**R1:**

**We are sorry that the reviewer cannot see the contribution of our work.**

**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 of real highly parallel applications.**

**Moreover, we consider this work ideal for the ISPASS conference, as it is a thorough evaluation and characterization of an architecture, showing the current obstacles and how to overcome them in the runtime or OS level.**

**R2:**

**Our work focuses on the examination of the new mobile-based asymmetric systems for their use in the next generation multi-cores. As the main challenge of the future multi-core system design is energy efficiency, many researchers are pushing towards the use of the energy efficient asymmetric multi-cores to build such systems and tackle the power wall. However, our work is not tied to a particular architecture. Our insights apply to asymmetric multi-cores in general. Regarding the applicability of ARM to HPC, the ARM HPC software ecosystem is maturing at all levels: compilers, libraries, tools; and important systems are going to be based on ARM technology (www.nextplatform.com/2016/06/23/inside-japans-future-exaflops-arm-supercomputer, http://www.theregister.co.uk/2017/01/16/arm\_mont\_blanc\_hpc\_european\_commission/, www.arm.com/hpc). Looking at ARM mobile SoCs in particular, here are some works that show runs in real hardware and simulations with HPC applications:**

**-N. Rajovic et.al: The Mont-Blanc Prototype: An Alternative Approach for HPC Systems, SC16**

**-N. Rajovic et.al: “Tibidabo1: Making the case for an ARM-based HPC system”, FGCS2014**

**-N. Rajovic et.al: “Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?”, SC13**

**-N. Rajovic et.al: Experiences With Mobile Processors for Energy Efficient HPC, DATE’13**

**-Butko et.al: “Design Exploration For Next Generation High-Performance Manycore On-chip Systems: Application To big.LITTLE Architectures”, ISVLSI’15**

**-K. Chronaki et.al.: Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures, ICS2015**

**-E.Castillo et.al: CATA: Criticality-Aware Task Acceleration for Multi-Core Processors, IPDPS2016**

**We can modify our introduction so that this becomes clear and does not confuse the reader about the usability of this paper.**

**R2&R3-PARSEC suitability:**

**We agree that, as both Reviewers noted, we could improve the motivation of this work in the introduction. PARSEC suite is the best candidate for such a preliminary evaluation. The mention of supercomputing and HPC is to show that previous works have already used mobile SoCs outside the mobile market. We selected PARSEC because it consists of real highly parallel applications that represent a wide range of workloads and are not tied to a particular field of study, making the study more relevant.**

**R3:**

**i.We agree that changing the baseline in some of the figures would help the reader.**

**R4:**

**i.Figure 4 is not normalized. It shows the average power in Watts. Figure 5 is normalized using as a baseline the energy with static threading using 4 little cores. We agree that this might be confusing so we plan to “homogenize” all figures and use the same baseline for all of them.**

**ii.If the task and chunk sizes are the same then the difference comes from the fact that loop-based dynamic scheduling requires a barrier at the end of each loop. A task-based implementation removes that barrier by using task-dependencies across loops/parallel regions, potentially overlapping work, resulting in less waiting. If the loop-based case is well-balanced, then, at same task and chunk sizes performance should be similar. The key insight is that dynamic task or loop scheduling is required in these systems. Our contribution is the detailed performance analysis of these options, considering software aspects such as task and chunk size, which seems relevant for a conference like ISPASS.**

**iii.The power ratio between big and little cores was calculated during this study and was from 2x to 2.5x depending on the application. As during the writing we saw that it didn’t add much insight to the reader we decided to omit it, but it is trivial to add it back on Table1. We don't have a definitive answer on why the power does not increase proportionally to the number of cores. We can speculate that, when adding asymmetry (little-cores), the task-based implementation gets much better utilization than the static case, but probably not 100%. This is hard to measure given the complex nature of the codes under evaluation. Our guess is that there is a small utilization drop of the big cores that is compensated by the addition of the little cores leading to a similar power than the only big-core scenario. The locality effects of adding a second cluster of cores with another L2 cache introducing potential invalidations in the big-core caches could be an explanation of the big-cores losing some utilization, even in the task-base case.**

**iv.We use the PARSECSs suite that contains the task-based versions of PARSEC. In this suite there are 10 of the original PARSEC benchmarks (two of them are not implemented using task-based). Out of these 10 benchmarks we exclude freqmine as it has no pthreads version to represent static threading and GTS.**

