Scheduling on Heterogeneous Multi-core Processors Using Stable Matching Algorithm

Muhammad Rehman Zafar
Department of Computer Science
Bahria University
Islamabad, Pakistan

Muhammad Asfand-e-Yar
Department of Computer Science
Bahria University
Islamabad, Pakistan

***Abstract*—Heterogeneous Multi-core Processors (HMP) are better to schedule jobs as compare to homogenous multi-core processors. There are two main factors associated while analyzing both architectures i.e. performance and power consumption. HMP incorporates cores of various types or complexities in a solitary chip. Hence, HMP is capable to address both throughput and productivity for different workloads by coordinating execution assets to the needs of every application. The primary objective of this study is to improve the dynamic selection of the processor core to fulfill the power and performance requirements using a task scheduler. In the proposed solution, there will be dynamic priority lists for tasks and available cores. The tasks to core mapping is performed on the basis of priorities of the tasks and cores.**

***Index Terms*—Heterogeneous, Performance, Scheduling, Multi-core processors, Stable matching**

I. Introduction

HMP is becoming mainstream because it has potential to reduce the power consumption and improve the performance than homogeneous core processors. In HMP architecture, individual cores have different computational capabilities as shown in Figure 1. Since, HMP architecture consists of a combination of small and big cores, it can perform better in larger computations [1]. HMP architectures opens up new challenges and possibilities for thread scheduling, load balancing and energy management. The performance and energy efficiency can be achived with HMP by allowing each job to run on core type which suits the most [2]. Furthermore, HMP can adequately reduce processor power utilization and can significantly build the performance and speed of execution. The HMP decreases the frequency of the processors which reduce the temperature of the system. In these processors, the amount of parallelism increased because of simultaneous execution of instructions on individual cores. The MB scheduler can dynamically select the relevant core to fulfill the power and performance requirements [3].

HMP can be characterized into two groups i.e. "performance asymmetry" and "functional asymmetry". In functional asymmetry, architecture cores have distinctive or overlapping instruction sets while in performance asymmetry, architecture cores vary in performance because of the difference in frequency and architecture [4]. Additionally, HMP architecture is more attractive and an alternative design as compare to homogeneous designs. HMP has a unique benefit in maximizing both execution performance and throughput. Although, the existing multi-core processors are mostly homogeneous which leads to an unpreventable problem i.e. reproducing smaller cores losses the throughput of the high-complexity single-threaded applications, though duplicating bigger cores yields the execution proficiency of the low-complexity low-priority threads. However, HMP incorporates cores of various types or complexities in a solitary chip, and hence it is capable to address both throughput and productivity for different workloads by coordinating execution assets to every application's needs [5].

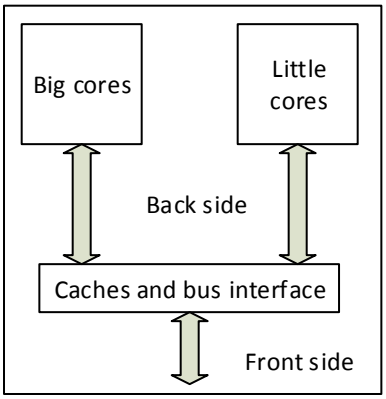

Fig. 1. HMP architecture

In future, many core and multi-core processors will comprise of heterogeneous cores that might expose a typical Instruction Set Architecture (ISA) but vary in features e.g. performance, size and energy utilization. A solitary processor will contain numerous small cores and a few bigger complex cores. Simple cores will remain scalar in order and may have a small cache and lower clock frequency. Complex cores will be super-scalar and might be outfitted with high performance and consume more power. Heterogeneous architectures are inspired by their capability to accomplish a higher execution per Watt than homogeneous architectures [6].

To understand this architecture, operating systems must be aware with heterogeneity. That is, it must allocate applications to keep running on proper cores. Consider a workload comprising of a logical application portrayed by Instruction Level Parallelism (ILP), and a memory-bound employment, for example, preparing exchange in a database. The logical application will be executing essentially quicker on an complex cores, though the database application may indicate equivalent execution on both sorts of cores. In this case, assigning tasks to cores that are fitting for them will accomplish huge power saving.

