Workload characterization for runtime memory management in emerging embedded platforms *

CS 237 Final Project - Biswadip Maity

University of California, Irvine

Abstract. Memory has emerged as a primary bottleneck for emerging embedded platforms that integrate heterogeneous compute units, and where the demands for balance between performance and energy efficiency become challenging. In the face of dynamic workloads, there are many opportunities to manage the memory subsystem efficiently at runtime to save energy without compromising quality. Previous works have used memory bandwidth utilization to determine memory requirements and develop runtime policies to configure system knobs (e.g., memory controller frequency) accordingly. However, bandwidth utilization as a singular metric is not always sufficient: policies for a range of workload scenarios require insight into an application's memory access pattern and working set size. Alternatively, memory profilers provide fine-grained information such as the memory access pattern for the entire virtual address space, and the load/store density of different regions of the memory. However, parsing this detailed information frequently at runtime induces excessive overhead. In this work, we propose a runtime profiling mechanism that considers both (1) the working set size of running workloads and (2) memory bandwidth utilization and computes the WBP (Working Set Size-Bandwidth Product). WBP can be estimated with low overhead, and the combined metrics provide insights which runtime policies can use to decide the system configuration for specific workload scenarios. Our early results show that a static configuration devised with this metric yields an optimal memory controller frequency for 8 out of 10 PARSEC workloads, demonstrating the promise of this approach.

Keywords: Main Memory · Memory bandwidth · Runtime Memory Management.

1 Introduction

Contemporary embedded systems are equipped with heterogeneous compute units with shared main memory to meet the varying memory capacity and bandwidth requirements of modern applications (e.g., Nvidia Tegra, Nvidia Xavier, AMD Accelerated Processing Unit). All computational resources (e.g., CPU, GPU, DSP, on-chip accelerator) share the main memory, resulting in memory

* Submitted to IESS 2019 with co-authors Bryan Donyanavard, Nalini Venkatasubramanian and Nikil Dutt

continuing to be the major performance and energy bottleneck for emerging embedded system platforms. To address this bottleneck, application developers use memory profilers to understand the runtime requirements of applications and attempt to reduce their memory footprint through code optimization.

At runtime, resource managers monitor the system state and implement policies to update the configuration (i.e., knobs) to meet the system's goals (e.g., maximize performance-per-unit-power). Traditionally, policies that target memory exploit memory bandwidth under-utilization at runtime by dynamically scaling the memory controller frequency in order to reduce power without compromising quality. However, embedded systems commonly support many hardware and software knobs which: (1) schedule applications differently, (2) set the frequency of heterogeneous compute units, and (3) set the degree of parallelism by changing the number of threads; all of which can affect the time required to process application data and, in turn, energy consumption.

Although memory profilers report the number of memory accesses (load/stores) and generate heatmaps (i.e., memory access density) for the profiled application, it is difficult for policies to manage the overhead of parsing this information at runtime. Furthermore, the virtual address space is huge and must refer to the page map to understand the physical location of data accesses. The overhead of analyzing the information from memory profilers makes the information impractical to consider for high-frequency runtime decision making. Thus, it becomes essential to monitor a coarse-grained metric(s), which can assist the runtime resource managers in efficiently determining the ideal system configuration.

To address these challenges, we propose a memory-driven application classification based on multiple dimensions in order to assist in developing policies for resource allocation strategies, specifically for DRAM. We combine (a) the size of memory required (in MBs) of the working set of applications, and (b) the runtime DRAM bandwidth requirement (last-level cache miss rate) to classify an application's memory requirement. Both measurements can be made with minimal overhead and can be used at runtime to evaluate the dynamic requirements of applications. In this work, this classification is used to determine static memory controller frequency to minimize the Energy-Delay Product (EDP). However, this classification can be further used to develop policies which decide the system configuration based on the classes of the application currently running on the system.

