ABSTRACT

Asymmetric multi-cores (AMCs) are a successful architectural solution for both mobile devices and supercomputers. These architectures combine different types of processing cores designed at different performance and power optimization points, thus exposing a performance-power trade-off. By maintaining two types of cores (fast and slow) AMCs have the potential to provide high performance under the facility power budget. However, there are significant challenges when using AMCs such as scheduling and load balancing.

In this thesis we initially explore the potential of these machines when executing current high-performance applications and we search for the most appropriate execution model. Specifically we evaluate several execution models on an ARM big.LITTLE AMC using the PARSEC benchmark suite that includes representative highly parallel applications. We compare schedulers at the user, OS and runtime system levels, using both static and dynamic options and multiple configurations, and assess the impact of these options on the well-known problem of balancing the load across AMCs. Our results demonstrate that scheduling is more effective when it takes place in the runtime system level as it improves the user-level scheduling by 23%; at the same time the heterogeneous-aware OS scheduling solution improves the user-level scheduling by 10%.

Following this outcome, this thesis focuses on increasing performance of AMC systems by improving scheduling in the runtime system level. Scheduling in the runtime system level is provided by the use of task-based parallel programming models. These programming models offer programmability and flexibility to the programmer as they consist of an interface and a runtime system to manage the underlying resources and threads. In this thesis we improve scheduling with task-based programming models by providing three novel task scheduling approaches for AMCs. These dynamic scheduling policies reduce total execution time either by detecting the longest or the critical path of the dynamic task dependency graph (TDG) of the application. They use dynamic scheduling and information discoverable during execution, fact that makes them implementable and functional without the need of off-line profiling. In our evaluation we compare these scheduling approaches

with an existing state-of the art heterogeneous scheduler and we track their improvement over a FIFO baseline scheduler. We show that the heterogeneous schedulers improve the baseline by up to 1.45× on a real 8-core asymmetric system and up to 2.1× on a simulated 32-core asymmetric chip.

Another enhancement we provide in task-based programming models is the adaptability to fine grained parallelism. The increasing number of cores on modern CMPs is pushing research towards the use of fine grained workloads, which is an important challenge for task-based programming models. Our study makes the observation that task creation becomes a bottleneck when executing fine grained workloads with task-based programming models. As the number of cores increases, the time spent generating the tasks of the application is becoming more critical to the entire execution. To overcome this issue, we propose TaskGenX. TaskGenX offers a solution for minimizing task creation overheads and relies both on the runtime system and a dedicated hardware. On the runtime system side, TaskGenX decouples the task creation from the other runtime activities. It then transfers this part of the runtime to a specialized hardware. We draw the requirements for this hardware in order to boost execution of highly parallel applications. From our evaluation using 11 parallel workloads on both symmetric and asymmetric systems, we obtain performance improvements up to 15×, averaging to 3.1× over the baseline.

Finally, this thesis presents a showcase for a real-time CPU scheduler with the goal to increase the frames per second (FPS) during the game-play on mobile devices using AMC systems.We design and implement the RTS scheduler in the Android framework. RTS provides an efficient scheduling policy that takes into account the current temperature of the system to perform task migration. RTS solution increases the median FPS of the baseline mechanisms by up to 7.5% and at the same time it maintains temperature stable.