Having the capacity to make intelligent decisions at run time is the objective of heterogeneity scheduling. While heterogeneity scheduling algorithms were proposed in the past which were focused at small scale multi-core frameworks. If increase the number of chips, dynamic checking might become excessively tedious and impractical. HMP gives another way of various computing abilities. When the processor performance and speed increases there are many challenges may rise e.g. heat dissipation and power utilization.

The primary focus of this paper is to improve the dynamic selection of the most appropriate processor core to fulfill the power and performance requirements using a task scheduler.

II. Literature Review

Previous work on HMP architecture insights the importance of heterogeneity for both performance and power efficiency. In a study [7], authors proposed and evaluated a single ISA HMP architectures to reduce processor power utilization first time. In this paper, authors exploited the time varying behavior of applications and power consumption using thread migration approach at run time. The design of this approach integrates heterogeneous cores indicating different facts in the performance/power design space during the execution of an application. The system software chooses the most appropriate core dynamically to meet the particular performance and power needs.

In another relevant study [8], authors proposed scheduler for different applications to schedule on different cores in single ISA HMP designs. In this study, authors have demonstrated that by scheduling applications on more power efficient HMP, throughput for static workloads will be increased. On the other hand, it reduces the response time for dynamic applications. Furthermore in [9], explored the standards to design single-ISA HMP. In this study, authors just concentrated on framework throughput and do not consider per-program execution.

Furthermore in [4], authors proposed a bias scheduling approach for performance asymmetric heterogeneous with different micro architecture cores. They identified the key metrics that distinguish the possible benefits of scheduling an application on a big core rather than on a small core and according to the core type that outfits the resource requirements of the application. Bias scheduler is very flexible and can be implemented on any scheduler. In this paper, authors have implemented bias scheduler on the top of Linux Scheduler and concluded that performance can be improved significantly. Bias scheduler monitors the application at run time and match the threads according to the core types to increase the throughput of the system.

Moreover in [6], authors proposed an approach that does not depend on dynamic performance monitoring. In this approach, the required information to make an appropriate decision for core assignments is provided with the application itself. The provided information is used as an architectural signature of the running application which composed of certain architecture independent features. These signatures can be generated offline and embedded into the binary of an application. Further, authors demonstrated prediction the sensitivity of an application and evaluated a scheduler model that uses these architectural signatures for scheduling on a heterogeneous system with various clock frequencies cores.

Further, authors in [2] proposed a lucky scheduling algorithm to schedule threads on HMP for energy efficiency and performance boost up. In lucky scheduling algorithm each task receives a dynamic number of tickets. The tickets will determine the switching of running tasks between small and big cores. Moreover, the idea of lucky scheduling algorithm is derived from a lottery scheduler system [10]. Additionally, the lottery scheduler system is used to select the lottery tickets dynamically.

In another relevant studies [11], [12], sampling based scheduling is applied using direct approach to determine the best scheduling policy on a HMP. In sampling based scheduling, samples of different workload are mapping to core dynamically at runtime. After that best mapping is selecting with respect to performance. Further, sampling based scheduling periodically migrate the workload between different cores. By using sampling based approaches performance is improving while, on the other hand it also introduces migration overhead. Moreover, migration overhead is directly proportional to number of cores.

To overcome the migration between different type of cores, authors proposed a Performance Impact Estimation (PIE) approach [13]. In this study PIE mechanism is used to predict best performing workload to core mapping. PIE collects Cycles per Instruction (CPI) stack, ILP, number of misses, latency per miss and the number of simultaneously outstanding misses to make prediction. On the basis of collected information it estimates the performance if the workload is to run on a different core types. Dynamic PIE performs scheduling at runtime.

III. Proposed Methodology

The proposed solution is using stable matching algorithm to assign the tasks to suitable cores. In this approach both cores and tasks have their own priority lists as shown in Table I and Table II. The proposed scheduler keeps track of the priorities and availability of cores. At the time of job mapping to core, the algorithm selects the best possible pair of task and core. Steps of the proposed algorithm is given below in Figure 2 and architecture diagram in Figure 3. The algorithm takes priority lists as an input in step 1. In step 2, set cores free to fetch

TABLE I
PRIORITY LIST OF EACH TASK

	Task 1	Task 2	Task 3
Priority 1	Core - SISD	Core - MIMD	Core - SIMD
Priority 2	Core - SIMD	Core - MISD	Core - SISD
Priority 3	Core - MIMD	Core - SISD	Core - MISD

