We are thankful to the reviewers for their valuable comments.

Below we address the comments of each Reviewer separately but we highly encourage all Reviewers to read all the addressed comments as there were similarities.

ReviewerA:

This work is novel as it demonstrates for the first time how the scheduling should be distributed across different layers of the software stack, not just the OS-layer. None of the previous works on AMCs has provided such a thorough and reliable performance and energy evaluation of real highly parallel applications.

None of the previous works has demonstrated the lack of flexibility of currently used OS schedulers on AMCs. Our work clearly shows the relying issues and provides ways to tackle them.

Adding more sophisticated scheduling algorithms to our evaluation could add some value to the paper. However these approaches are mainly implemented in the runtime system level and use extra information of the platform or the application. What we wanted to compare in this paper is representative approaches that are commonly used and well known.

Our motivation for this work relies on the need for energy efficient servers. Mobile SoCs provide high performance in low power and this motivates us to test on them highly parallel applications like PARSEC.

ReviewerB:

This study shows that the most efficient scheduling approach is when it is implemented in the runtime system. This is a new insight that has not been shown in any of the prior studies.

Our paper proves this as it compares two OS approaches against the runtime approach. We demonstrate that moving the scheduling in the platform-unaware runtime system performs better even from heterogeneity-aware OS scheduler (GTS).

The target for this work is researchers of mobile computing platforms that aim to enhance performance as well as HPC researchers that aim limit energy consumption by introducing asymmetry.

We have fairly evaluated the schedulers on static core frequencies. This is not the average power, neither the peak. Working on the peak power usually was causing shutdowns of the machine due to overheating, thing that was limiting us from completing this evaluation.

ReviewerC:

This paper compares different scheduling techniques. Out of these techniques only GTS is heterogeneity-aware. This is the commonly used approach on devices with AMC architecture.

Comparing this technique against heterogeneity-unaware scheduling methods shows that using the platform-unaware task-based approach obtains better results than GTS. Based on these observations, we conclude that GTS (or any OS-scheduler) should transfer part of (if not all) the scheduling responsibility to the runtime system. This component has better visibility of the application requirements and promotes load balance. This can be achieved by using a baseline system-unaware OS scheduler (e.g.CFS) and letting the runtime layer handle the scheduling. This is one key insight from this work.

ReviewerD:

The reason why loop-static performs better than thread-static is the implementation: loop-static implementations create several loops that are split into chunks according to the number of cores. Because in each implementation there are several loops the chunks of work created are a lot smaller than the pieces of work created when static-threading is implemented. This helps on minimizing synchronization costs and this is why performance is increased.