosing a candidate stage in step two, we optimally choose a stage to further increase its stage time by ΔT. Since the original Tmax is increased, the throughput is compromised accordingly which is shown in the Figure 7 .

6. POWER MODEL

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:

$$\mathrm{Pow} = \mathrm{K_a} \,.\, \mathrm{f.} \, \mathrm{V_2} \qquad (4)$$

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:

$$E = C . \text{Pow}\, f \tag{5}$$

We can rewrite it as:

$$E = C. E_f \tag{6}$$

$$V = C. K_a . V_2 \tag{7}$$

Where E_f, 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:

$$f = K_b . (V - V_T)(V - V_T)^2 . V \tag{8}$$

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

However, our algorithm is able to guarantee a minimal performance loss in this scenario. 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 which is shown in the Figure 8.

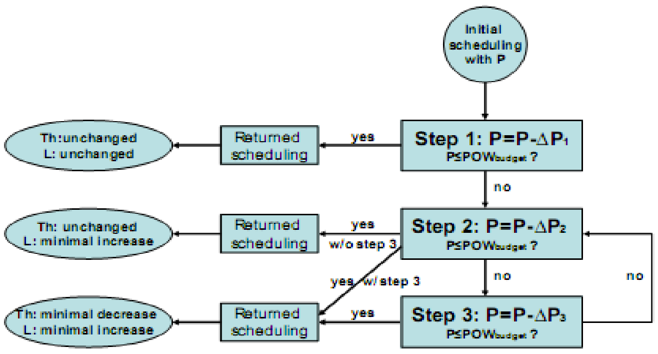

Initial scheduling with P
Step 1: P=P-ΔP₁ P≤POWbudget ?
yes
Returned scheduling
Th:unchanged L: unchanged
no
Step 2: P=P-ΔP₂ P≤POWbudget ?
yes w/o step 3
Returned scheduling
Th: unchanged L: minimal increase
yes w/ step 3
no
no
Step 3: P=P-ΔP₃ P≤POWbudget ?
yes
Returned scheduling
Th: minimal decrease L: minimal increase

Figure 8. The power-aware parallel-pipeline scheduling algorithm.

In our system, since energy increases when performance degrades (i.e., a longer execution time), we assign a higher weight to performance by setting αc = 0.15 αm. Specifically, the lost factors for GPU cores and memory are calculated as:

$$l\_C_i^t = \alpha_c . C_{ie}^t + (1 - \alpha_c)\,(l\_C_{ip}^t) \tag{9}$$

Then we combined core and memory lost functions together by a factor, which balances core impact and memory impact in influencing system performance and energy.

$$\text{Total Loss}_{ij}^{t} = \emptyset . l\_\ C_i^t + (1 - \emptyset) . l\_\ m_j^t \tag{10}$$

Shows how total lost function is obtained. For different CPU-GPU systems, by tuning φ value the system can achieve balance between core and memory influence. In our hardware tested, 0.33 is the value reflects system characteristic derived from experiments. Based on the total loss, the weights used in the frequency scaling algorithm can be updated as follows.

$$\text{weight}_{ij}^{(t+1)} = \left((\text{weight}_{ij}^t)(1 - (1 - \beta). \text{Total Loss}_{ij}^{t}\right) \tag{11}$$

The algorithm will be more robust to system noise. A smaller β gives more weight on loss factor of current time interval. The algorithm will respond to workload change in a short time. In our experiment, we select β = 0.2 to filter out system noise with quick workload change response. Almond the entire N × M weights (assume we have N core frequency levels and M memory frequency levels), the highest one is selected and its corresponding core and memory frequencies are enforced in the next period.

6. EXPERIMENTAL RESULTS

The power and thermal-aware scheduling results for different benchmarks from the embedded system synthesis benchmarks suite or generated using the MATLAB tool. The process of Task scheduling is shown in the Figure 9. The ILP solutions were generated using OPENMP. The simulations were performed on a dual core system of 3 GHZ processor. The output execution and the results using the fuzzy logic techniques are shown in the Figures 10,11and12.The execution time and the relative deadline are adjusted to get the desired priority of the tasks. The modified priority order based on the fuzzy logic output is shown in the Figure 13. The corresponding waveforms are depicted in the Figure 14. 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. The hardware prototype model for the Real - Time task scheduling in Heterogeneous Multicore Processor for Microgrid Power Management is shown in the Figure15.