2 Motivation

In 1965 Gordon E. Moore predicted that manufacturers would keep doubling the number of transistors on a chip almost every two years [11]. The developments of the semiconductor industry by leaps and bounds have introduced billions of transistors on tiny dies. The law has also set the pace for software running on the underlying hardware. Small computers which are present in most households devices (e.g., Smartwatches, Smart-lights, Mobile devices) and personal computers are now equipped more resources than can be powered. With the growth in em-

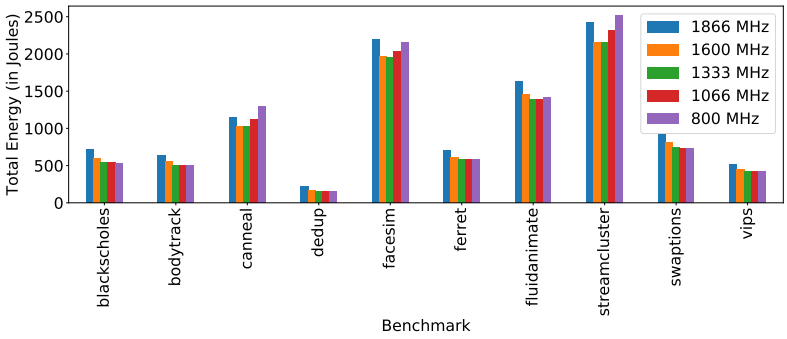

Fig. 1. Total energy consumed as memory controller frequency is varied statically from 800MHz to 1866MHz for PARSEC workloads on the Jetson TX2. The optimal frequency which results in the minimum energy spent is not always the lowest (800MHz).

bedded compute resources, applications have now become extremely demanding. Applications generate data at a rate which poses challenges to the main memory latency and bandwidth. Thus, the performance and energy bottleneck in today's embedded system platforms is no longer the compute resource itself, but processing the data efficiently. One current challenge for computing systems is that the processing of data is performed far away from the data [9, 7].

Although caching is traditionally applied to address this challenge, the working set size of applications may be significant enough to exceed the cache capacity. The growth in working set size leads to cache misses and requires main memory accesses to fetch data for the processor. The movement of data from the DRAM to the processor consumes 19% of total system power [3] and cache miss takes more than 50% of the cycles [7]. Previous works have used memory bandwidth utilization to determine memory requirements and develop runtime policies to address the issue of memory energy consumed. [3] proposes to set the memory controller frequency at the lowest frequency (i.e., 800MHz) when the memory is lightly loaded (memory bandwidth is under 2000 MB/s).

Figure 1 shows the effect of memory controller frequency on total energy consumption for a variety of benchmarks executing on a contemporary embebedded platform, the Jetson TX2. A single instance of 10 workloads from the PARSEC benchmark suite [2] is executed on the Jetson TX2 [1] with fixed memory controller frequency at various levels, and a runtime system calculates the total energy spent. The first observation we make about the workloads executed in Figure 1 is the memory bandwidth utilization is low: it never exceeds 430 MB/s. According to the previously described policy, one would expect the lowest frequency to be the best configuration for the system. However, the lowest frequency leads to the minimum energy consumption for only 2 out of 10 workloads. When operating at the lowest frequency, although the power consumption

is minimal, the extended execution time results in increased energy consumption in 8 out of 10 cases. Even from this simple instance we can conclude that in order to account for additional energy consumption due to excess latency, we must consider more than just bandwidth when a policy decides on the frequency to be assigned for the embedded platform.

3 Diversity of working set size in embedded applications

Denning defines the working set of an application as a collection of recently referenced pages of an application's virtual address space[5]. Working set is an efficient tool to measure the memory demands of an application. A similar metric, the resident set, is defined as the subset of an application's segments which is present in the main memory at a given time. In 2015, Kanev *et. al.* [7] analyzed all of Google's Data Center workloads, and the results show that working set sizes for applications are growing rapidly and this results in a negative impact on the application's performance. Large working set sizes lead to expensive cache misses, and thus provides a good insight into the applications memory footprint at runtime.