**Major-ii.From our experience with working on multiple frequency combinations, we saw that the outcomes of these studies do not differ between different settings. This is mainly because the most important reason behind the observed performance ratios of big and little is their micro-architecture (big=out-of-order, little=in-order). In this study we stick with the specific frequency combination because when we set the frequency of big cores to the maximum (2GHz), the machine was turned off due to high temperatures, as these applications were running for long times.**

**Minor-all:We will address all minor comments**

**-----------------------------------------------------------------------------------------------------------------------------**

**R1:**

**We are sorry that the reviewer cannot see the contribution of our work.**

**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 of real highly parallel applications.**

**Moreover, we consider this work ideal for the ISPASS conference, as it is a thorough evaluation and characterization of an architecture, showing the current obstacles and how to overcome them in the runtime or OS level.**

**R2:**

**Our work focuses on the examination of the new mobile-based asymmetric systems for their use in the next generation multi-cores. As the main challenge of the future multi-core system design is energy efficiency, many researchers are pushing towards the use of the energy efficient asymmetric multi-cores to build such systems and tackle the power wall. However, our work is not tied to a particular architecture. Our insights apply to asymmetric multi-cores in general. Regarding the applicability of ARM to HPC, the ARM HPC software ecosystem is maturing at all levels: compilers, libraries, tools; and important systems are going to be based on ARM technology (see https://www.nextplatform.com/2016/06/23/inside-japans-future-exaflops-arm-supercomputer/ and http://www.theregister.co.uk/2017/01/16/arm\_mont\_blanc\_hpc\_european\_commission/). More information is available here: www.arm.com/hpc. Looking at ARM mobile SoCs in particular, here are some works that show runs in real hardware and simulations with HPC applications:**

**-N. Rajovic et.al: The Mont-Blanc Prototype: An Alternative Approach for HPC Systems, SC16**

**-N. Rajovic et.al: “Tibidabo1: Making the case for an ARM-based HPC system”, FGCS2014**

**-N. Rajovic et.al: “Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?”, SC13**

**-N. Rajovic et.al: Experiences With Mobile Processors for Energy Efficient HPC, DATE’13**

**-Butko et.al: “Design Exploration For Next Generation High-Performance Manycore On-chip Systems: Application To big.LITTLE Architectures”, ISVLSI’15**

**-K. Chronaki et.al.: Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures, ICS2015**

**-E.Castillo et.al: CATA: Criticality-Aware Task Acceleration for Multi-Core Processors, IPDPS2016**

**We can modify our introduction so that this becomes clear and does not confuse the reader about the usability of this paper.**

**R2&R3-PARSEC suitability:**

**We agree that, as both Reviewers noted, we could improve the motivation of this work in the introduction. PARSEC suite is the best candidate for such a preliminary evaluation. The mention of supercomputing and HPC is to show that previous works have already used mobile SoCs outside the mobile market. We selected PARSEC because it consists of real highly parallel applications that represent a wide range of workloads and are not tied to a particular field of study, making the study more relevant.**

**R3:**

**i.We agree that changing the baseline in some of the figures would help the reader.**

**R4:**

**i.Figure 4 is not normalized. It shows the average power in Watts. Figure 5 is normalized using as a baseline the energy with static threading using 4 little cores. We agree that this might be confusing so we plan to “homogenize” all figures and use the same baseline for all of them.**

**ii.If the task and chunk sizes are the same then the difference comes from the fact that loop-based dynamic scheduling requires a barrier at the end of each loop. A task-based implementation removes that barrier by using task-dependencies across loops/parallel regions, potentially overlapping work, resulting in less waiting. If the loop-based case is well-balanced, then, at same task and chunk sizes performance should be similar. The key insight is that dynamic task or loop scheduling is required in these systems. Our contribution is the detailed performance analysis of these options, considering software aspects such as task and chunk size, which seems relevant for a conference like ISPASS.**

**iii.The power ratio between big and little cores was calculated during this study and was from 2x to 2.5x depending on the application. As during the writing we saw that it didn’t add much insight to the reader we decided to omit it, but it is trivial to add it back on Table1. We don't have a definitive answer on why the power does not increase proportionally to the number of cores. We can speculate that, when adding asymmetry (little-cores), the task-based implementation gets much better utilization than the static case, but probably not 100%. This is hard to measure given the complex nature of the codes under evaluation. Our guess is that there is a small utilization drop of the big cores that is compensated by the addition of the little cores leading to a similar power than the only big-core scenario. The locality effects of adding a second cluster of cores with another L2 cache introducing potential invalidations in the big-core caches could be an explanation of the big-cores losing some utilization, even in the task-base case.**

