Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era

Thannirmalai Somu Muthukaruppan[1], Mihai Pricopi[1], Vanchinathan Venkataramani[1], Tulika Mitra[1] and Sanjay Vishin[2]
[1]School of Computing, National University of Singapore
[2]Cambridge Silicon Radio
{tsomu,mihai,vvanchi,tulika}@comp.nus.edu.sg, Sanjay.Vishin@csr.com

ABSTRACT

Asymmetric multi-core architectures integrating cores with diverse power-performance characteristics is emerging as a promising alternative in the dark silicon era where only a fraction of the cores on chip can be powered on due to thermal limits. We introduce a hierarchical power management framework for asymmetric multi-cores that builds on control theory and coordinates multiple controllers in a synergistic manner to achieve optimal power-performance efficiency while respecting the thermal design power budget. We integrate our framework within Linux and implement/evaluate it on real ARM big.LITTLE asymmetric multi-core platform.

Categories and Subject Descriptors

C.1.4 [**PROCESSOR ARCHITECTURES**]: Parallel Architectures

General Terms

Algorithms, Design, Performance

Keywords

Asymmetric Multi-core, Power Management, Feedback controller.

1. INTRODUCTION

Computing systems have made an irreversible transition towards parallel architectures with multi-cores and many cores. However, power and thermal limits are rapidly bringing the computing community to another crossroad where a chip can have many cores but a significant fraction of them are left un-powered, or dark, at any point in time [7]. This phenomenon, known as dark silicon, is immediately visible in the embedded computing space where the restricted form factor rules out elaborate cooling mechanisms.

The dark silicon era is driving the emergence of asymmetric multi-cores where the cores share the same ISA (instruction- set architecture) but their micro-architectures offer diverse power/ performance characteristics. This heterogeneity enables better match between application demand and computation capabilities leading to substantially improved energy-efficiency [18, 9]. Indeed, ARM has recently announced big.LITTLE [2] processing for mobile platforms where high performance, out-of-order Cortex A15 cluster is coupled with energy-efficient in-order Cortex A7 cluster in the same chip as shown in Figure 1.

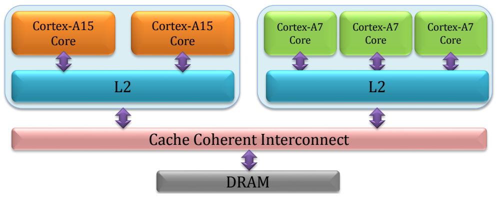


Figure 1: ARM big.LITTLE asymmetric multi-core.

We present a comprehensive power management framework for asymmetric multi-cores — in particular ARM big.LITTLE architecture in the context of mobile embedded platforms — that can provide satisfactory user experience while minimizing energy consumption within the Thermal Design Power (TDP) constraint. Compared to homogeneous multi-cores, power management is challenging on asymmetric multi-cores under limited TDP budget. We set out to design our framework with the following objectives:

- The dramatically different power-performance behavior of the cores implies that we need to identify the right core for the right task at runtime and migrate the tasks accordingly.
- The power hungry complex cores should be employed sparingly and only when absolutely necessary.
- Dynamic Voltage and Frequency Scaling (DVFS) as a control knob is available per cluster rather than per core within a cluster necessitating appropriate load balancing strategies. A cluster should run at the minimum frequency level required for adequate user experience so as to conserve energy.
- The restricted TDP budget precludes certain combination of frequencies for the different clusters. For example, it may be necessary to power down A7 cluster when A15 cluster is running at maximum frequency, a canonical example that illustrates the impact of the dark silicon era. Thus power budget has to be allocated opportunistically among clusters.
- If a system exceeds the power budget, the quality-of-service (QoS) of the tasks should degrade gracefully.
- The framework should be integrated in a commodity operating system without altering any of its desirable properties.

While, there exists solutions in the literature focusing on at least a subset of the objectives mentioned earlier, each of these solution have been generally designed to operate independently. It should be clear that deploying them together requires a carefully coordinated approach that is aware of the complex interplay among the individual solutions. For example, once the system exceeds the TDP of the entire chip, the power budgets for the clusters have to be reduced, which implies scaling down the voltage and frequency levels of the clusters, and consequently degrading the QoS of the tasks that triggered the thermal emergency in the first place. However, once the system load decreases (e.g., some tasks leave the system), this process has to be reversed and the QoS of the tasks should be restored back to the original level. This requires synergistic interaction among the different solutions so as to ensure *safety* (operate under power budget) and *efficiency* (optimal tradeoff between power and performance), while maintaining *stability*, i.e., avoiding oscillation between different operating points.

We design a *hierarchical power management framework* that is based on the solid foundation of *control theory* and integrates multiple controllers to collectively achieve the goal of optimal energy-performance tradeoff under restricted power budget in asymmetric multi-core architectures. Moreover, we build our framework as an extension of *Linux completely-fair scheduler* while preserving all of its desirable properties such as fairness, non- starvation etc. We take advantage of Heart Rate Monitor [8] infrastructure in Linux to set the performance goal for a task and to monitor its execution progress as a measure of QoS. Finally, our Linux-based hierarchical power management framework is implemented on *real ARM big.LITTLE platform* exploiting all the control knobs provided on the platform, namely, per cluster DVFS, cluster power down, and task migration within and across clusters.

To the best of our knowledge, ours is the first work to *provide a comprehensive power management approach for asymmetric multi-cores under limited power budget* and definitely the first one to *integrate the solution in a commodity operating system (Linux) running on real platform (ARM big.LITTLE)*. Our key contributions are

- Our power management framework successfully achieves all the objectives enumerated earlier.
- Our solution builds on a formal control-theoretic approach that provides guarantees for safety, efficiency, and stability.
- Our hierarchical framework carefully coordinates the controllers to avoid inter-controller interference.
- We integrate our framework within the confines of Linux and implement it on a test version of the ARM Big.Little asymmetric multi-core architecture and report power, performance results from this real chip (as opposed to simulation).
- We experimentally evaluate and establish the superiority of our approach compared to the state-of-the-art.