3.1 PARSEC benchmark suite

The PARSEC benchmark suite [2] is a popular suite for evaluating multiprocessor-based embedded platforms. It consists of 12 workloads (nine applications and three kernels) from recognition, mining, and synthesis (RMS) domain, as well as representative system applications. The workloads compose a diverse representative of working set size, degree of parallelism, off-chip traffic, data-sharing that can be representative of the diverse workloads executed on emerging embedded platforms. Based on the working set, two broad classes of workloads are distinguished:

1. **Working set smaller than 16MB**: These applications have limited requirement for large cache size, and the working set can fit in the last level shared cache. Example applications are `bodytrack` and `swaptions`.
2. **Working set larger than 16MB**: These applications have a very large working set. Even with large caches cannot meet the requirements for this group without off-chip memory access to the DRAM. Example applications are `canneal`, `ferret`, `facesim`, `fluidanimate`. When the input size grows, the working set can even reach gigabytes due to algorithms that operate on large amounts of collected data.

Note as the number of cores increases with the degree parallelism, so does the bandwidth requirement. `bodytrack` makes off-chip memory accesses in short, but bandwidth-intensive bursts. When several instances of the same application execute concurrently, these short bursts limit the scalability.

The sampling of benchmarks from PARSEC demonstrate highly variable memory requirements that depend on multiple factors. The number of load/store

operations, size of the last-level cache, and the number of parallel threads are some of the factors that affect the bandwidth. These various sources of memory bottleneck provide opportunities for resource managers to address the bottleneck at runtime for unpredictable workloads. Working set can effectively represent some of these dynamics, and we show that when combining the working set size information along with the memory bandwidth at runtime can lead to efficient system configurations.

4 Related work

Current runtime memory management techniques which contribute to the runtime decision making are discussed in this section.

4.1 Runtime policies using Memory Bandwidth Utilization

As emerging embedded systems move towards multiple heterogeneous compute units sharing the main memory, the memory subsystem continues to dominate as the major performance and energy bottleneck. Several techniques have been employed to address this bottleneck, and *Dynamic Voltage and Frequency Scaling (DVFS)* of the memory controller is a primary one[4]. Higher frequency yields higher throughput but consumes more power. DVFS can reduce the memory power by 10.4% on average, 20.5% maximum for the SPEC 2006 benchmarks as shown in [3] without compromising on quality. [8] uses an approximation equation (called MAR-CSE) based on the correlation of the memory access rate and the critical speed for the minimum energy consumption. The memory access rate is the ratio of the total number of cache misses (including both instruction and data cache misses) to the number of instructions executed. MAR-CSE predicts the voltage and frequency at runtime. Although this technique does not measure the memory bandwidth directly, the memory access rate is also an indirect measure of the memory bandwidth.

4.2 Runtime policies using Workload information

Recent work [10] looked into the combined effect of application compute/memory-intensity, thread synchronization contention, and nonuniform memory accesses pattern to develop a runtime energy management technique by performing DVFS on CPU cores. However, they do not consider memory bandwidth utilization and do not change the memory controller frequency.

4.3 Runtime policies using reflection or prediction

Most compute units are configurable to run at different frequencies. At runtime, based on the current resource requirement, the manager needs to decide the optimal operating point to meet the goals of the system. CPU DVFS has been explored extensively in literature [13] and most systems use it to meet system

goals. Linux provides several governors to manage the CPU frequency at runtime (e.g. `ondemand`, `performance`, `powersave`). Recent work uses reflective system model to predict the system behavior when different frequencies are selected[6], [12].

4.4 Runtime policies using Task Mapping

The Operating System scheduler has a responsibility of (1) Deciding the schedule of running applications and (2) Deciding which threads run on which cores. Modern systems provide heterogeneous compute resources which have different power/performance. The smaller cores are more power efficient but have less computational capacity than the bigger cores. Task mapping has shown [14] energy saving of up to 51% when combined with DVFS.

5 Evaluation

5.1 Methodology

The proposed approach combines the Working Set Size and memory bandwidth by calculating their product (WBP). The objective is to find the operating frequency, which leads to the lowest Energy-Delay Product (EDP). Results show that applications belong to one of three classes.

