**Reviewer #1: Summary**

This paper evaluates power, performance, energy, EDP of different scheduling strategies on a real big/little architecture while running parallel workloads (based on different programming models) in order to understand their effectiveness in the context of Asymmetric Multi-Cores. The evaluation demonstrates that GTS (OS level) and Task-based (runtime level) scheduling strategies perform better than static scheduling strategies and that task-based scheduling provides the most performance improvement among the strategies evaluated.

Comments

- 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**

- 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?

- 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.

- 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?

- 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**

- 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?

- 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.

- 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 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?

- 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?

- 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.

- 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 indicated in the discussion that loop-dynamic is more efficient on coarse grained parallel applications than task-based scheduling. Why is this the case?

Minor Comments

- It would be good to have the average speedup numbers in Figure 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.

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.

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.

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

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.

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.

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**

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. **🡪 overhead is less than 3%**

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.

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.

11. The authors use a lot of space for introducing PARSEC, but most readers will be familiar with it. This discussion should be shortned.

12. Figure placement needs to be improved. A lot of figures are placed quite far away from where they are discussed which reduces readability.

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.

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.**