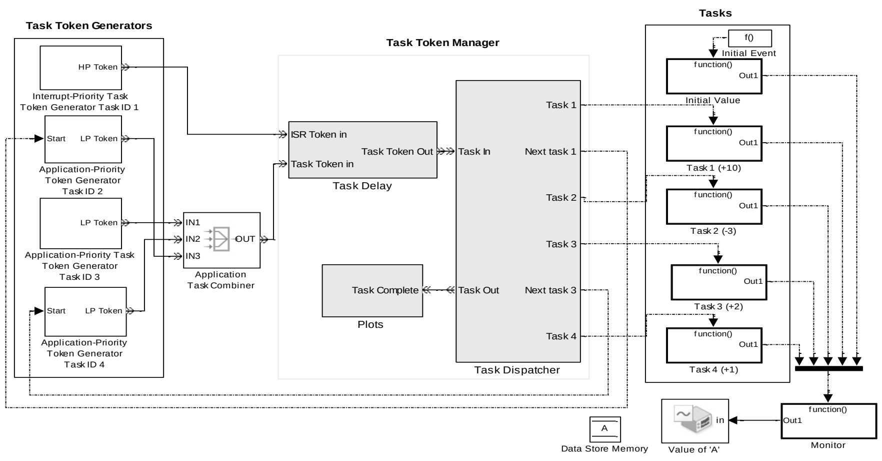
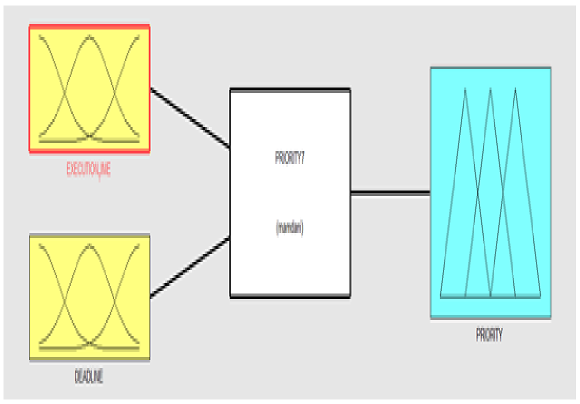