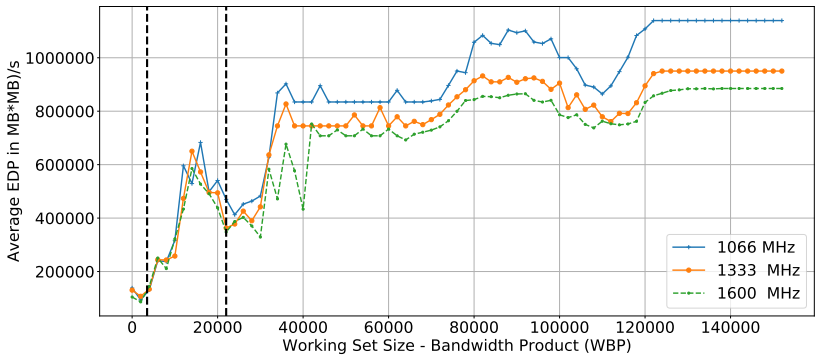

Fig. 2. Change in average EDP (Energy-Delay Product) of PARSEC workloads across frequencies at different values of WBP (Working Set Size - Bandwidth Product).

We conduct experiments to observe the correlation between WBP and EDP. 10 PARSEC benchmarks are executed with different memory controller frequencies. At regular intervals (*sensing windows*) of 200ms, the following metrics are recorded:

1. **Working Set Size**: Linux versions 4.3 introduced idle page flags to track memory utilization. Once a process starts, the idle bits corresponding to all the virtual pages in that process are set to 1 to indicate that they have not been referenced. Whenever the process issues memory read/write requests, the idle bit corresponding to the virtual page is set to 0 by Linux kernel. A 0 implies that the page is not idle. We selected the window size as 200ms, however exploring more window durations and feasibility of adaptive windows remain as future work. Every 200ms, the number of 0 bits in the idle page flags are read and used to calculate the working set size as 0 as follows:

$$WorkingSetSize = \text{Number of active pages} \times \text{Size of each page}$$

2. **Memory Bandwidth**: ARM cores include logic to gather various statistics on the operation of the processor and memory system during runtime based on a Performance Monitoring Unit (PMU). PMU provides hardware counters for different events, which is used to profile application behavior. The counter values of `L2_CACHE_REFILL` and `MEM_ACCESS` on each core is monitored to understand the memory traffic at runtime every 200ms (Sensing Window Length = 0.2s). Since the L2 is the last level shared cache on the Jetson, the memory bandwidth can be calculated using these values as:

$$MemoryBandwidth = \frac{\sum_{i=1}^{activeCores} \texttt{L2\_CACHE\_REFILL}_{Core\_i} \times DataBusSize}{SensingWindowLength}$$

3. **Memory Power and System Power**: Nvidia drivers read the power measurements from onboard sensors connected by `I2C`. The separate domains for system power and memory powers help understand the energy consumed by the memory separate from the rest of the system.

4. **Latency/Delay**: The amount of time that the workload took to complete is indicative of the compute and memory latency. The CPU governor is set to 'userspace' and the frequency of all CPU cores is set to maximum for all the experiments. Thus, changes in memory latency due to the change in memory controller frequency is reflected in the total execution time.

Our objective is to find the operating frequency, which leads to the lowest Energy-Delay Product (EDP). To that effect, we are interested in finding changes in EDP across frequencies at different values of WBP (Working Set Size - Memory Bandwidth Product). EDP changes for PARSEC workloads across different frequencies are presented in Figure 2. At low WBP, the EDP is not affected by different frequencies. This is because the working set size of the application is small, and the memory requests are served from cache. Thus, operating at lower frequency does not have any effect on the EDP. As the WBP increases, memory requirement increases, and higher frequencies perform better. Frequencies lower than 1066 MHz (e.g., 800 MHz) have a very high latency whereas frequencies higher than 1333 MHz (e.g., 1866 MHz) consume too much power during the execution. Thus, they never obtain optimal EDP for any of the workloads. Hence, frequencies 800 MHz and 1866 MHz are not included in Figure 2.

Table 1. Classification of PARSEC workloads based on average WBP (Working Set Size - Memory Bandwidth Product) during runtime.

