Evaluating Dynamic Scheduling of Tasks in Mobile Architectures using ParallelME Framework *

Rodrigo Carvalho[1], Guilherme Andrade[2], Diogo Santana[1], Thiago Silveira[3], Daniel Madeira[1], Rafael Sachetto[1], Renato Ferreira[2], and Leonardo Rocha[1]

[1] Universidade Federal de São João del Rei, Brazil
{rodrigo,diogofs,dmadeira,sachetto,lcrocha}@ufsj.edu.br
[2] Universidade Federal de Minas Gerais, Brazil
{gandrade,renato}@dcc.ufmg.br
[3] Tech., Tsinghua University, China
zhuangzq16@mails.tsinghua.edu.cn

Abstract. Recently we observe that mobile phones stopped being just devices for basic communication to become providers of many applications that require increasing performance for good user experience. Inside today's mobile phones we find different processing units (PU) with high computational capacity, as multicore architectures and co-processors like GPUs. Libraries and run-time environments have been proposed to improve applications' performance by taking advantage of different PUs in a transparent way. Among these environments we can highlight the ParallelME. Despite the importance of task scheduling strategies in these environments, ParallelME has implemented only the First Come Firs Serve (FCFS) strategy. In this paper we extended the ParallelME framework by implementing and evaluating two different dynamic scheduling strategies, Heterogeneous Earliest Finish Time (HEFT) and Performance-Aware Multiqueue Scheduler (PAMS). We evaluate these strategies considering synthetic applications, and compare the proposals with the FCFS. For some scenarios, PAMS was proved to be up to 39% more efficient than FCFS. These gains usually imply on lower energy consumption, which is very desirable when working with mobile architectures.

Keywords: Dynamic scheduling · Parallel Mobile architectures.

1 Introduction

Recently we observe a growth in developing new technologies. An example is the evolution of traditional processors, which have become massively parallel and heterogeneous in response to the increasing demand posed to them by the new challenges in many different areas. Mobile devices as well are currently experiencing a substantial growth in processing power. They stopped being just

* This work was partially supported by CNPq, CAPES, Fapemig, INWEB and MAsWeb.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72

devices for basic communication among people to be providers of many applications such as Internet access, media playing, and general purpose applications, all requiring increasing performance for good user experience. Inside today's mobile phones we find processing units with high computational capacity, as multicore architectures and co-processors like GPUs, making these devices very powerful.

Effectivetly use these devices is still a challenge. Many applications from different domains need to explore all of the available Processing Units (PUs) in a coordinated way to achieve higher performance. Faced with this requirement, libraries and run-time environments have been proposed, providing a set of ways to improve applications' performance by using the different PUs in a transparent way to mobile developers [9, 10, 3, 4, 1]. Among these libraries and run-times systems, we highlight ParallelME [4], a Parallel Mobile Engine designed to explore heterogeneity in Android devices.

At the moment ParallelME has implemented just a simple *First Come First Serve* scheduling strategy and all results reported by in recent works [4, 14] used just that strategy. The purpose of this work is to extend the ParallelME, implementing and evaluating different dynamic scheduling strategies[4], whose results reported in the literature are promising in traditional architectures [6, 5, 15]. More specifically, we implemented two different scheduling strategies: (1) Heterogeneous Earliest Finish Time (HEFT) and; (2) Performance-Aware Multiqueue Scheduler (PAMS). HEFT was originally implemented in StarPU [7]. PAMS (Performance Aware Scheduling Technique) was proposed in [5]. Both uses knowledge of the tasks to create the task queues. In order to evaluate our strategy, we prepared a experimental set using a tunnable synthetic application, and compared our proposals with the FCFS implemented in ParallelME. With our results, we show that the new scheduling strategies, in special PAMS, achieves the best results in different scenarios, further improving the ParallelME's performance. For some scenarios, PAMS was up to 39% more efficient than FCFS.

2 Related Work

This section presents an overview of the main programming frameworks for parallel applications in mobile systems and dynamic scheduling strategies.