2. RELATED WORK

The asymmetric multi-cores, due to the power/performance trade-off [10, 6] introduce additional complexity to the scheduler. [11] proposed a scheduling algorithm for asymmetric cores that are simply symmetric cores using different frequency levels. [9] identified the key metrics for mapping a task to the appropriate core to optimize performance. [17] proposed an asymmetric- aware scheduler, where ILP and TLP threads are scheduled in fast and small cores, respectively. Operating system support for heterogeneous architecture with non-identical ISA was proposed in [12]. However, none of these techniques consider power management.

There exist plethora of works [5, 13, 14, 20] focusing on power management of homogeneous multi-core systems based on the control theory. [5] adapts the number of cores and frequency using an offline regression technique to keep the power below threshold. [14] allocates the chip power budget to each of the power islands based on absolute performance metric. [13] provided power capping technique for many core system consisting of both single thread and multi-threaded applications. [15] present a hierarchical control system for power management in server farms. In contrast, our power management framework operates on an asymmetric architecture and selectively penalizes the cores and the tasks under thermal emergency. [6] developed energy-aware scheduling for a single task on Intel QuickIA heterogeneous platform with two cores. In contrast, we provide a general framework that can handle any number of tasks and cores, satisfy QoS and thermal constraints, minimize energy, and is implemented in Linux on a real platform.

3. ARM BIG.LITTLE ARCHITECTURE

The big.LITTLE architecture is a system-on-chip comprising high performance Cortex-A15 cluster and power efficient Cortex-A7 cluster (see Figure 1). The test chip we use in this work contains three A7 cores and two A15 cores. All the cores implement ARM v7A ISA. Table 2 in Appendix details the architecture configurations. The architecture provides DVFS feature per cluster. Note that all the cores within a cluster should run at the same frequency level. Moreover an idle cluster can be powered down if necessary. We now provide a detailed power- performance characterization of the architecture. The detailed experimental setup used for this study appears in Appendix 8.1.

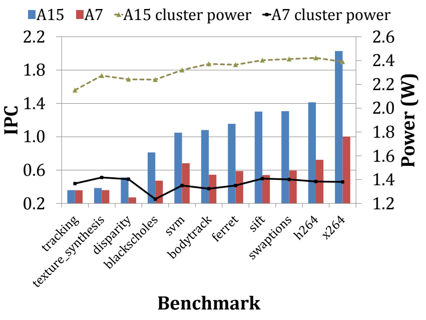


Figure 2: IPC & Power of A7 and A15 at 1GHz

Power-Performance Tradeoff. Figure 2 plots the Instructions Per Cycle (IPC) and the average power (Watt) for each benchmark on A7 and A15 cluster, respectively. For this experiment we set the frequency level of both clusters at 1GHz. Note that we can only measure the power at cluster level rather than individual core level. So the power reported in this figure corresponds to the power in a cluster even though only one core is running the benchmark application, while other cores are idle. Clearly, A15 has significantly better IPC compared to A7 (average speedup of 1.86) but far worse power behavior (1.7 times more power than A7 on an average).

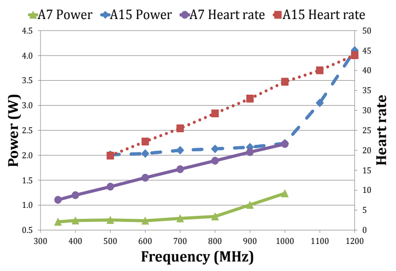
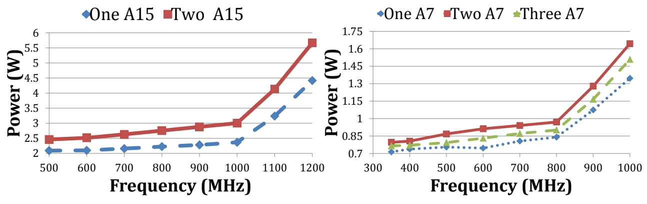


Figure 3: Power and heart rate with varying frequency.

Impact of DVFS. As mentioned earlier, our objective is to provide satisfactory user experience or QoS at minimal energy. We employ Heart Rate Monitor [8] infrastructure to set the performance goal and measure the QoS of a task. Heart rate is defined as the throughput of the critical kernel of a task, for example, number of frames per second for a video encoder. Figure 3 plots the heart rate and power for `blacksholes` from PARSEC benchmark suite on A7 and A15 at difference frequency levels. We observe that the heart rate increases linearly with increasing frequency on a core. Also as the IPC of A15 is better than A7, the heart rate can be improved by migrating a task from A7 to A15 at the same frequency level (but at higher power cost). Finally, the power generally increases linearly with increasingly frequency on a core; but there is a sudden jump at 800MHz for A7 and 1GHz for A15 due to change in voltage level.



(a) A15 Cluster Power (b) A7 Cluster Power

Figure 4: Impact of number of active cores on cluster power.

Impact of active cores on cluster power. As noted earlier, we can set frequency and measure power only at cluster level. Also we can only power down a cluster, but not individual cores. Thus, even if a core is idle, it still consumes power. Here we evaluate the impact of active cores on power consumption of the cluster. For this experiment, we run the same benchmark application on one, two, and three cores in A7 cluster as well as one and two cores on A15 cluster at different frequency levels. It is interesting to observe(Figure 4) that the A7 and A15 cluster have completely different power behavior with respect to the number of active cores.

In A7 cluster, even at the highest frequency level (1GHz), there is only 0.3 Watt difference between one active core and three active cores. In the A15 cluster, on the other hand, there is roughly 1.5 Watt difference in power between one active core and two active cores. For both clusters, it is important to perform load balancing and run all the cores at the lowest possible frequency level.

