We would like to thank the Reviewers for their valuable comments to improve the quality of this work. Below we answer the specific comments of each reviewer and we also point out the changes that we performed in our paper following their suggestions. We have updated the Figures 3, 4, 5, 6 to increase readability.

Reviewer #1:

1. *Based on the description of the A-7 and the A-15 core in Section 2 it seems like the smaller A-7 core has a larger L1 associativity (4 way) than the bigger A-15 core core (2 way) although both have the same cahe capacity. Is this the case?*
    - Yes, the L1 associativity of the A7 core is 4-way associative and the associativity of the A15 core is 2-way. This information can be found in the technical reference manuals of these architectures in the following links: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0464d/DDI0464D_cortex_a7_mpcore_r0p3_trm.pdf http://infocenter.arm.com/help/topic/com.arm.doc.ddi0438c/DDI0438C_cortex_a15_r2p0_trm.pdf
2. *Although Section 3 introduces three different OS-level scheduling strategies the evaluation only compares against GTS scheduling. It would be helpful if the static threading approach presented in the evaluation is also introduced/discussed in this section. In static threading are the threads pinned to the cores? How does static threading work in the case of applications that utilize custom thread pool implementation?*
    - We have updated the text and have added subsection 3.4 within Section 3 to describe static threading in more detail.
3. *It would be interesting if the perf ratio (in Table 1) is also computed assuming the same frequency for the big and the little core. It would help attribute the improvement achieved big core over the little core to the clock frequency and to the core micro-architecture.*
    - This is indeed an interesting experiment. The table below shows the obtained ratios with frequency=1200MHz on both big and little cores. As shown on this table, the performance ratio is indeed affected by the frequency, but there is still performance difference even when they run on the same frequency.

App	Perf-paper	Perf same freq
Blackscholes	2,18	1,44
Bodytrack	4,16	2,12
Canneal	1,73	1,13
Dedup	2,67	1,46
Facesim	3,40	1,9
Ferret	3,59	1,84
Fluidanimate	3,32	1,63
Streamcluster	3,48	2,11
Swaptions	2,78	1,78

4. *In Figure 3, 4+0 and 0+4 configurations on average provide similar speedups irrespective of the scheduling strategy. However 4+0 configuration provides improvement with task based scheduling specifically for bodytrack and fluidanimate. Why is this the case?*
    - The task-based implementation of bodytrack is highly optimized to perform better than the pthreads implementation of the benchmark by overlapping communication with computation. This is because the task-based approach allows more programming flexibility by introducing tasks and task dependencies. The details of this implementation can be found in the paper by Chasapis et.al. "PARSECSs: Evaluating the Impact of Task Parallelism in the PARSEC Benchmark Suite". We have updated the text in paragraph 7 of Section 5.1 to explain this.
    - In the case of fluidanimate there was a typo in the data producing the charts which is now corrected. We have updated the chart on Figure 3 with the correct values. As a data-parallel application, fluidanimate does not improve much with the task based model for the 0+4 configuration.
5. *Although the performance of the three scheduling strategies are similar for many applications assuming a homogeneous 4+0 configuration, the average power consumption for task-based is comparatively much higher (blackscholes, bodytrack, dedup, facesim) . Is it because task-based scheduling is not as efficient as other static threading for symmetric multicore configurations? If so, why?*
    - The reason that for some applications the power consumption with the task-based solution is higher than the other approaches is that the task-based approach utilizes the big cores more effectively. The runtime system scheduling in the task-based approach leads to faster scheduling decisions, thus less idle time of the big cores. For that reason the power dissipation of the big cores with the task-based approach is observed to be higher. Less idle time of the cores means faster processing, thus increase in performance.
6. *The static threading results are discussed in detail in Section 5.1. However the results for GTS and task-based scheduling strategies are not discussed to the same extent. For instance, why is GTS successful in exploiting 2+2 configuration only for facesim, fluidanimate, streamcluster and swaptions and not for others?*
    - We agree that we should add more explanation regarding the GTS results. Specifically the reason that GTS is successful in exploiting the 2+2 configuration for these applications is that GTS is dynamically moving the threads around the cores depending on the CPU utilization. From this, it is expected that GTS will generally perform better than static threading for the asymmetric configurations (this is the reason it is designed for). However, for ferret that is an application with highly sophisticated parallelization static threading can achieve equally good results for the asymmetric configuration, avoiding the overhead of thread switching among the CPUs. On the other hand, canneal is an application that is by default memory intensive and with a low performance ratio, thus it is by definition hard for an OS scheduler that performs context switching to increase performance. The rest of the applications have the expected behavior for GTS which is to increase performance for asymmetric systems. We have added paragraph 7 in Section 5.1 to explain these scenarios.