Workload	WBP (in MB^2/s)	Class	EMC Frequency
`dedup`	2100	C_1	1066 MHz
`swaptions`	3000	C_1	1066 MHz
`bodytrack`	3600	C_2	1333 MHz
`ferret`	3700	C_2	1333 MHz
`blackscholes`	5100	C_2	1333 MHz
`vips`	9700	C_2	1333 MHz
`fluidanimate`	16000	C_2	1333 MHz
`canneal`	22000	C_3	1666 MHz
`facesim`	62222	C_3	1666 MHz
`streamcluster`	119000	C_3	1666 MHz

An approach which classifies applications based on their WBP profile is proposed. For each class, the goal is to select a memory controller frequency, which leads to the minimum EDP. Two thresholds for WBP are selected based on their EDP profile at different frequencies. In this work, the operating frequency is determined statically based on the average WBP profile of workloads. However, we acknowledge the possibility of a runtime policy which checks the WBP at regular intervals to change the frequency during application execution. The classification and thresholds proposed are:

1. C_1 (Small memory footprint): $0MB^2/s <= WBP < 3500MB^2/s$ These applications have a low working set size and a low memory bandwidth. The size of L2 cache is large enough to accommodate most of the requests to the memory which the working set references for this class of application. When running C_1 applications, **the system is configured at 1066 MHz**.
2. C_2 (Medium memory footprint): $3500MB^2/s <= WBP < 22000MB^2/s$ These applications have moderate memory requirement. The L2 cache cannot accommodate all the requests to the memory. The working set size is also considerable and generates requests which need to go to main memory due to limited last level cache. When running C_2 applications, **the system is configured at 1333 MHz**.
3. C_3 (Large memory footprint): $WBP > 22000MB^2/s$ These applications have a high memory requirement. Operating the system at 1600 MHz clearly gives the lowest EDP. The working set size for this class of application is large and spread out to different regions in the memory. This incurs a lot of cache misses and generates requests to the main memory. When running C_3 applications, **the system is configured at 1600 MHz**.

5.2 Experimental Setup

We use Jetson TX2 [1] from Nvidia, an embedded System-On-Chip (SOC) platform to evaluate our proposed technique. The Jetson has heterogeneous compute cores (quad-core ARM Cortex A57 and dual-core Nvidia Denver2) distributed in two clusters along with an onboard 256-core Pascal GPU. Each of the clusters has separate frequency domains. Jetson has a shared memory architecture, and all the resources (CPU clusters, GPU) share the main memory. Jetson TX2 has an 8GB 128-bit LPDDR4 memory and a 32GB eMMC 5.1 for onboard storage. The Cortex cores come with: 48KB L1 instruction cache (I-cache) per core; 32KB L1 data cache (D-cache) per core. The Denver cores have 128KB L1 I-cache per core; 64KB L1 D-cache per core. All the cores share an L2 Unified Cache of 2MB.

Jetson uses an External Memory Controller (EMC) to manage the off-chip memory traffic. The EMC has different operating frequencies ranging from `4800KHz` to `1866MHz`. The onboard ARM Cortex Real-time (R5) Boot and Power Management Processor (BPMP) changes the memory controller frequency through kernel drivers. During all the experiments, the Denver cores are switched off, and the Cortex A57 cores are configured at the highest frequency to isolate the effect of the EMC operating frequency.

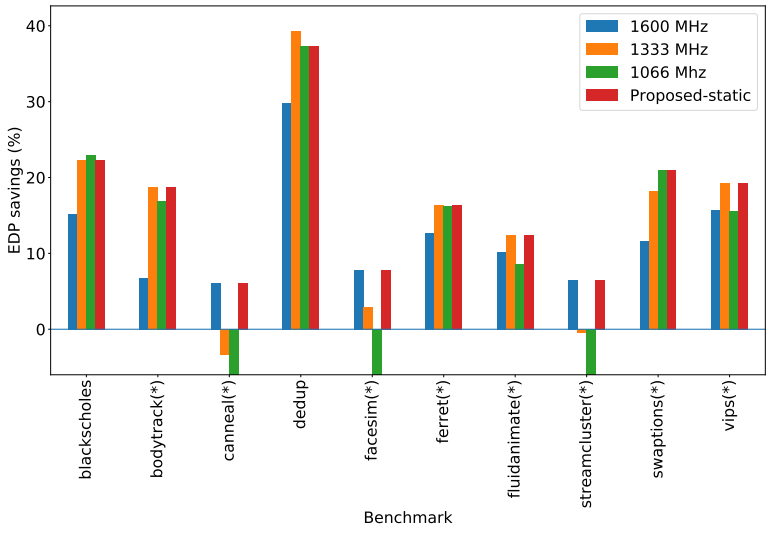

Fig. 3. EDP reduction with proposed classification using WBP compared to static-frequency baselines. Applications marked with a * have an optimal static configuration with the proposed scheme.