Migration Cost. Task migration across clusters is important to exploit the unique advantage of asymmetric multi-cores. We perform a set of experiments to quantize the migration cost within and across clusters (refer Appendix). We observed that the migration cost across clusters is higher than the cost within the cluster. Thus, task migration for load balancing within a cluster can be performed more frequently, whereas migration decisions across clusters should be taken at longer time intervals.

4. POWER MANAGEMENT FRAMEWORK

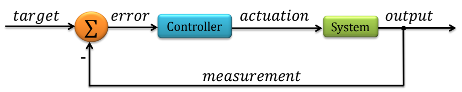


Figure 5: Feedback based Controller.

An overview of our hierarchical power management framework is presented in Figure 6. We incorporate several feedback based controllers in our framework. A controller measures the output metric and compares it with the reference or target metric as shown in Figure 5. The error is minimized by manipulating the actuators of the target system. The actuation policy is determined by the model of the target system being designed. We employ PID (Proportional-Integral-Derivative) controllers $z(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$, where $z(t)$, $e(t)$, K_p, K_i and K_d are the output of the controller, error, proportional gain, integral gain, and derivative gain, respectively.

We have two types of tasks in our system; *QoS* and *non-QoS* tasks. A *QoS* task is one that demands certain user-defined throughput (e.g., video encoder, music player), while the *non-QoS* tasks do not specify any QoS requirement. As noted in Section 3, we specify the QoS of a task in terms of its heart rate.

The framework consists of three different types of controllers: per-task resource share controller, per-cluster DVFS controller, and per-task QoS controller. Each QoS task in the system is assigned a resource share controller and a QoS controller. The resource share controller (bottom left in Figure 6) of a QoS task Q_i manipulates the CPU share available to Q_i so that it can meet the target heart rate $hr_{ref}(Q_i)$. The per-task QoS controller (top in Figure 6) is inactive when the entire system is lightly loaded. However, when the total power of the chip exceeds the TDP, the QoS controller slowly throttles the target heart rate $hr_{ref}(Q_i)$ so that the workload in the system decreases to a sustainable level and brings it back to the user- defined level when the thermal emergency is over.

We have two cluster controllers corresponding to A7 and A15 clusters. The objective of the controller for cluster Cl_m (bottom right in Figure 6) is to apply DVFS such that the utilization remains close to the target utilization $u_{ref}(Cl_m)$. The utilization of a cluster is determined by the maximum utilization of its cores. Thus, we periodically invoke a load balancer to ensure even utilization among the cores within a cluster. We also invoke a migrator periodically (at a much longer interval compared to the load balancer) to migrate the tasks between the clusters if necessary. Finally, we have a chip-level power allocator (extreme right in Figure 6) that throttles the frequency of the clusters and forces QoS controller to degrade target heart rates when the total power exceeds the TDP.

The key challenge here is to coordinate the various controllers, load balancer, migrator, and chip- level power allocator. We achieve a synergistic coordination with two mechanisms. First, the different components in our framework are invoked at different timescales. The per-task resource share controller and load balancer are invoked most frequently, followed by per-cluster DVFS controller and per-task QoS controller, then the migrator, and finally the chip-level power allocator. This ensures that a task attempts to reach its QoS target by first manipulating its share in a core or through migration within a cluster. If this fails, then it tries to change the frequency of the cluster. As a last resort, the task is migrated to another cluster. The thermal emergency takes quite a long time to develop; hence the power allocator is invoked least frequently.

Second, the controllers communicate with each other through designated channels. For example, the resource shares of the tasks within a core (both QoS and non-QoS) determines its utilization, which is provided as input to the cluster controller. More interestingly, when the power exceeds TDP, the power allocator increases the target utilization levels of the clusters $u_{ref}(Cl_m)$. This indirectly achieves the goal of decreasing power as the cluster controller is forced to lower its frequency in order to meet the increased target utilization. In parallel, the power allocator also sends a heart rate throttling factor ($hr_{throttle}(Q_i)$) to each QoS controller which makes them slowly degrade their target heart rate. This reduced heart rate is communicated to the resource share controller, who in turn, reduces the CPU share of the QoS tasks and hence the processor utilization to a more sustainable level. Overall, the system stabilizes to a level where the total power is just below the TDP.

Per-Task Resource Share Controller. We employ one resource share controller per QoS task. The target heart rate of a task Q_i is defined as a range $hr_{ref}(Q_i) = [hr_{ref}^{min}(Q_i), hr_{ref}^{max}(Q_i)]$ and is set by the QoS controller. The objective of the resource share controller is to keep the measured heart rate $hr(Q_i)$ in the target heart rate range. This is achieved by regulating the slice $s(Q_i)$ of time provided to the task Q_i in the scheduler. For example, a task that does not meet the minimum heart rate would demand more resource, which translates to more slices of time. The manipulation

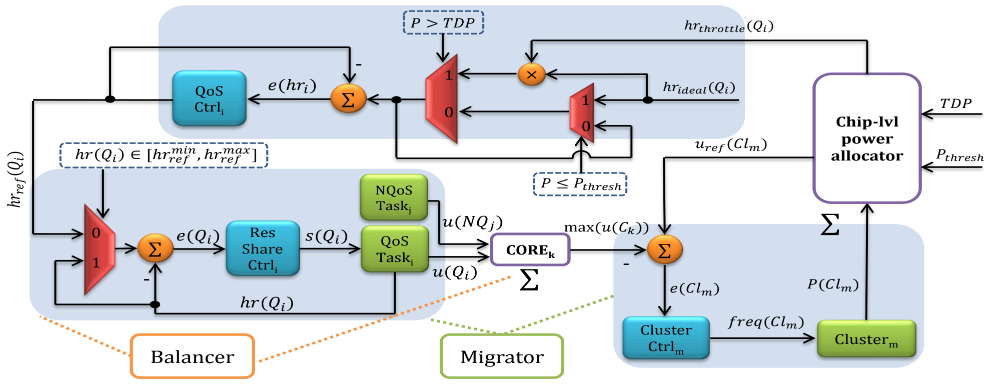


Figure 6: Overview of the hierarchical power management system coordinating multiple controllers.

of the slice value of a task within Linux completely fair scheduler is explained in detail in Appendix 8.1. If the measured heart rate is within the reference range, then the controller does not need any action and hence the target heart rate $hr_{ref}(Q_i)$ is set to the measured heart rate $hr(Q_i)$ so that error is zero in the controller.