**iv.We use the PARSECSs suite that contains the task-based versions of PARSEC. In this suite there are 10 of the original PARSEC benchmarks (two of them are not implemented using task-based). Out of these 10 benchmarks we exclude freqmine as it has no pthreads version to represent static threading and GTS.**

**Major-ii.From our experience with working on multiple frequency combinations, we saw that the outcomes of these studies do not differ between different settings. This is mainly because the most important reason behind the observed performance ratios of big and little is their micro-architecture (big=out-of-order, little=in-order). In this study we stick with the specific frequency combination because when we set the frequency of big cores to the maximum (2GHz), the machine was turned off due to high temperatures, as these applications were running for long times.**

**Minor-all:We will address all minor comments**

**R1:**

We are sorry that the reviewer cannot see the contribution of our work.

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 of real highly parallel applications.

Moreover, we consider that this work is ideal for the ISPASS conference, as it is a thorough evaluation and characterization of a new architecture showing the current obstacles and how to overcome them in the runtime or OS level.

**R2:**

Is ARM a good choice for parallel HPC or data analytics applications? Please give evidence (existing publication or empirical results) showing that the choice is overall appealing in terms of performance, energy, and usability.

Our work focuses on the examination of the new mobile-based asymmetric systems for their use in the next generation multi-cores. As the main challenge of the future multi-core system design is energy efficiency, many researchers are pushing towards the use of the energy efficient asymmetric multi-cores to build such systems and tackle the power wall. However, our work is not tied to a particular architecture. Our insights should apply to asymmetric multi-cores in general. Regarding the applicability of ARM to HPC, the ARM HPC software ecosystem is maturing at all levels: compilers, libraries, tools; and important systems are going to be based on ARM technology (see <https://www.nextplatform.com/2016/06/23/inside-japans-future-exaflops-arm-supercomputer/> and <http://www.theregister.co.uk/2017/01/16/arm_mont_blanc_hpc_european_commission>/). More information is available here: www.arm.com/hpc. Looking at ARM mobile SoCs in particular, here are some works that show runs in real hardware and simulations with HPC applications:

- N. Rajovic et.al: The Mont-Blanc Prototype: An Alternative Approach for HPC Systems, SC16

- N. Rajovic et.al: “Tibidabo1: Making the case for an ARM-based HPC system”, Future Generation Computer Systems, Volume 36, July 2014, Pages 322-334

- N. Rajovic et.al: “Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?”, SC13

- N.Rajovic et.al: Experiences With Mobile Processors for Energy Efficient HPC, DATE’13

-Butko et.al: “Design Exploration For Next Generation High-Performance Manycore On-chip Systems: Application To big.LITTLE Architectures”, ISVLSI’15

-K. Chronaki et.al.: Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures, ICS2015

-E.Castillo et.al: CATA: Criticality-Aware Task Acceleration for Multi-Core Processors, IPDPS2016

What the reviewer is asking for motivation is actually the purpose of this work. We don’t claim that these architectures are ready to face HPC applications. However this is what our study is about as there is a long way for ARM to HPC. In our work we draw interesting conclusions towards this as the exploration of the most appropriate scheduling approach and the generic behavior of such benchmarks.

We can modify our introduction so that this becomes clear and does not confuse the reader about the usability of this paper.

**R2 & R3: (PARSEC suitability issue)**

We agree that, as Reviewers 2 and 3 noted, we could improve the motivation of this work in our introduction section and we plan to add this. PARSEC suite is the best candidate for such a preliminary evaluation. The mention of supercomputing and HPC is to show that previous works have already used mobile SoCs outside the mobile market. We selected PARSEC because it consists of real highly parallel applications that represent a wide range of workloads and are not tied to a particular field of study, making the study more relevant.

**R3:**

In a lot of the figures the baseline is relative to 1 little core yet for the investigation the base system is different. For example figure 6 where the starting system is 4 large cores and the object of the study is to determine if adding little cores is beneficial. The graph would be easier to interpret if either a line of the real baseline (e.g. 4 big cores) was drawn across so that one can see relative improvement or not OR just use the 4 big core case as the baseline to normalize too.

We agree that changing the baseline in some of the figures would help the reader. We can modify these figures accordingly.