7. *The last paragraph in Section 5.1 makes an observation about static threading and its limitation in the context of asymmetric configurations. IMHO based on the results this summary could also mention that static threading is the best approach in case of homogeneous configurations.*
    - We agree that the average performance results of the applications indicate that all approaches have similar performance and static threading achieves the lowest energy consumption, which makes it the most successful approach for symmetric configurations. However, there are applications like bodytrack, dedup and fluidanimate where more sophisticated solutions like GTS and task based perform better even for the homogeneous configurations. In addition, taking into account the energy and EDP results, we can see that on average static threading achieves similar results to the other approaches for the homogeneous configurations, thus it would be unfair to state that it is the best approach for such cases.
8. *In Section 5.2 it is mentioned that the average improvement is 15% over the symmetric configuration when 4 extra cores are added. This improvement however seems much smaller than what is indicated in Figure 2. Why is there such a considerable gap between the ideal and the actual improvement?*
    - The ideal speedup reported in Figure 2 is an ideal and practically unachievable speedup from all the applications. This speedup is theoretical and assumes that there are no inter-task dependencies neither synchronization points or memory overheads. Thus it is expected that the real evaluation results cannot reach the performance reported in Figure 2. The reason for adding this chart in the paper is so that we can give to the reader a theoretical ideal performance for each application on this platform, irrespective of implementation or scheduling mechanism used. We have added a brief explanation in order to demonstrate the usefulness of this chart in the second to last paragraph of Section 4.2.
9. *The energy results seem to indicate that it is inherently energy inefficient to keep the big cores busy and it is always better from the energy standpoint to carry out as much work as possible on the small cores. Why is this the case?*
    - This is correct. The design and layout of the little cores has been optimized for power efficiency while the design of the big cores targets higher performance levels at the cost of less energy-efficiency. Experimentally, the power dissipation of little cores is as much as 13 times less than the power dissipation of big cores (0.1W for 1 little vs 1.3W for 1 big, for bodytrack). As a consequence, approaches that lead to higher utilization of the big cores achieve better performance at the cost of less energy efficiency. We have added a comment in the last paragraph of Section 5.2 to highlight this insight.
10. *Although streamcluster and swaptions are grouped in a similar bin the results presented in Figure 7 indicate that 4+4 configuration provides improvement compared to 4+0 configuration only for swaptions and not for streamcluster. Moreover the improvement over static for the two other scheduling strategies is considerable for swaptions than for streamcuster. What can thi be attributed to?*
    - Swaptions shows an increased improvement with GTS and task-based solutions compared to streamcluster. This is because of the implementation and the nature of the two

applications. The task graph of streamcluster presents multiple parallel regions that are spawned and synchronized. Due to the multiple synchronization points, GTS and task-based cannot increase performance of streamcluster as much. Swaptions on the other hand, that is also a data-parallel application has less synchronization points, thing that allows GTS and task-based to exploit asymmetry. We have updated the text and have added this information on the 7th paragraph of Section 5.2, page 17.

11. *In general it would help the reader to follow the discussions in detail in Section 5.1 and 5.2 if there were individual subsections discussing the results for each scheduling strategy.*
    - Indeed his suggestion would help the reader, so we added one paragraph after section 5.2 titled as *Discussion* that summarizes the findings for each scheduling approach.
12. *To my understanding static-threading and loop-static both divide work statically between the threads without taking the heterogeneity of the system into account. What is it that causes Loop-static to perform considerably better than static-threading for 4+x configurations (when x>=1).*
    - It is true that loop-static theoretically performs the same scheduling as static threading. However there is an important difference. Loop-static does not pin the threads to cores but allows the GTS scheduler to migrate threads among cores. This is the default option for the OpenMP programming model. As we used the default version of it, the programming model's execution leaves the freedom to the OS scheduler to move threads to different types of cores. We have updated the text of the paper in the 5th paragraph of Section 5.3 to explain this.
13. *It is indicated in the discussion that loop-dynamic is more efficient on coarse grained parallel applications than task-based scheduling. Why is this the case?*
    - We agree that the text is making a generalization about the behavior of task-based and loop-dynamic approaches depending on the task granularity, without enough evidence to support this statement. This motivated us to research on the performance of swaptions and investigate more thoroughly why the behavior indicates loop-dynamic to perform more efficiently than task-based. During this process, we had to re-evaluate the performance of this application after regenerating the binaries. The results with the updated binary files showed that loop-dynamic and task-based have the same performance, thus we updated the data on the chart of Figure 9. The latest results make sense, since the implementation of the two approaches practically does not differ, as they both generate the same number of tasks and assign them dynamically to the available threads. We have updated the text according to the new findings in Section 5.3.