Per-Cluster DVFS Controller. Let a core C_k consist of N QoS and P non-QoS tasks. Then its current utilization is $u(C_k) = \sum_{i=0}^{N} u(Q_i) + \sum_{j=0}^{P} u(NQ_j)$ where $u(Q_i)$ and $u(NQ_j)$ are the utilizations of the QoS task Q_i and non-QoS tasks NQ_j, respectively. The *core* component in Figure 6 is responsible for measuring the utilization of each individual core. As the frequency can be controlled only at cluster level, the utilization of cluster Cl_m defined as $u(Cl_m)$ is set to the maximum utilization $max(u(C_k))$ across all the cores within the cluster.

The DVFS controller attempts to achieve the target utilization $u_{ref}(Cl_m) = max(u_{ideal}, u_{target}(Cl_m))$ where u_{ideal} is a constant specifying the ideal target utilization and $u_{target}(Cl_m)$ is the target utilization set by the power allocator under thermal emergency. Using $max(u(C_k))$ as the measured metric and $u_{ref}(Cl_m)$ as the reference metric, the cluster-level PID controller actuates the frequency of the cluster.

Chip-Level Power Allocator. When the total power of the chip exceeds the TDP, the power allocator needs to throttle the frequency of the clusters and the QoS of the tasks. Let P_m be the current power measured for cluster Cl_m. The target power $\overline{P_m}$ for cluster Cl_m is calculated using the following equation

$$\overline{P_m} = P_m - \left((P - TDP) \times \frac{(T^{qos} - T_m^{qos})}{T^{qos}} \right) \tag{1}$$

where P is the total power of the chip given by $P = \sum_{m=0}^{M} P_m$, T^{qos} is the total number of QoS tasks in the system and T_m^{qos} is the total number of QoS tasks in the cluster Cl_m. From Equation 1, it is evident that the reference power allocated to the cluster is proportional to the number of QoS tasks in the cluster. From the reference power budget allocated to each cluster, the power allocator computes $u_{target}(Cl_m)$ using the following equation,

$$u_{target}(Cl_m) = u_{ideal} + u_{ideal} \times \frac{P_m - \overline{P_m}}{P_m} \tag{2}$$

In the event of TDP violation, the power allocator increases the target utilization $u_{ref}(Cl_m)$ of the cluster, which in turn causes cluster-level DVFS controller to set a lower frequency for the cluster. As our controllers are reactive in nature, the power may exceed the TDP for a short time interval. The gain factors within the DVFS controller are set appropriately so that it stabilities the power below the TDP within the specified time interval (typically few seconds [16]) as demonstrated in Section 5.

When TDP is violated, the power allocator also sets a throttle factor $hr_{throttle}(Q_i)$ for each QoS task Q_i in a hierarchical manner. The throttle factor $hr_{throttle}(Cl_m)$ for a cluster is proportional to its penalty factor as defined via higher than ideal utilization.

$$hr_{throttle}(Cl_m) = 1 - \frac{(u_{target}(Cl_m) - u_{ideal})}{u_{target}(Cl_m)} \tag{3}$$

The cluster throttle factor is further divided among the cores

$$hr_{throttle}(C_k) = hr_{throttle}(Cl_m) \times \frac{u(C_k)}{u_{avg}(Cl_m)} \tag{4}$$

where $u_{avg}(Cl_m)$ is the average utilization in cluster Cl_m across all the cores. Finally, the throttle factor of a QoS task in a core ensures that the penalty of a task is proportional to its utilization.

$$hr_{throttle}(Q_i) = hr_{throttle}(C_k) \times \frac{\sum_{i=0}^{N} u(Q_i)}{u(C_k)} \tag{5}$$

Once the system escapes from the thermal emergency, the power allocator needs to set back $hr_{throttle}(Q_i) = 1$. During the thermal emergency, the clusters reduce their frequency and the QoS tasks reduce their workload, the power decreases just below the TDP. However, the QoS of the tasks cannot be brought back to their ideal QoS level as the system will again oscillate back to thermal emergency. The QoS of the tasks can be restored only when the workload decreases because (a) one or more tasks leave the system and/or b) the tasks exhibit phase behavior. This is reflected in the drop in power consumption of the system. Thus, we chose an empirically determined power threshold P_{thresh} below which the $hr_{throttle}$ is set to one (as shown in Figure 6).

Per-Task QoS Controller. The QoS controller provides the graceful degradation of the QoS measure in case of thermal emergency by manipulating the target heart rate $hr_{ref}(Q_i)$. The input to this controller is the user-defined ideal heart rate range $hr_{ideal}(Q_i) = [hr_{ideal}^{min}(Q_i), hr_{ideal}^{max}(Q_i)]$. When the power is below the TDP, the power allocator sets $hr_{throttle}(Q_i) = 1$ and this controller sets $hr_{ref}(Q_i) = hr_{ideal}(Q_i)$. In case of thermal emergency, the controller strives to set the reference heart rate $hr_{ref}(Q_i) = hr_{throttle}(Q_i) \times hr_{ideal}(Q_i)$.

Load Balancer and Migrator. In our framework, the *Balancer* ensures that the cores within the cluster are evenly load balanced in terms of the utilization. The *Migrator* migrates the set of tasks that do not achieve their target heart rate at maximum frequency in the A7 cluster to the A15 cluster. Similarly, a task is migrated from A15 cluster to A7 cluster when the measured heart rate $hr(Q_i)$ is above the maximum target heart rate $h_{ref}^{min}(Q_i)$ at the minimum frequency in the A15 cluster.