They show results for power, performance, and EDP. Improving the motivation for the work would make the paper stronger by showing the application/workload that is the target of this AMC architecture. For example there is mention of supercomputing and mobile market but the PARSEC benchmark suite is really neither.

We agree that, as also Reviewer 2 noted, we could improve the motivation of this work in our introduction section and we plan to add this. PARSEC suite is the best candidate for such a preliminary evaluation. The mention of supercomputing and HPC is to show that previous works have already used mobile SoCs outside the mobile market. We selected PARSEC because it consists of real highly parallel applications that represent a wide range of workloads and are not tied to a particular field of study, making the study more relevant.

**R4:**

i. What is the baseline system for Figure 4? The columns corresponding to a system with 0 big cores and 4 small cores and static threading are not equal to 1 (In Figure 5, all columns for the same system are 1).

i. Figure 4 is not normalized. It shows the average power in Watts. Figure 5 is normalized using as a baseline the energy with static threading using 4 little cores. We agree that this might be confusing so we plan to “homogenize” all figures and use the same baseline for all of them.

ii. It is not clear why exactly the task-based approach is better for smaller sized work chunks and dynamic scheduling is better for coarse-grained applications. How do the two compare when the task sizes for the task-based approach and the chunk sizes used for dynamic scheduling are the same?

ii. If the task and chunk sizes are the same then the difference comes from the fact that loop-based dynamic scheduling still has a barrier at the end of each loop. A task-based implementation could remove that barrier using task-dependencies across loops/parallel regions, potentially overlapping work, resulting in less waiting. If the loop-based case is well-balanced, then, at same task and chunk sizes performance should be similar. The key insight is that dynamic scheduling is a hard requirement in these systems, whether it is based on tasks or loops. Our contribution is the detailed performance analysis of these options, also considering software aspects such as task and chunk size, which seems relevant for a conference like ISPASS.

The task-based approach offers flexibility in the parallelization as well as during execution. The dependence analysis mechanisms that the task based approach offers is able to speedup execution compared to loop dynamic scheduling that uses barrier synchronization between the parallel loops. The task sizes are similar to the chunk sizes as they consist of the same piece of code.

iii. In figure 8, since the task-based approach keeps all available cores (both big and small) busy, shouldn't the average power consumption increase as the number of small cores are increased? What is the power ratio for the big and small cores for the evaluated applications? Updating Table 1 with this information will be helpful for readers.

iii. The power ratio between big and little cores was calculated during this study and was from 2x to 2.5x depending on the application. As during the explanation of the results we saw that it didn’t add much of insight to the reader we decided to omit it, but it is trivial to add it back on Table 1. We don't have a definitive answer on why the power does not increase proportionally to the number of cores. We can speculate that, when adding asymmetry (little cores), the task-based implementation gets much better utilization than the static case, but probably not 100%. This is hard to measure given the complex nature of the codes under evaluation. Our guess is that there is a small utilization drop of the big cores that is compensated by the addition of the little cores leading to a similar power than the only big-core scenario. The locality effects of adding a second cluster of cores with another L2 cache introducing potential invalidations in the big core caches could be an explanation of the big cores losing some utilization, even in the task-base case.

The asymmetry of the system helps on maintaining the power dissipation on the same levels even when adding little cores to the system. As the workload is balanced among the cores, the high power dissipation of big cores is reducing and the dissipation of little cores is increasing. So as a result the average power remains stable. Contrarily, with static threading avg power dissipation is being reduced because little cores get the same workload as big cores.

iv. Why were only 9 of the 13 Parsec benchmarks used in the evaulation?

iv. We use the PARSECSs benchmark suite that contains the task-based versions of PARSEC. In this suite there are 10 of the original PARSEC benchmarks (two of them are not implemented using task-based). Out of these 10 benchmarks we exclude freqmine as it has no pthreads version to represent static threading and GTS.

Major comments:

ii. The paper provides results for only one combination of frequencies of the big and small cores (big cores at 1.6GHz, small cores at 800MHz). Other frequency combinations may show different behavior for some benchmarks, please include results for other combinations (at least for section V.A) as well.

From our experience with working on multiple frequency combinations, we saw that the outcomes of these studies do not differ between different settings. This is mainly because the most important reason behind the observed performance ratios of big and little is their micro-architecture (big=out-of-order, little=in-order). In this study we stick with the specific frequency combination because when we set the frequency of big cores to the maximum (2GHz), the machine was turned off due to high temperatures, as these applications were running for long times.

Minor comments:

We plan to address all minor comments