**We are thankful to the reviewers for their valuable comments.**

**This paper extensively highlights the limitations of the state-of-the-art OS scheduling approaches (e.g. GTS) and demonstrates how to tackle them by using task-based parallel runtime systems. This work is novel as it demonstrates for the first time how the scheduling responsibility should be distributed across different layers of the software stack, not just the OS layer. To our knowledge, none of the previous works on asymmetric multi-core architectures has provided such a thorough and reliable performance, energy and power evaluation on real HPC applications.**

**The GTS approach is the most sophisticated dynamic OS job scheduling approach on the market and is deployed with most big.LITTLE boards. GTS is commercially available and is provided by ARM to offer performance and energy efficiency. GTS provides good results in multi-programmed workload scenarios, as has been shown in some publications from ARM/Samsung. However, our findings show that GTS is not ready to handle correctly workloads with a single parallel application running on the asymmetric system. We are not aware of any publication that highlights this limitation. In contrast, we prove that task-based programming models achieve much better results than GTS. We truly believe that this is a novel contribution to the community.**

**Additionally, our work exploits the use of assistant cores for the runtime activity. The findings of this section propose modifications on the task-based scheduling approach so that one core is devoted for the execution of the sequential code regions and of the runtime overheads. This part of the paper shows that applications with high sequential code regions benefit from executing them on a big core, while most of the applications benefit when the sequential code is executed on the first available core of the system.**

**We consider that this paper is a valuable piece of work that provides researchers with useful results for building and utilizing future asymmetric multi-core systems. The use of real hardware and real HPC workloads make this paper a reliable and unique illustration of the efficient exploitation of asymmetric systems, taking into account the performance and energy tradeoffs. For the above reasons, we believe that qualifying the paper as “Published before or openly commercialized” needs to be justified by a reference to a publication or product, otherwise we would kindly ask ReviewerC to reconsider this qualification.**

**We consider the set of applications used as representative and we agree that we should justify this in the text as ReviewerD correctly pointed. Moreover, to address the comment of ReviewerB, these applications are not embarrassingly parallel, but contain heterogeneous task types. We will also have to address that the application of the PARSECSs suite that we omit is freqmine, because it lacks a pthreads implementation, thus we could not perform the GTS and the static-threading experiments. Also as ReviewerD suggested, we will add possible application-based optimizations in the conclusions section. Finally, we plan to clarify in the text that the design of facesim and fluidanimate limits their results to core counts that are either multiples or powers of 2 respectively.**

**We agree that a few parts of the text could be improved. For example, as ReviewerA points, we will clarify that the phrase of page2 has the meaning understood by the Reviewer. We also plan to improve the related work by adding the missing references of Guided Self Scheduling. However GSS focuses on parallel loops while PARSECSs benchmarks use multiple parallelization strategies such as pipelines and unstructured parallelism. Furthermore, the text includes the description of Cluster-Switching and IKS for reasons of completeness. We agree that this part can be significantly reduced. Finally, as ReviewerD points, we will clarify that in the case of bodytrack, I/O tasks (with low CPU utilization) are scheduled to little cores when using GTS creating a bottleneck and limiting performance.**

**==========the end==============**

**=================Removed by Kallia===========**

**Finally, qualifying the paper as "Published before or openly commercialized" (Reviewer B) without giving any reference to a publication or product is really unfair and arbitrary. We completely disagree with this claim (as we have explained above) and we ask the reviewer either to provide such references or change his qualification.**

**###################**

**Specific questions:**

**###################**

**###RevA**

**1. "page 2": Exactly, this is the case. We will clarify the sentence.**

**2. "page 5": It is a limitation of the applications: fluidanimate requires a power of 2 cores and facesim an even number of cores. We will add this description. Also, the reviewer is right that Figure 2 is not used in the evaluation section and we will consider removing it.**

**3. Related work. Please, note that GTS already implements OS level task migration specifically developed for asymmetric multi-cores. With respect to Guided Self Scheduling, we will add the reference for sure, but we would like to note that GSS focuses on parallel loops while PARSEC benchmarks also use other parallelization strategies such as pipelines and unstructured parallelism.**

**###RevB**

**1. Objective and novelty: Please check the section "Main contributions and novelty"**

**2. Please, note that selected applications are not embarrassingly parallel, but contain heterogeneous task types.**

**###RevC**

**1. As the reviewer notes, thanks to this study, we can provide possible optimizations to applications useful for asymmetric multi-cores. We will add this discussion in the conclusions of the paper.**

**2. We plan to highlight why the set of benchmarks used from the PARSECSs suite is considered representative.**

**Why nine benchmarks? PARSEC has 13. PARSECSs has 11 (vips and raytrace are missing). We discarded x264 and freqmine (why?).**

**3. We included the description of Cluster Switching and In-Kernel Switching to provide a historical perspective of OS schedulers for asymmetric multi-cores. However, we agree that this description can be significantly reduced.**

**4. GTS does not use application-specific techniques as it only monitors CPU utilization. As a consequence, in the case of bodytrack, I/O tasks (which have lower CPU utilization) are executed on the little cores, which are in the critical path of the application. We will clarify**

**================Removed by Miquel===============**

**Moreover, we demonstrate that the addition of (at least one) little core to a homogeneous big-core system degrades system's performance even if the number of cores is increased. The commercially approved GTS dynamic scheduler does not manage to increase performance and effectively utilize the little cores. We choose the most naive scheduling approach (FIFO scheduler) among the wide variety of task-based schedulers since we want to point out that in its simplest form, a task-based scheduler constantly increases performance and keeps energy stable. We consider this a novel insight that does not appear in prior works.**

**==> Doesn't add much w.r.t. what we already say.**

**We agree that a few parts of the text could be improved so that they do not raise questions to the reader. For example we plan to explain that the design of fluidanimate and facesim benchmarks limits their results to core counts that are either multiples or powers of 2 respectively. Moreover, we plan to highlight why the set of benchmarks used from the PARSECSs suite is considered representative.**

**==> Included in detailed comments.**

**================old===============**

**The authors believe that the limitations of state-of-the-art OS scheduling approaches (e. g. GTS) are not well understood by the research community, especially when deployed over very unbalanced architectures (e. g. 4 big + 1 LITTLE cores). Even more, this paper is novel in the sense that it demonstrates for the first time how the scheduling responsibility should be distributed across different layers of the software stack, not just the OS layer. That is a key discovery to enable efficient asymmetric multi-core designs.**

**It is well known that currently the main challenge for future system designs is energy efficiency. Homogeneous multi-core systems are suffering by high energy consumption and this is the reason why mobile processors are nowadays heterogeneous, as we want the battery to last long. Many researchers are pushing towards the use of asymmetric multi-cores as a solution to tackle the power wall when running scientific applications. Our study is thoroughly stressing such a platform to evaluate its maturity of running high processing parallel applications like PARSEC.**

**Our contributions and insights are not provided by previous works since in our work we use the real hardware, with real applications and existing state of the art scheduling approaches. We believe that our paper clearly shows the performance and energy trade-off in such systems and what scheduling approach should be chosen.**