5. EXPERIMENTAL EVALUATION

We implement our hierarchical power management framework, called *HPM*, on Versatile Express Development Platform [2] that includes ARM big.LITTLE chip. We use Ubuntu 12.10 Linaro release for Versatile Express [3] and Linux kernel release 3.6.1. The chip is equipped with sensors to measure frequency, voltage, power, and energy consumption of each cluster. Detailed description of the prototype implementation appears in Appendix 8.1.

We deploy PID controllers for per-task resource share controllers, per-task QoS controllers, and per-cluster DVFS controller. The gain parameters used for these controllers, the intervals for invocation of different components in our framework, and additional parameters are all presented in Table 6 in Appendix.

We use applications from PARSEC [4], Vision [19] benchmark suites and *h264* from SPEC benchmark [1]. We use the sequential version of the PARSEC benchmarks as QoS tasks. We specify and track the heart rates for the QoS tasks using Heart Rate Monitor infrastructure [8] integrated with our Linux kernel. We use the applications from Vision benchmark suite as non-QoS tasks. The details of these benchmarks appear in Table 3 in Appendix. Note that some of the benchmarks are computationally demanding (e.g., *x264*) and requires hardware accelerators for execution. As we run software-only versions of these benchmark, they achieve low heart rate even on A15 core at highest frequency.

The evaluations are designed to demonstrate that HPM achieves the following objectives: (1) HPM can exploit asymmetry to provide significant energy savings compared to symmetric multi-cores, (2) HPM performs better than the Linaro scheduler, (3) HPM can respond to thermal emergency in a graceful manner, and (4) HPM does not interfere with the desired properties of Linux CFS, namely, fairness and non-starvation of the non-QoS tasks (see Appendix).

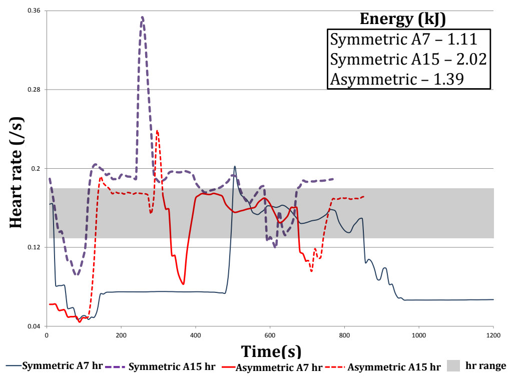


Figure 7: x264: Heart rate on symmetric & asymmetric multi-core.

Asymmetric versus symmetric multi-core. We use *x264* benchmark that exhibits phases with varying performance requirements during execution. The symmetric architectures are emulated using only A7 cluster or A15 cluster. We run *x264* benchmark on each of these configuration. All the configurations use HPM framework; but inter-cluster migration is disabled for symmetric architectures. Figure 7 plots the heart rate on the asymmetric and symmetric configurations. The heart rate line type specifies the cluster on which the task is running: continuous line corresponds to A7 cluster and dashed line corresponds to A15 cluster. The gray shaded area shows the specified heart rate range.

On symmetric configurations, the measured heart rate is below the minimum heart rate most of the time when executing on A7 cluster, while the heart reate mostly exceeds the maximum heart rate when running on A15 cluster. As expected, the energy consumption is very low (1.11kJ) in A7 cluster and quite high in A15 cluster (2.02kJ). The asymmetric multi-core provides the best of both worlds. On the asymmetric architecture, we can see that the application migrates to A15 cluster for the demanding phases and moves back to A7 cluster as the computational demand decreases. The HPM manages to maintain the heart rate within the reference range with a very low energy consumption (1.39kJ), which is 68% less than the energy consumption on A15 cluster alone.

HPM versus Linaro scheduler. We compare HPM scheduler with Linaro scheduler kernel release 3.6.1, where we activate the power conservative governor. The Linaro scheduler is aware of the different performance capabilities of the asymmetric cores, but it does not react to different performance requirements of the QoS tasks. Once the task load (defined as time spent on the runqueue of the processor) increases above a predefined threshold, the Linaro scheduler moves the task to the more powerful core. However, it never migrates the task back to the weaker core when workload reduces. We launch three QoS tasks, *x264*, *bodytrack*, *h264*, on three A7 cores. The results are shown in Figure 8. In all the subgraphs the X-axis shows the time in seconds. The Y-axis in the first three subgraphs shows the measured heart rate of the QoS tasks under HPM and Linaro. Additionally, the figure shows the specified heart rate range for the tasks as grey shared area. The last subgraph shows power comparison between the two approaches.

bodytrack and *h264* meet their specified heart rate on A7 cluster. As *x264* does not meet its heart rate on A7 all the time, it is migrated to A15 cluster by HPM when necessary. All the while, HPM keeps the heart rate of all the applications within the specified range. The Linaro scheduler, on the other hand, migrates all the tasks to the A15 cluster based on task load. As a result, the tasks complete execution much earlier compared to HPM; but exceeds the heart rate by a large margin consuming significantly more energy. On an average, the system consumes 2.27W using our scheduler compared with 5.83W consumed under Linaro scheduler.

Table 1 quantitatively shows the average power consumption and heart rate miss percentage (i.e., how much time a QoS task spends below its minimum specified heart rate) for HPM and Linaro scheduler using identical experimental setup but five different combination of QoS benchmarks. In general, a small loss in performance of the QoS tasks in our framework is heavily compensated by the average power reduction. The Linaro scheduler performs quite badly even in terms of performance in the two highlighted experiments. This is because the benchmarks are very demanding. Linaro scheduler moves them all to the A15 cluster, where they suffer from lack of resources, even at the highest frequency level. HPM uses the resources more efficiently and miss rate is reduced along with considerable reduction in power consumption. The results clearly demonstrate that HPM exploits the asymmetric architecture much more efficiently than current Linux scheduler.