TABLE II
PRIORITY LIST OF EACH CORE

	Core 1	Core 2	Core 3
Priority 1	Task - SISD	Task - MIMD	Task - SIMD
Priority 2	Task - SIMD	Task - MISD	Task - SISD
Priority 3	Task - MIMD	Task - SISD	Task - MISD

```
1: Input: priority lists of tasks and cores
2: Initialize each core to be free.
3: while (some core is free and hasn't assigned to every task)
   {
4:      Choose such a core c
5:      t = 1st core on c's list to whom c has not yet assigned
6:      if (t is not  assigned)
7:              choose c and t to be assigned
8:      else if (t prefers c to its assigned task c' and c' is free )
9:              choose c and t to be assigned, and c' to be free
10      else
11              t rejects c
   }
12: Output: stable tasks to cores mapping
```

Fig. 2. Stable matching algorithm to map tasks to cores

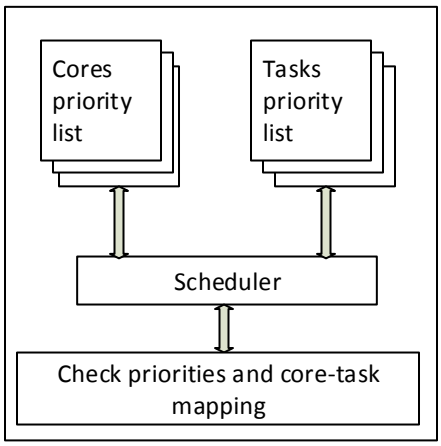

Fig. 3. Proposed scheduler architecture

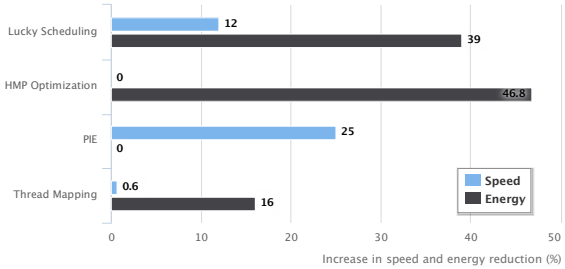

Fig. 4. Dynamic approaches comparison

jobs according to their priorities. The while loop in step 3 will execute until until all cores and task mapped. Further, in step 4, select a core and fetch a task for execution in step 5. In step 6, 8, 10 if else conditions are checked. If condition is true and task is not assigned to other core, the task t will be assigned to that core in step 7. Otherwise, task is not executing and prefers other core c on assigned core c', task t will be assigned to core c and c' core will set free in step 9. If priorities of both task and core do not match task will rejects the core.

IV. PERFORMANCE EVALUATION OF EXISTING SYSTEMS

The algorithms discussed in Section II have demonstrated that the performance on HMP is increased but they have a number of drawbacks. If the number of core types is large, monitoring of a thread or subsets of threads become infeasible. The operating system needs to track a lot of information and performance will be affected. Also, the amount of time for monitoring the thread will increase. Table III shows the comparison of different schedulers and experimental results of existing approaches and Figure 4 shows the comparison of different dynamic approaches. In [2], authors have used one core of speed 3.2 GHz and three cores of speed 0.8 GHz. The L3 cache of size 6MB shared between cores. By using luck scheduler algorithms authors achieved 12% inc rease in speed and energy consumption is reduced by 39%. Further, in [6], authors used static approach to schedule jobs on different cores. In this study authors have categorized the experiments into two groups i.e "high heterogeneous workload" and "typical workload". In high heterogeneous workload, authors used two cores of speed 2.0 GHz and other two cores of speed 3.0 GHz. With this approach, throughput of the HMP is increased by 4.7%. While in typical workload, authors used all four cores of different speed such as 2.0 GHz, 2.33 GHz, 2.67 GHz and 3.0 GHz and achieved 2.7% increase in speed of task completion.

Moreover, in [9], authors focused on energy consumption reduction by scheduling task statically and dynamically. In static approach, authors used 2 to 8 cores of speed 2.1 GHz and reduced the 31.9% power consumption with 2.6% performance loss. While, in dynamic approach authors have used L2 cache of size 3.5 MB additionally and saved 46.8% energy with 10% performance loss.