2.1 Parallel Mobile Frameworks

Nowadays, there are several high-level frameworks designed to facilitate the development of parallel applications in heterogeneous mobile architectures. Among these frameworks, we can highlight OpenCL and RenderScript, both providing tools for programming generic functions that can be executed in both GPUs and CPUs in Android OS, being the basis for several works [11, 2]. OpenCL is a framework originally proposed for desktop systems which allows the execution of

[4] Dynamic Scheduling Strategies evaluating the characteristics of the tasks at runtime, considering a limited view of the execution of entire application and thus a more challenging scenario.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72

user code (kernels) in statically-chosen processing units. RenderScript, in turn, is a framework designed by Google to perform data-parallel computations in Android devices, transparently running user code in heterogeneous architectures. Besides the frameworks presented above, we can find others in the literature, such as Pyjama [10] and Cuckoo [13]. Pyjama focuses on multi-thread programming with an OpenMP-like programming model in Java. Cuckoo [13], provides a platform for mobile devices to offload applications in a cloud using a stub/proxy network interface.

Although the presented frameworks share the same design goal, they have different features and programming interfaces, limited by it's complex programming abstraction, preventing their popularization among mobile developers. Recently we find in literature the framework Parallel Mobile Engine (ParallelME)[4], that was designed to explore heterogeneity in Android devices. ParalleME automatically coordinates the parallel usage of computing resources keeping the programming effort similar to what sequential programmers expect. ParallelME distinguishes itself from other frameworks by its high-level programming abstraction in Java and the ability to efficiently coordinate resources in heterogeneous mobile architectures.

2.2 Dynamic Scheduling Strategies

Many proposals of dynamic schedulers are found in literature [8, 17, 16, 12]. In [8] the authors focus on a strategy that minimizes the workload between processing units. The task distribution between the PUs is made randomly, but the inactive PUs can run tasks scheduled to another if it becames idle, using an approach called "work stealing". In [17] the authors presents Dynamic Weighted Round Robin (DWRR), which is focused on compute intensive applications. It's policy assigns a weight to each task in each PU and use this weight to sort the task queue of each PU. In [16], the authors present a strategy based on run time predictors. In this, the scheduler makes a prediction of the execution time for each task as well as the total time for a given PU to run all its tasks. Based on this prediction, the tasks are associated the less busy PUs, aiming to balance to load between the PUs. In [12] is presented a similar proposal, where such execution models are based on past run-time histories.

Another scheduling strategy that presents good results is HEFT (Heterogeneous Earliest Finish Time) [7]. HEFT uses a queue for each available processing unit and task distribution across queues is computed according to the processing capacity of each unit, based on expected execution time of previous tasks. Based on this strategy, in [5] the authors proposed another strategy called called PAMS (Performance-Aware Multiqueue Scheduler). Instead to consider the expected execution time as HEFT, PAMS takes into account performance variabilities to better utilize hybrid systems. Therefore, it uses speedup estimates for each task in order to maintain the order of tasks in the queues for each available processing unit. In this paper we implement HEFT and PAMS strategies in ParallelME and evaluate them using several synthetic applications with different characteristics.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72

3 Experimental Evaluation

This section presents the experimental results that were conducted to evaluate the implemented scheduling strategies in ParallelME: *FCFS HEFT*, and *PAMS*.

3.1 Workload Description

This subsection describes the workloads that were used to evaluate our dynamic scheduler. These workloads are composed of a synthetic application that allows fully control over several characteristics of the tasks. In a heterogeneous architecture a determining factor for the scheduler is the execution time ratio for the tasks on the different processing units, termed *relative speedup*. To create the workloads for our tests, first we divided our load into three groups:

- **Group 1:** Mostly CPU tasks.
- **Group 2:** Mostly GPU tasks.
- **Group 3:** Balanced CPU and GPU tasks.

In Group 1, the tasks on CPU consumes less time than on GPU. In Group 2, the inverse is observed (GPU are faster than CPU). Tasks in Group 3 are not significantly different in terms of execution time when executed on the CPU or on the GPU. For each group we varied the execution time for CPU and GPU, creating two different classes:

- **Class 1:** Tasks with high relative speedup (higher than 5x). For this, a bad decision can yield large performance hit on the application execution time.
- **Class 2:** Tasks with low relative speedup (lower than 5x). For this class, the performance penalty on the bad decisions are less significant.

Combining these three groups and the two classes above, we have created six distinct workloads, as presented in Table 1.

Table 1. Workloads used in our tests.