	HPM scheduler		**Linaro scheduler**	
Benchmarks	**Avg Power(W)**	**hr miss %**	**Avg Power(W)**	**hr miss %**
swap_h264_x264	3.35	8.27	6.18	5.95
swap_h264_body	3.88	13.39	6.06	9.80
h264_body_black	**4.19**	**15.65**	**6.00**	**33.99**
black_x264_h264	**4.21**	**19.93**	**6.19**	**29.76**
x264_body_h264	2.27	9.61	5.83	7.41

Table 1: Quantitative comparison of HPM with Linaro scheduler.

Response under TDP constraint. This experiment evaluates the efficiency of HPM in managing the chip power below the TDP through DVFS and graceful degradation of the QoS of the tasks if necessary. For fair comparison, we add a feature to the Linaro

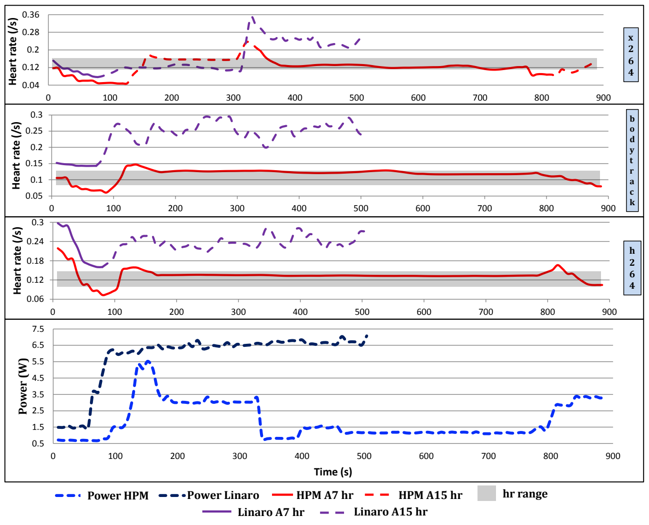


Figure 8: HPM versus stock Linaro scheduler equipped with DVFS governor and inter-cluster migration.

scheduler that switches off the A15 cluster once the power exceeds the TDP threshold. We use *bodytrack*, *swaptions*, and *h264* where the first two benchmarks have high workload and are migrated to A15 cluster. *swaptions* is the most demanding one and sets the frequency of the A15 cluster to the highest value. As we cannot control the frequency of individual cores, the core with *bodytrack* is forced to run at a higher frequency than required and hence its heart rate exceeds the target. Figure 9a shows the heart rate of *swaptions* (the application with maximum impact on power) together with the median value of the target heart rate range. The subgraph at the bottom of Figure 9a shows the chip power and the specified TDP cap. In this experiment, we dynamically change the TDP cap between 4-8W to demonstrate how the scheduler adapts to TDP budget. Once the chip power exceeds the TDP, the power allocator immediately increases the target utilization value of the clusters, which forces the DVFS controllers to decrease the frequency, and thereby reduce total chip power. Meanwhile, the power allocator also sets the heart rate throttle values, which in turn makes the QoS controllers reduce the target heart rates correspondingly bringing down the workload to a more sustainable level. HPM always maintains the heart rate of *swaptions* at the target value. Note that the target heart rate is decreased by the QoS controller when the power is above the TDP, thereby degrading the performance of the tasks. Once TDP is increased, the target heart rate switches back to the user-specified ideal value.

In case of the modified Linaro scheduler (Figure 9b) the A15 cluster is switched on and off frequently in response to increase in chip power beyond TDP. This oscillation happens because the workload is not throttled when the A15 cluster is switched off. As soon as A15 cluster is switched off, the power decreases much below the TDP, the tasks again migrate back to A15, the power increases above TDP, and the cycle continues. This frequent powering down of clusters and consequent migration makes *bodytrack* and *swaptions* run below their target heart rate most of the time under modified Linaro scheduler. This experiment confirms that a holistic approach is required to maintain the chip power below TDP; our approach not only decreases the frequency of the clusters but also solves the root cause of increased power by slowly degrading the QoS of the tasks. As a result, our approach reaches a stable and sustainable level both w.r.t. the heart rate and the chip power.

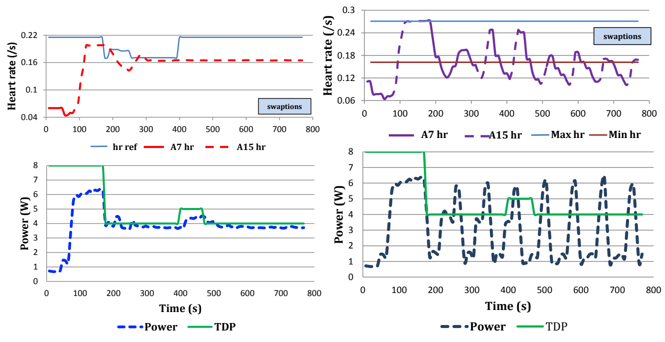

(a) HPM (b) Cluster switch off

Figure 9: Comparison of HPM and Linaro extended with cluster switch-off policy under TDP constraint.

6. CONCLUSION

We present a power management framework for asymmetric multicores that carefully coordinates multiple controllers. It is integrated with Linux on ARM big.LITTLE platform. It exploits asymmetry among the cores through selective migration and employs DVFS to minimize power consumption while satisfying QoS constraints. Our technique combines graceful QoS degradation along with DVFS to reach stable and sustainable execution under TDP.

ACKNOWLEDGMENTS: This work was supported by CSR research funding, and Singapore Ministry of Education Academic Research Fund Tier 2 MOE2009-T2-1-033, MOE2012-T2-1-115.