Further, authors in [13], performed experiments using both static and dynamic approaches. In dynamic approach authors increased the speed by 25% with four big cores of speed 2.1 GHz, L1 cache of 32 KB, L2 cache of 256 KB and L3 cache of size 4 MB. While, in static approach, they used the same cache with four small cores of speed 2.1 GHz and reduced the

TABLE III
COMPARISON OF JOB SCHEDULERS IN HMP.

ID	Scheduler	Clock Speed (GHz)	No. of Cores	Cache	Speed (%)	Energy (%)	Perf. Loss(%)
[2]	Dynamic	3.2x1, 0.8x3	4	L3(6MB)	12	39	-
[6]	Static	2.0x2, 3.0x2	4	-	4.7	-	-
	Static	2.2, 2.33, 2.67, 3.0	4	-	2.7	-	-
[9]	Static	2.1	2 to 8	-	-	31.9	2.6
	Dynamic	2.1	2 to 8	L2(3.5MB)	-	46.8	10
[13]	Dynamic	2.0	4 big	L1(32KB), L2(256KB), L3(4MB)	25	-	-
	Static	2.0	4 small	L1(32KB), L2(256KB), L3(4MB)	47	-	-
[14]	Dynamic	2.5	64 to 1024	L2(256KB)	0.6	16	-

47% task completion time. Furthermore, in [14], authors used dynamic approach with 64 to 1024 cores of speed 2.5 GHz and L2 cache of size 256 KB. They gained the speed of 0.6% and saved 16% energy consumption.

The above discussion shows the static and dynamic approaches to schedule jobs on HMP. The dynamic approaches have proved to be beneficial but due to the overhead between switching cores they yield a low performance. On the other hand, static approaches do not have dynamic scheduling, however, they use a predefined allocation of cores. The proposed approach in this study provides a solution to both of these issues. It dynamically takes care of priorities of cores and available tasks. Afterward, it maps each task to its preferred core which in turn reduces the switching overhead that will increase the throughput of the HMP and provide an energy efficient approach to process computation intensive tasks.

V. CONCLUSION AND FUTURE WORK

The HMP design is vast and there are many ultimate design choices to be made. The type of cores vary from simple in order to complex out of order cores. There are many possible configurations of core types and number of cores. Therefore, job to core mapping is both challenging and important for HMP to achieve optimum performance. Previous studies focused on static and dynamic techniques for scheduling tasks on different cores as presented in the above discourse. The main objective of scheduling tasks by employing these techniques was to save energy and enhance throughput. However, there were associated challenges related to switching tasks among cores such migrating overhead.

In the proposed solution, there will be dynamic priority lists for tasks and available cores. Here, each core will be assigned to that task which is at the 1st priority in core's list. If 1st priority task is not available then next task in the priority list will be assigned and so on. By following this approach, task execution and processor task mapping will yield reduction in task completion time. Thereby, task switching and migrating overhead will be optimized as well. In future, the main objective is to implement and simulate the proposed technique to evaluate its results in real time environment.

REFERENCES

[1] Q. Chen, Y. Chen, Z. Huang, and M. Guo, "Wats: Workload-aware task scheduling in asymmetric multi-core architectures," in *26th International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE, 2012, pp. 249–260.

[2] V. Petrucci, O. Loques, and D. Mosse, "Lucky scheduling for energy-efficient heterogeneous multi-core systems," in *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, 2012.

[3] L. A. Priyadarshni, "Heterogeneous multi core processors for improving the efficiency of market basket analysis algorithm in data mining," *CoRR*, vol. abs/1409.6679, 2014.

[4] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 125–138.

[5] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 927–930.

[6] D. Shelepov and A. Fedorova, "Scheduling on heterogeneous multicore processors using architectural signatures," in *In Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with the 35th International Symposium on Computer Architecture*, 2008.

[7] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE, 2003, pp. 81–92.

[8] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, p. 64, 2004.

[9] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. ACM, 2006, pp. 23–32.

[10] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 1994, p. 1.

[11] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*. ACM, 2006, pp. 29–40.

[12] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*. ACM, 2010, pp. 29–40.

[13] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3. IEEE Computer Society, 2012, pp. 213–224.

[14] G. Liu, J. Park, and D. Marculescu, "Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems," in *IEEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 2013, pp. 54–61.