Figure 9.Task scheduling process



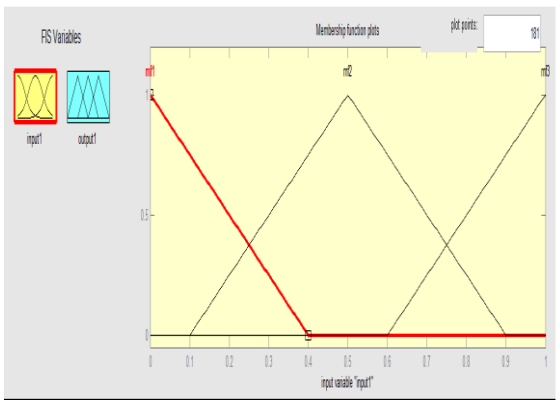


Figure 10. Executing output according to Rules

Figure 11. Input Waveform

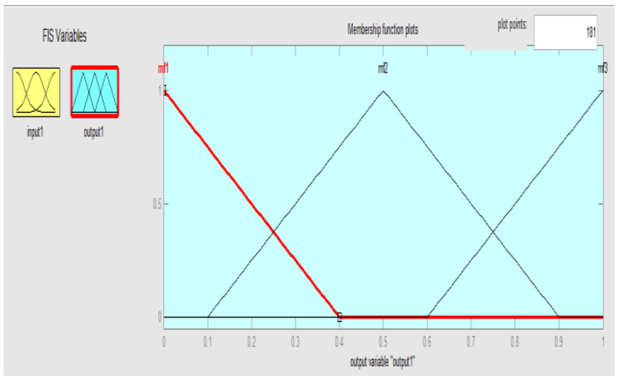


Figure 12 . Output Waveforms

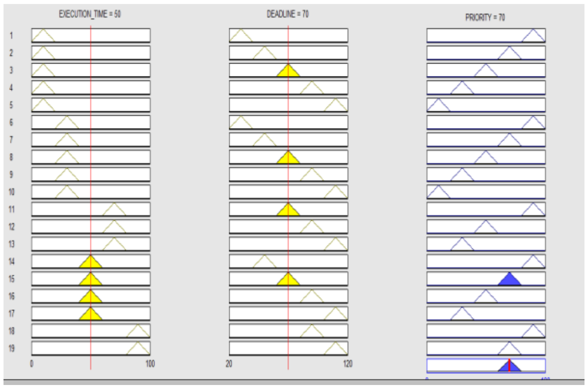


Figure 13. Adjusting Execution and Deadline for Getting Priority

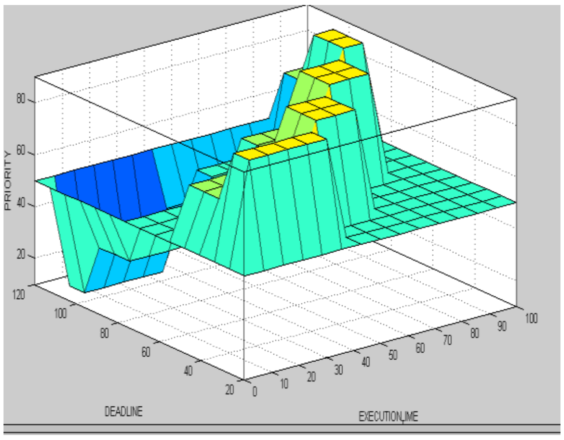


Figure 14. Output Waveform

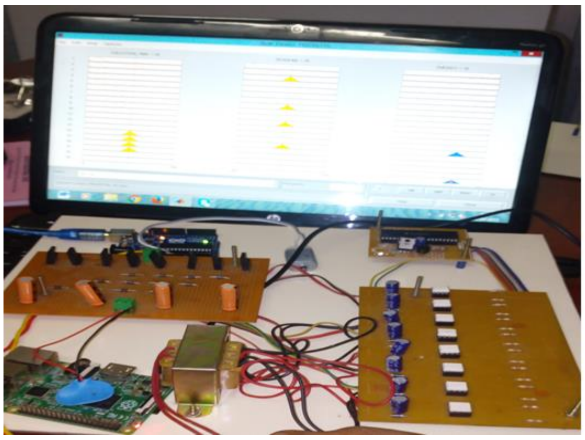


Figure 15. Prototype Hardware model

7. CONCLUSION

Current research on GPU-CPU systems focuses mainly on the performance aspects, while the energy efficiency of such systems receives much less attention. There are few existing studies that start to lower the energy consumption of GPU-CPU architectures, but they address either GPU or CPU in an isolated manner and thus cannot achieve maximized energy savings. In this paper, we have presented Green GPU, a holistic energy management framework for GPU-CPU heterogeneous architectures. Our solution features a two-tier design. In the first tier, Green GPU dynamically splits and distributes workloads to GPU and CPU based on the workload characteristics, such that both sides can finish approximately at the same time. As a result, the energy wasted on staying idle and waiting for the slower side to finish is minimized. In the second tier, Green GPU dynamically throttles the frequencies of GPU cores and memory in a coordinated manner, based on their utilizations, for maximized energy savings with only marginal performance degradation. Likewise, the frequency and voltage of the CPU are scaled similarly. We implement Green GPU using the CUDA framework on a real physical tested with Nvidia GeForce GPUs and AMD Phenom II CPUs. Experiment results with standard Rodinia benchmarks show that Green GPU achieves 21.04% average energy savings and outperform several well-designed baselines.

REFERENCES

[1] Cavium octeon processor family.http://www.caviumnetworks.com/OCTEONMIPS64.html.
[2] IBM BladeCenter System. http://www-03.ibm.com/systems/bladecenter.
[3] IBM POWER7 Systems. http://www-03.ibm.com/systems/power.
[4] Intel ixp2xxx product line of network processors. http://intel.com/design/network/products/npfamily/index.html.
[5] Intel xeon machine. http://www.Intel.com/Xeon.

[6] Chen, D. Zhang and Z. Wang, "Research of the heterogeneous multi-Core processor architecture design [J]", *Computer Engineering and Science*, vol. 33, no. 12, (2011), pp. 27-36.

[7] S K. Baruah, "*Partitioning real-time tasks among heterogeneous multiprocessors*", In: Proc. of the 2004 International Conference on Parallel Processing, ICPP, Toronto, Canada, (2004), pp. 467-474.

[8] R. Li, Y. Liu and X. Cheng, "A Survey of task scheduling research progress on multiprocessor", *Journal of Computer Research and Development*, vol. 45, no. 9, (2008), pp. 1620-1629.

[9] J. Li and S. Jin, "Research on static task scheduling strategy based on heterogeneous multi-core processors [J]", *Computer Engineering and Design*, vol. 34, no. 1, (2013), pp. 178-184.

[10] J. Jiang, "Research on embedded software key issues of heterogeneous multi-core processor [D]", Chong Qing University, (2011).

[11] M Shanmugasundaram , R. Kumar and Harish M Kittur "Performance Analysis of Preemptive Based Uniprocessor Scheduling" in the *International Journal of Electrical and Computer Engineering (IJECE)* Vol. 6, No. 4, August 2016, pp. 1489 -1498

[12] Medhat H Awadalla "Processor Speed Control for Power Reduction of Real-Time Systems" *International Journal of Electrical and Computer Engineering (IJECE)* Vol. 5, No. 4, August 2015, pp. 701-713.

[13] S. Albers, "Energy-efficient algorithms," Commun. ACM, vol. 53, no. 5, pp. 86–96, May 2010. [Online]. Available: http://doi.acm.org/10.1145/1735223.1735245.

[14] P. Cichowski, J. Keller, and C. Kessler, "*Energy-efficient mapping of task collections onto manycore processors*," in Proc. 5th Swedish Workshop on Mulitcore Computing (MCC 2012), 2012.

BIOGRAPHIES OF AUTHORS

Mrs Lavanya Dhanesh received her B.E. degree in Electrical and Electronics Engineering from Bharathiyar University at 2002. She completed her M.E. in Embedded System Design from Anna University ,Chennai at 2009. She is a Research Scholar at Sathyabama University and currently working at Panimalar Institute of Technology,Chennai

Dr P.Murugesan has done his specialization in Power Systems. He is currently working as a professor /EEE at S.A.Engineering college,Chennai. He gives research ideas and much interested in the field of low-power computing, fault tolerance and real-time embedded systems.