7. REFERENCES

[1] SPEC CPU Benchmarks. `http://www.spec.org/benchmarks.html`.
[2] ARM Ltd., 2011. `http://www.arm.com/products/tools/development-boards/versatile-express/index.php`.
[3] Linaro Ubuntu release for Vexpress, November 2012. `http://releases.linaro.org/12.10/ubuntu/vexpress/`.
[4] Bienia et al. The PARSEC benchmark suite: characterization and architectural implications. PACT, 2008.
[5] Cochran et al. Pack & Cap: Adaptive DVFS and thread packing under power caps. In *MICRO*, 2011.
[6] Cong et al. Energy-efficient scheduling on heterogeneous multi-core architectures. In *ISLPED*, 2012.
[7] Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.
[8] Hoffmann et al. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In *ICAC*, 2010.
[9] Koufaty et al. Bias scheduling in heterogeneous multi-core architectures. In *EuroSys*, 2010.
[10] Kumar et al. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO*, 2003.
[11] Li et al. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *ACM/IEEE conference on Supercomputing*, 2007.
[12] Li et al. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *HPCA*, 2010.
[13] Ma et al. Scalable power control for many-core architectures running multi-threaded applications. *ACM SIGARCH*, 2011.
[14] Mishra et al. CPM in CMPs: Coordinated power management in chip-multiprocessors. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*.
[15] Raghavendra et al. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGOPS*, 2008.
[16] Rotem et al. Power-management architecture of the intel microarchitecture code-named sandy bridge. *MICRO*, 2012.
[17] Saez et al. A comprehensive scheduler for asymmetric multicore systems. In *EuroSys*. ACM, 2010.
[18] Van Craeynest et al. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ISCA*, 2012.
[19] Venkata et al. Sd-vbs: The San Diego Vision Benchmark suite. In *IISWC*, 2009.
[20] Wang et al. Adaptive power control with online model estimation for chip multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 2011.

8. APPENDIX

8.1 big.LITTLE Platform with Linux

In this section, we explain in detail our target platform and prototype implementation of our HPM technique. In our evaluation, we used the real Versatile Express development platform [2] as shown in Figure 10. It is a flexible, configurable and modular developing platform that allows quick prototyping of hardware and software projects. The system comprises a motherboard on which modular daughter boards can be plugged. The big.LITTLE processor is part of the daughter board (TC2) pointed in the Figure 10. Table 2 describes the architecture of the A15 (big cores) and A7 (LITTLE cores). The motherboard handles the interconnection between the daughter board and the peripherals by using a FPGA bus interconnection network.

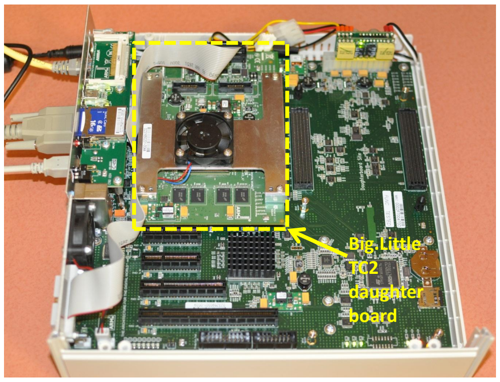


Figure 10: Picture of the Vexpress board.

Processor	Cortex-A7	Cortex-A15
Issue Width	2	3
Pipeline Stages	8-10	15-24
L1$	32kB 4-way	512kB 2-way
L2$	512kB 8-way	1MB 16-way
Frequency Levels	8	8
Frequency Range(MHz)	350-1000	500-1200
Voltage Range(mV)	900 - 1050	900 - 1050

Table 2: Cortex-A7 and Cortex-A15 specifications.

The board boots an Ubuntu 12.10 Linaro release for Versatile Express [3]. The platform firmware runs on an ARM controller (MCC) embedded on the motherboard and handles the load of the Linux kernel while booting. The Linux file system is installed on the Secure Digital (SD) card where all our benchmarks are saved. The TC2 daughter board is also equipped with sensors for measuring the frequency, voltage, current, power and energy consumption per cluster. The board also supports the change of voltage and frequency per cluster.

The migration cost among cores within A15 cluster is 54 μsec – 105 μsec depending on the frequency level, while the cost within A7 cluster is 71 μsec – 167 μsec. However, the migration costs between clusters are somewhat high: 1.88ms – 2.16ms for moving from A7 to A15 cluster at different frequency levels, and 3.54ms – 3.83ms for a move from A15 to A7 cluster.

Heart Rate Monitor. We use the Application Heartbeats framework proposed in [8] as a mechanism to measure the performance of an application. The API provided in this framework provides a QoS metric in terms of *heartbeats* which are periodic signals sent by an application to track its progress. The QoS metric provided by the framework is called *heartrate* (i.e, the number of heartbeats per second). For example, in video encoding applications the heartbeats can be registered every frame. Thus, the heart rate measured would be the number of frames per second. The interested reader is referred to [8] for more information on Heartbeats Framework.

Table 3 describes the benchmarks used in our experiments together with the inputs. Table 4 summarizes heartbeat insertions in the benchmarks [8].

Benchmark	Heartbeat Location
swaptions	Every "swaption"
h264	Every frame
bodytrack	Every frame
x264	Every frame
blackscholes	Every 25000 options

Table 4: Heartbeats in QoS benchmarks.

Profiling Section. The Linux version [3] provides hardware monitor (*hwmon*) interface to communicate with the sensors located in the test chip. We use the perf tool provided with the kernel [3] to obtain performance related metrics like instructions per cycle (IPC). Powering off the cluster and adjusting the clock frequency were made possible by accessing the oscillator related drivers provided in the kernel. The legal voltage and frequency ranges for the clusters are shown in Table 2.

Managing task slice and Migrator. Linux scheduler uses the notion of time slicing for allocating the resources to the running tasks in the system. At every system tick (10ms in our experiments), the kernel computes the time slice that the next tasks should receive. By default, the CFS scheduler fairly divides a relatively fixed period of time (6ms in our experiments) and allocates the slice to the task. The slice dictates the duration for which the task can consume the core. Our Resource Share Controller manipulates the computed slice for the QoS task by the original Linux mechanism by gradually increasing the time slice when a higher utilization is required or reducing the slice when less utilization is required. For non-QoS tasks, the CFS scheduler will try to fairly share the remaining time period among them. Linux kernel uses cpumask to decide the affinity of the tasks. Migrator component in our HPM alters the cpumask associated with each task to attain the desired scheduling decision.

