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.

**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**
   * 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>

1. 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?
   * **TODO** We have updated the text and have added a small subsection within section 3 to describe static threading in more detail
2. 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.
   * **TODO** This would indeed be an interesting experiment. However, we consider that for our evaluation the most insightful performance ratio to report is the one used in our real environment evaluation. We think that adding one more metric would confuse the reader and would not add much insight to the results.
3. 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?
   * **TODO**
4. 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? **🡪 less idle time**
   * **TODO** 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 more the big cores. This means that task-based sends more work to be executed on the big cores and this has as a result to increase the power dissipation, because big cores increase power.
5. 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?
   * **TODO** We agree that we should add more explanation regarding the GTS results. We (will) updated the document to explain these scenarios. 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 application have the expected behavior for GTS which is to increase performance for asymmetric systems.
6. 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.
   * **TODO**
7. 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?
   * **TODO** The ideal speedup reported in Figure 2 is an unachievable speedup from all the applications. This speedup is completely theoretical and assumes that there are no inter-task dependencies neither synchronization points or memory issues. Thus it is expected that the real evaluation results cannot reach the performance reported in Figure 2.
8. 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?
   * **TODO** This is due to the high power dissipation of the big cores compared to the little cores.
9. 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?
   * **TODO** Swaption 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.
10. 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.
    * **TODO** This suggestion is useful. We could add one paragraph titled as *discussion* and add a couple of sentences summarizing the findings for each scheduling policy.
11. 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).
    * **TODO** Even if the outcome of the scheduling is more or less the same with static threading and loop-static, still their implementation is different. Loop-static is implemented in a runtime system that is highly optimized. All the generated loop iteration consists of fine grained tasks that can be split among cores in a more fair manner. On the other hand, static-threading is implemented using pthreads in the application’s code. The resulting parallelization from static threading is more coarse and likely more imbalanced than the loop-static case.
12. 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?
    * **TODO** This is one of the outcomes of this evaluation. The reason is based on the observation that loop dynamic is the best option for swaptions, which is the most coarse grained application

Minor Comments

1. It would be good to have 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.

**Reviewer #3:**

The paper has a few aspects that I like:

+ Scheduling on asymmetric multi-cores is an important problem  
+ Evaluating policies on real hardware is a strength

However, there are also a few aspects that need to improve before I can recommend publication:

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.
   * **TODO** 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.
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.
   * **TODO** Energy in our experiments is proportional to the power and the execution time. Thus the higher the power dissipation observed, the higher the energy consumption.
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.
   * **TODO** We (will) rephrased the introduction as follows in these paragraphs
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
   * **TODO**
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.
   * **TODO** what to do?
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.
   * **TODO** 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 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 scope of the runtime system of the GTS control.  
     Performing experiments to address the impact of the governor is out of the scope of this paper. This sensitivity analysis is indeed interesting. We will add it as future work in 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. **🡪 report min and max values on the charts**
   * **TODO** We will report the min and max values on the detailed performance charts for each case
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.
   * **TODO** **🡪 overhead is less than 3%, is this statement enough?**
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.
   * **TODO** This normalization is just to help the reader relate to a prior performance and for simplicity. We (will) explain this choice in the text
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.
    * **TODO** We will change Figure 2 accordingly to show the number of big cores on one line and the number of little cores on the other line
11. The authors use a lot of space for introducing PARSEC, but most readers will be familiar with it. This discussion should be shortned.
    * **TODO** what to do?
12. Figure placement needs to be improved. A lot of figures are placed quite far away from where they are discussed which reduces readability.
    * **TODO we can change this…**
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.
    * **TODO** 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. **🡪 energy is linked to performance and power. Power is really low in the configuration SMC with four little cores so energy is low as well.**