Minor Comments

1. *It would be good to have the average speedup numbers in Figure 2.*
    - We have added the average speedup numbers in Figure 2.
2. *It would help the reader of figure 3,4 and 5 are placed on the same page.*
    - We have updated the figures' placing in the current draft.

Reviewer #3:

1. *Findings are not very insightful. „It is fairly obvious that an application that is optimized for running on a homogeneous multi-core will run poorly on an asymmetric multi-core. Also, it is expected that a task based implementation „ which automatically schedules new tasks when others complete „will be a better fit for such architectures. The paper needs to add insight beyond this observation. For example, how do different task-based approaches perform? This is a much more interesting question as you would then compare approaches which one would expect to perform well.*
    - One very important insight of this paper is that even though the state-of-the-art solutions for asymmetric systems suggest scheduling in the OS level (GTS), this is not the optimal. This paper provides the very important insight that scheduling should take place in the runtime system. Comparing different scheduling policies within the runtime system is also very interesting and our paper introduces a related study on section 5.3, but further research on runtime level scheduling is out of the scope of this specific paper.
      In addition, even if it is expected for a task based implementation to be a better fit for these architectures, there is currently no study quantifying it on a real system, which makes our work novel. Quantifying such results requires a big effort in characterizing all parts of the evaluation including the applications as well as the scheduling approaches.
      We have added some of these insights in the conclusions section of the paper.
2. *The energy/power/EDP analysis is confusing. It seems that when performance goes up, power consumption goes up. When performance goes down, power consumption goes down. This makes sense as higher performance means the cores are working harder which results in more switching and high power. Energy is generally proportional to the amount of work to be done (i.e., instructions in the program). Is this intuition supported by your results? Please explain.*
    - The intuition of the reviewer is correct. The design and layout of the little cores has been optimized for power efficiency while the design of the big cores targets higher performance levels at the cost of less energy-efficiency. Experimentally, the power dissipation of little cores is as much as 13 times less than the power dissipation of big cores (0.1W for 1 little vs 1.3W for 1 big, for bodytrack). As a consequence, approaches that lead to higher utilization of the big cores achieve better performance at the cost of less energy efficiency. We have added a comment in the last paragraph of Section 5.2 to highlight this insight.
3. *The authors do not state clearly what were the main results of the experiments. Currently, they present a lot of numbers, but it is unclear what the key findings are and how the numbers back up these findings.*
    - As suggested by the reviewer, we have summarized the main insights of this study in the conclusions of the paper.
4. *The introduction does not indicate what are the root causes of the poor performance of current OS and runtime schedulers and how the runtime system approaches can overcome these issues (second to last paragraph). Foreshadowing the main findings in the introduction would make the paper much easier to read*

- We have updated the introduction in paragraphs 4 and 5 to indicate the causes of poor performance.

5. *The authors go into too much detail on the platform in Section 2. This section should only include the details that are needed to understand the results, and the authors can refer to the technical documentation of the platform for further details.*
    - We have shortened this section to include only the most important details of the platform and we provide the appropriate design document references.
6. *The authors fix the frequency of the cores to avoid overheating. Did you consider mounting a heatsink and possibly a fan? Static power depends on temperature so controlling temperature is critical to get consistent power measurements. Also, I'm concerned that (arbitrarily) fixing the frequencies of the big and small cores may affect the performance of different scheduling approaches. Intuitively, I would expect that the bigger the performance difference between the cores, the better the TBP approach will perform compared to the other approaches. Some sensitivity analysis on this issue should be added.*
    - Our systems are using heatsinks and fans in order to maintain their temperature. Still though, performing real experiments for a long time can cause overheating. Setting the core frequency is essential for our study not only to avoid overheating, but also to make sure that the reported performance is not affected by changes in the CPU frequency due to the DVFS governor. If we do not fix the cpu frequency, the running DVFS governor would change the frequency of each core dynamically and this is out of the runtime system or the GTS control. We have modified the text in the first paragraph of Section 4.1 to explain this decision.