In Table 5 we show the minor modifications that we did in the Linux kernel in order to implement our HPM scheduler.

8.2 Controller Features

Table 6 summarizes our HPM framework, describes the terminologies and provides the gain factor values associated with each of the controllers employed in our experiments. The invocation period of RSC is a user-defined value. For example, for video encoding it can typically be 30 frames per second, which translates to RSC being evoked every 30 frames. The invoke period was chosen in such a way that the per-task resource share controller and load balancer are invoked most frequently followed by DVFS controller, per-task QoS controller, migrator and finally the chip-level power allocator.

Benchmark	Benchmarks suite	Description	Inputs
swaptions	PARSEC	QoS — Monte Carlo (MC) simulation to compute the prices.	sim_native
bodytrack	PARSEC	QoS — Tracks a human body with multiple through a series of image sequence.	sim_native
x264	PARSEC	QoS — Video encoder.	sim_native
blackscholes	PARSEC	QoS — Solves partial differential equation to calculate the prices for a portfolio.	sim_native
h264	SPEC 2006	QoS - Video encoder.	foreman
disparity	Vision	non-QoS — Motion, tracking and stereo vision	fullhd
sift	Vision	non-QoS — Image Analysis	fullhd
tracking	Vision	non-QoS — Motion, tracking and stereo vision	fullhd

Table 3: Benchmarks description.

Function	Description	# lines
scheduler_tick()	Fire controllers based on system tick.	30
load_balance()	HPM Balancer within the cluster.	12
run_rebalance_domains()	HPM Migrator across the clusters.	47
sched_slice()	Manipulate the QoS time slices.	5

Table 5: Linux kernel modifications.

Controller name	Metrics	Symbol	Value
Resource Share Controller (RSC)	target heart rate	hr_{ref}	tuned by QoSC
	measured heart rate	hr	measured by the task
	slice	s	actuator tuned by RSC
	proportional gain	K_p^{RSC}	0.8512
	integral gain	K_i^{RSC}	0.01241
	derivative gain	K_d^{RSC}	0.00941
	invoked period	$T^{RSC} = \beta \times \frac{1}{hr_{ideal}}$	determined by the hr_{ideal}
	heart rate measurement frequency factor	β	user-defined
CORE component	core utilization	u_k	measured by each core
	invoked period	$T^c = 4 \times max(T^{RSC}(Q_i))$	determined by the task with max hr_{ideal}
DVFS Controller (DVFSC)	target cluster utilization	u_{ref}	estimated by CHIP component
	measured cluster utilization	$max(u(C_k))$	measured by CORE component
	cluster frequency	$freq$	tuned by DVFSC
	proportional gain	K_p^{DVFSC}	0.9533
	integral gain	K_i^{DVFSC}	0.2572
	derivative gain	K_d^{DVFSC}	0.0014
	invoked period	$T^{DVFSC} = 5 * T^c$	slower than the CORE component
QoS Controller (QoSC)	ideal hr	hr_{ideal}	user-defined
	throttle factor	$hr_{throttle}$	estimated by CHIP component
	target reference hr	$hr_{ideal} \times hr_{throttle}$	product of ideal hr and throttle factor
	measured reference hr	hr_{ref}	measured by the task
	proportional gain	K_p^{QoSC}	0.74175
	integral gain	K_i^{QoSC}	0.0214
	derivative gain	K_d^{QoSC}	0.0045
	invoked period	$T^{QoSC} = 30 * T^{RSC}$	much slower than RSC
CHIP level power allocator	thermal design power	TDP	user-defined
	threshold power	P_{thresh}	user-defined
	throttle factor	$hr_{throttle}$	estimated by CHIP component
	invoked frequency	$T^{ch} = 2 \times T^{QoSC}$	slower than DVFSC
Balancer	invoked period	$T^b = 2 \times T^{RSC}$	faster than DVFSC
Migrator	invoked period	$T^m = 4 \times T^{QoSC}$	slower than QoSC

Table 6: Controller Parameters.

8.3 Additional Experiment Results

We now present some additional experimental results.

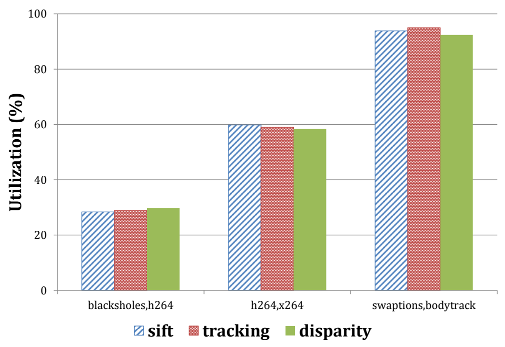


Figure 11: Fairness of non-QoS tasks.

Fairness of Non-QoS tasks. Our HPM framework is built on top of the existing Linux kernel scheduler. This set of experiments validate that we do not interfere with the scheduling of the non-QoS tasks handled by the Completely Fair Scheduler (CFS), which guarantees equal (fair) share of processor utilization among the tasks.

We run three experiments each with three non-QoS tasks (*sift*, *tracking*, *disparity*) and two QoS tasks. The behavior of the QoS tasks dictates the amount of A7 cluster utilization that CFS can provide to the non-QoS tasks. The first experiment uses *blackscholes* and *h264* as QoS tasks that satisfy the target heart rate on A7 cluster with close to maximum target utilization. Thus the CFS scheduler clusters together non-QoS tasks on the third available core. Figure 11 shows that the non-QoS tasks have equal share of utilization (33%).

The second experiment involves *h264* and *x264*; *x264* has a demanding execution phase where HPM migrates it to A15 cluster. Mostly the three non-QoS tasks run on their cores receiving 60% utilization, while *h264* runs on A7 cluster.

The final experiment uses *swaptions* and *bodytrack*, both of which migrate to A15 cluster and non-QoS tasks receive almost 100% of the A7 cluster utilization.