Some Ideas and Principles for Achieving Higher System Energy Efficiency: Slack-Based, Coordinated, Hardware/Software Cooperative Management of Heterogeneous Resources

Onur Mutlu, Carnegie Mellon University (onur@cmu.edu, 412-268-1186, `http://www.ece.cmu.edu/~omutlu`)

This whitepaper briefly describes *some* broad ideas, principles, and research directions that seem promising, in the author's opinion, to achieve a system that is overall more energy efficient (and also higher performance) than today's systems.

Energy is performance - be frugal: In today's systems, which are overwhelmingly power-limited, energy is performance. If one can do the same "task" with less energy, one can do more of the same *task* or other *tasks* by spending the same amount of energy as before, increasing system throughput.[1] Therefore, doing *the same task at the same performance but more efficiently* is key to improving system performance as well as energy. Hence, we should design building blocks (e.g., functional units, cores, caches, libraries, software components) that are as simple and efficient as possible without significantly sacrificing performance.

Not all *tasks* are equal - spend as much energy on a *task* as it really needs: Not all *tasks* that need to be done are equally important. For example, not all instructions are equally critical to system performance [5]. Not all network packets or memory requests have the same criticality [2, 3]. Not all cache blocks are equally important [11]. Not all threads limit performance [1, 4, 6]. Not all queries/requests in a server need to be serviced fast. Not all threads or programs are of the same importance. Efficient systems should identify criticality and bottlenecks at different levels of processing (instruction, request, task, program, system) and accelerate/prioritize critical/bottleneck *tasks* while spending little energy on non-critical *tasks*. This could be done by 1) identifying the "slack" present in each *task* and 2) designing resources and 3) managing execution such that the slack of *every task* is always as close to zero as possible. All three are promising and unsolved research directions.

Consolidate many *tasks* on shared hardware resources while providing the QoS each *task* needs: Efficient utilization of hardware resources can be achieved by consolidating more *tasks* on the same hardware to exploit idleness of resources. However, consolidation leads to contention and potential starvation, leading to inefficiency. Therefore, efficient systems should incorporate mechanisms that provide QoS to different *tasks*. A unifying principle to designing such mechanisms is the identification of the slack/criticality/latency-sensitivity each *task* has and allocating more resources to those *tasks* that have little slack. We have recently shown that 1) identifying the slack of network-on-chip packets and prioritizing packets with lower slack [2], 2) identifying limiter threads in parallel applications using cooperation between the hardware and the runtime system and prioritizing such threads in the memory controller [4], 3) identifying latency-sensitive applications and prioritizing them in the memory scheduler [7] can yield significant performance improvements. We believe these techniques can also improve energy. Many other such opportunities exist to manage resources using the concepts of slack, criticality, and latency-sensitivity.

Coordinate the management of shared resources: If one resource prioritizes a critical *task* while another deprioritizes it, system efficiency degrades. Similarly, if one resource implements a technique that is aimed to improve performance/efficiency (e.g., the core generates multiple concurrent memory requests assuming they will be serviced in parallel by the memory system) while another employs policies that destroys the benefits of the technique (e.g., the memory controller serializes the memory requests [10]), system efficiency degrades. For best efficiency, therefore, different resources should work in tandem with each other: policies employed in cores, caches, interconnects, and memory controllers should be co-designed or made aware of each other. As we have shown in recent work, designing the interconnect packet scheduling policy [2] and memory controller scheduling policy [10, 8] such that they are aware of the cores' policies, and designing the last-level cache policies such that they are aware of memory controller policies [9] can significantly improve performance. We believe much more potential exists in coordinated optimization of policies across resources.

Exploit architectural and technology heterogeneity: Heterogeneous or partially reconfigurable components (not only processing elements, but also shared resources with different power/performance tradeoffs and fitting different computation/access/communication patterns/needs) are essential to enable power-efficiency using automatic resource management. If components are the same, energy and performance of a *task* will be suboptimal regardless of on what component a *task* is executed. Heterogeneous components enable customization opportunities such that a *task* can be executed on the best-fit resource, i.e. one that executes the *task* most efficiently while satisfying its performance requirements. We posit that not only cores, but also interconnects, caches, memory controllers, and entire memory hierarchies can be designed in a heterogeneous manner and the runtime system can be designed to automatically choose the best-fit components for each *task* dynamically. Management of heterogeneity can be accomplished using the concepts of slack, criticality, and latency-sensitivity such that *tasks* with low slack and high criticality/latency-sensitivity are allocated higher performance and higher power resources whereas non-critical *tasks* are allocated low-power resources, as we have shown in recent work [12, 6].

Coordinate hardware and software: To accomplish all of the above efficiently, we strongly believe hardware/software cooperation is essential. Work should be divided across the layers such that the overall system is the most efficient. Examples abound in both our and others' past works. Many future research opportunities exist in all of the above directions.

[1] A. Bhattacharjee et al. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *ISCA-36*, 2009.

[2] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Aergia: Exploiting packet latency slack in on-chip networks. In *ISCA-37*, 2010.

[3] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Prefetch-aware shared resource management for multi-core systems. In *ISCA-38*, 2011.

[4] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, O. Mutlu, and Y. N. Patt. Parallel application memory scheduling. In *MICRO-44*, 2011.

[5] B. Fields, S. Rubin, and R. Bodik. Focusing processor policies via critical-path prediction. In *ISCA-28*, 2001.

[6] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt. Bottleneck identification and scheduling in multithreaded applications. In *ASPLOS-XVII*, 2012.

[7] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO-43*, 2010.

[8] C. J. Lee et al. Improving memory bank-level parallelism in the presence of prefetching. In *MICRO-42*, 2009.

[9] C. J. Lee et al. DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems. Technical report, UT-Austin, 2010.

[10] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA-35*, 2008.

[11] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for MLP-aware cache replacement. In *ISCA-33*, 2006.

[12] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS-XIV*, 2009.

[1] A "task" can be thought of an arbitrary unit of work, for the purposes of this whitepaper. It can be an instruction, a memory request, a function, a thread, a collection of threads, an entire application, a collection of applications, etc. We believe the principles outlined to improve energy efficiency is fundamental across these different entities, which we will refer to as *tasks*.