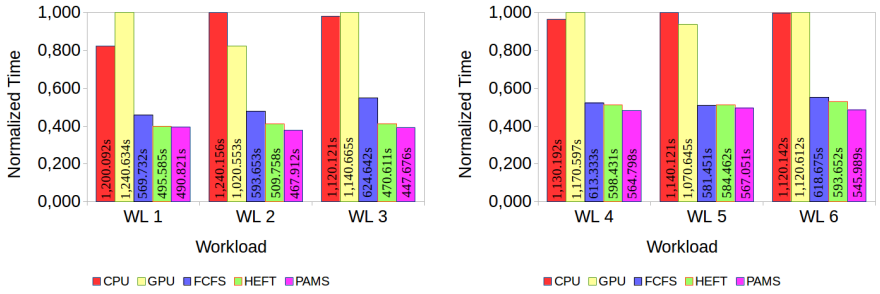
Name	Group	Class	Name	Group	Class
Workload 1 (WL 1)	1	1	Workload 4 (WL 4)	1	2
Workload 2 (WL 2)	2	1	Workload 5 (WL 5)	2	2
Workload 3 (WL 3)	3	1	Workload 6 (WL 6)	3	2

3.2 Experimental Results

This section presents the experimental results obtained with the new schedulers, tested with the workloads described in section 3.1. All tests were performed on a **Motorola Moto E 2nd Gen. running a quad-core Cortex A7 CPU @1.2GHz and an Adreno 302 GPU, with 1GB of RAM**. Three tests were performed. Firstly all workloads were scheduled only in CPU and only in GPU. After these two tests, the schedulers were tested using both CPU and GPU, using the schedulies strategy FCFS, HEFT and PAMS. Each workload were composed of 100 tasks. The results presented in Figure 1 are the average results after 10 executions, normalized by the higher execution time for each workload. Our

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72

objective is to evaluate the behavior of each scheduler policy, identifying their impact on application performance. As expected, independently of the scheduling strategy, all workloads were faster when using both CPU and GPU (2x on average). Also the better results were obtained in the workloads with higher relative speedup (workloads 1-3, all in Class 1).



(a) High Relative Speedup (b) Low Relative Speedup

Fig. 1. Application performance considering specific Processing Units and Scheduling Strategies. Lower is better.

When comparing each scheduler performance considering the two workload classes, we noticed that in Class 1 workloads, the performance gain of the HEFT and PAMS schedulers compared to the FCFS scheduler is greater than when we make the same comparison in Class 2 workloads. This happens because in Class 1 workloads the correct association of the task with the best PU is crucial to the performance of the application. When executing a task in the worse PU for it, the execution time is at least 5 times worse. Observing the performance of the scheduling algorithms in Class 2 tasks, the difference is much lower, that is, the average times obtained by the schedulers HEFT and PAMS are closer to those that were obtained by the scheduler FCFS. This can be explained because the relative speedup of these workloads is low. Therefore, even if the scheduler associates a task to a less appropriate PU, this does not damage too much the total execution time, since the execution times in both PUs are closer.

Finally, comparing the performance of each scheduler implemented in ParallelME, we can observe that the PAMS algorithm is more efficient than HEFT and FCFS for all workloads. Figure 1 shows that PAMS was faster than both FCFS and HEFT for all workloads and HEFT was only slower than FCFS on workload 5. PAMS and HEFT were respectively 18% and 12% more efficient than FCFS on average. The two new schedulers achieved their best when executing workload 3 (higher relative speedup and using both CPU and GPU). In this workload, PAMS was 39% and HEFT 33% more efficient than FCFS.

In order to better understand why performance differs among schedulers, we evaluate how tasks are distributed among the PUs available for each of the scheduling strategies implemented in ParallelME. The results of our studies are presented in Figure 2. In FCFS scheduler regardless of the characteristics of the workload, approximately 50% of the tasks are executed by the CPU and 50% by the GPU, giving a poorer performance. The HEFT scheduler presents a slightly

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72

better distribution than FCFS. This happens because HEFT scheduler considers the best PU for each task according to the estimated task runtime as well as the estimated working time for each PU. Through this strategy, the tasks, for the most part, are executed by the most appropriate PU. As described in previous sections, the HEFT scheduler queues are static, that is, after a task is assigned to a PU, it will no longer be processed by another.

Fig. 2. Profile of tasks assignmentto devices for each task executed in our application.

Finally, in PAMS algorithmthe distribution of the tasks for the PUs has a more direct relation with the characteristics of each workload. Most of the workload 1 and 4 (2 and 5) tasks (approximately 60%) were processed by the CPU (GPU). We also noticed that in workloads 3 and 6 the tasks were divided evenly, with 50% of the tasks processed by the CPU and 50% by the GPU. Since PAMS uses shared queues, a PU can process jobs allocated to another PU when it is idle (no jobs in its queue). This task distribution strategy used by PAMS makes it more efficient, as it minimizes idle time.