      Performing experiments to address the impact of the governor is out of the scope of this paper. This sensitivity analysis is indeed interesting. We have added it as future work in the last paragraph of the conclusions of this paper.
7. *The authors state that they report the average over five runs, but they don't report the average variability (e.g., standard deviation). Please report this.*
    - Due to the large amount of data reported in this paper, adding the stdev or min and max values on the charts for each bar significantly reduces the visibility of the charts.
    - This table shows the max stdev for the speedup of task-based, GTS and static threading among all applications:

-	- MAX	- MEDIAN
- Task based	- 0,75	- 0,03
- GTS	- 1,11	- 0,05
- Static threading	- 0,53	- 0,02

    - We have modified the text to report the stdev for our results in the second paragraph of Section 4.1.
8. *Power measurements are collected online and may interfere with the running process. Does this affect all techniques equally? Please report how performance differs when power measurement is enabled vs. disabled.*
    - Our experimental methodology for the energy is verified to be adding negligible overhead in the execution of the application. We have verified that the additional overhead of the

measuring daemon was always less than 3%. More specifically, in most cases performance was not affected and for some cases there was an overhead of around 1-2%. These negligible values are not enough to alternate the outcomes of this study. We have clarified these overheads in the third paragraph of Section 4.1.

9. *Why do you normalise to four cores and static threading? Is this the configuration that consumes most energy or is there some other reason? Please explain.*
    - This normalization is just to help the reader to have a performance reference. We use the specific reference as it is the less aggressive one. Using this reference we assess the impact on the energy consumption by the increase of the big cores. Looking at the increase of energy on top of the configuration using four little cores we can easily see how much energy is increased by adding big cores. We have updated our explanation for this choice in the last paragraph of Section 4.1.
10. *The labelling of the figures is confusing (e.g., Figure 2). I would prefer to have the number of little cores on one line and big cores on the other line -- all clearly labeled. Another option is to consistently use the B+L labelling the authors introduce. The key issue is that it should be possible to understand the figure without reading the explanation in the text.*
    - We have updated Figure 2 in the draft to show the number of big cores on the top line and the number of little cores on the bottom line. We have also included the average results, as was suggested by Reviewer 1.
11. *The authors use a lot of space for introducing PARSEC, but most readers will be familiar with it. This discussion should be shortened.*
    - We have updated the description of the applications to take into account the comment of the reviewer. Due to repetition with other publications, we eliminated the description of each application on Table 1.
12. *Figure placement needs to be improved. A lot of figures are placed quite far away from where they are discussed which reduces readability.*
    - We have updated our figure placement in the current draft so that the figures and the text describing them are closer.
13. *The argument for choosing GTS over CS and IKS (footnote on page 12) is weak. Sometimes, less advanced techniques are better than more advanced techniques for non-intuitive reasons. It would have liked to see an experiment which shows that you are in fact comparing to the best performing Linux scheduler.*
    - This is the currently supported scheduler in our board so it was not feasible to use another approach. However, it is considered to be the most efficient for asymmetric systems and it is the only one with which we can have a fair comparison against task-based. For example, cluster switching allows only up to 4 cores to be in use and the resulting system is always homogeneous. This would not allow us to evaluate the scheduling approaches on an asymmetry-based environment. IKS is also limited, as it allows only up to 4 cores to execute simultaneously. We (will) add this in the text to explain these reasons in addition to the fact that GTS is the most efficient.
14. *It sounds a bit strange that the SMC with the four small cores is the most energy efficient configuration (final paragraph, Sec. 5.2), given that some applications get a significant speed-up*

when moving to the big cores. Does the actually energy consumption increase faster than the speed-up? Please explain.

- In the case of little cores, their design and layout has been optimized for power efficiency. Opposed to this, big cores are designed to target higher performance levels at the cost of higher energy consumption. As a consequence, approaches that lead to higher utilization of the big cores achieve better performance at the cost of less energy efficiency. The comment that we have added in the last paragraph of Section 5.2 to highlights this insight.