4 Conclusion and Future Works

In this paper we extended the ParallelME framework by implementing and evaluating two different dynamic scheduling strategies, HEFT and PAMS. We performed a comparative analysis, contrasting the performance of ParallelME original scheduler (FCFS) against the new scheduling strategies. We used a synthetic application in which we could control the tasks behavior, to create the six distinct workloads. Regarding the scheduling strategies, the results shows that PAMS was faster than both FCFS and HEFT for all workloads. HEFT was only slower than FCFS in workload 6. On average, PAMS and HEFT were respectively 18% and 12% more efficient than FCFS. The best performance was achieved for workload 3, on which PAMS and HEFT were, respectively, 39% and 33% more efficient than FCFS. It is important to mention that the gains in terms of execution time usually imply on lower energy consumption, which is desirable in mobile architectures. As future work we want to evaluate more elaborate scheduling strategies that may also improve the performance of ParallelME framework.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72

References

1. RenderScript. https://developer.android.com/guide/topics/renderscript
2. Acosta, A., Almeida, F.: Performance Analysis of Paralldroid Generated Programs. In: Parallel, Distributed and Network-Based Processing (PDP), 2014. pp. 60–67
3. Acosta, A., Almeida, F.: Performance analysis of paralldroid generated programs. In: Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on. IEEE (2014)
4. Andrade, G., de Carvalho, W., Utsch, R., Caldeira, P., Alburquerque, A., Ferracioli, F., Rocha, L., Frank, M., Guedes, D., Ferreira, R.: ParallelME: A Parallel Mobile Engine to Explore Heterogeneity in Mobile Computing Architectures, pp. 447–459. Euro-Par 2016
5. Andrade, G., Ferreira, R., Teodoro, G., da Rocha, L.C., Saltz, J.H., Kurç, T.M.: Efficient execution of microscopy image analysis on cpu, gpu, and MIC equipped cluster systems. In: IEEE SBAC-PAD 2014, Paris, France. pp. 89–96
6. Andrade, G., Ramos, G., Madeira, D., Sachetto, R., Clua, E., Ferreira, R., Rocha, L.: Efficient dynamic scheduling of heterogeneous applications in hybrid architectures. In: ACM SAC 2014. pp. 866–871
7. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures (2011)
8. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. Journal of the ACM **46**(5), 720–748. https://doi.org/10.1145/324133.324234
9. Frost, G.: Aparapi: Using GPU/APUs to accelerate java workloads. https://github.com/aparapi/aparapi (2014)
10. Giacaman, N., Sinnen, O., et al.: Pyjama: OpenMP-like implementation for Java, with GUI extensions. In: Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores. ACM (2013)
11. Gupta, K.G., Agrawal, N., Maity, S.K.: Performance analysis between Aparapi (a parallel api) and Java by implementing sobel edge detection algorithm. In: Parallel Computing Technologies (PARCOMPTECH). IEEE (2013)
12. Jooya, A., Baniasadi, A., Analoui, M.: History-aware, resource-based dynamic scheduling for heterogeneous multi-core processors. Computers Digital Techniques, IET **5**(4), 254 –262 (july 2011). https://doi.org/10.1049/iet-cdt.2009.0045
13. Kemp, R., Palmer, N., Kielmann, T., Bal, H.E.: Cuckoo: A computation offloading framework for smartphones. In: MobiCASE. pp. 59–79. Springer (2010)
14. d. Moreira, W., Andrade, G.N., Caldeira, P.H., Goncalves, R.U., Ferreira, R.A., Rocha, L.C., d. Sousa, R., Avelar, M.N.: Exploring heterogeneous mobile architectures with a high-level programming model. In: 29th IEEE SBAC-PAD. pp. 25–32 (2017)
15. da Rocha, L.C., Mourão, F., Andrade, G., Ferreira, R., Parthasarathy, S., Melo, D., Toledo, S., Chakrabarti, A.: D-sthark: Evaluating dynamic scheduling of tasks in hybrid simulated architectures. In: ICCS 2016, California, USA. pp. 428–438
16. Smith, W., Taylor, V., Foster, I.: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: Scheduling Strategies for Parallel Processing. pp. 202–219. Springer-Verlag (1999)
17. Teodoro, G., Sachetto, R., Sertel, O., Gurcan, M., Jr., W.M., Catalyurek, U., , Ferreira, R.: Coordinating the use of gpu and cpu for improving performance of compute intensive applications. IEEE International Conference on Cluster Computing (Sep 2009)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_72