5.3 Experimental results

Ten PARSEC benchmarks are classified into one of the three classes presented in Section 5.1. Based on the proposed memory controller frequency for each class, each benchmark is executed with a static configuration. The results are presented in Figure 3. We expect to see lower EDP with the proposed approach when compared to techniques which do not use the memory information when deciding the EMC Frequency.

The average WBP (Working Set Size - Memory Bandwidth Product) at runtime for PARSEC benchmarks are presented in Table 1. Every 200ms, the working set size and memory bandwidth are measured using the Linux idle page tracker and `L2_CACHE_REFILL` PMU event counter as described in Section 5.1. The EDP values obtained with this configuration is compared with other static configurations in Figure 3. From the results, we see that the proposed scheme can find the optimal configuration for eight out of ten PARSEC workloads. The proposed scheme can achieve on average 16.7% (max: 37.3%) reduction in EDP when compared to the most aggressive memory controller frequency (1866 MHz). The results are compared with an optimal scheme obtained by executing all workloads at all possible EMC frequencies and choosing the frequency that yields minimum EDP. The optimal scheme can achieve an average reduction of 17.1% (max: 39.3%) EDP when compared to the most aggressive memory controller frequency (1866 MHz).

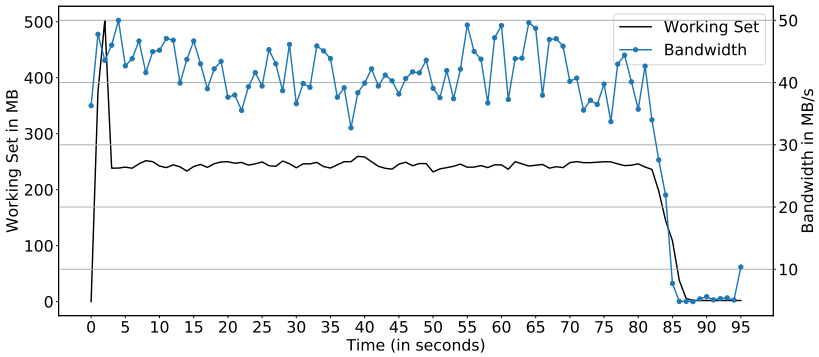

Fig. 4. Runtime memory profile of `blackscholes`. Dynamic memory access pattern calls for the exploration of a runtime policy.

6 Conclusion and Future Work

With the evolution of more heterogeneous embedded computing platforms that share a common main memory, the traditional memory bottleneck becomes even

more severe in the face of balancing performance/quality and energy efficiency. In this work, we presented a profiling metric, WBP (Working Set Size - Memory Bandwidth Product) that combines (1) memory bandwidth utilization and (2) working set size of running workloads. Based on the changes of EDP (Energy-Delay Product) with frequency at different WBP, we classify applications into three categories of memory requirement. Each class has a separate system configuration. Our initial results show that a static configuration correctly estimates the optimal frequency for 8 out of 10 PARSEC workloads, demonstrating the promise of our approach. The static scheme of setting memory controller frequency for each class can save EDP by 16.7% on average (37.3% maximum). However, benchmarks like `blackscholes` exhibit dynamic memory accesses patterns, as shown in Figure 4. A static configuration is not sufficient to address these bursts of memory accesses. Thus we are working on the next steps to perform DVFS adaptively/dynamically during application execution. Although the current work uses fixed sensing windows of 200ms, it may not be optimal, and different window sizes require exploration. Adaptive sensing window size along with runtime policies using the proposed classification technique remains as future work.

References

1. Nvidia Jetson TX2 Architecture. Available at https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.
2. Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite. In *PACT*, page 72, New York, New York, USA, 2008. ACM Press.
3. Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th International Conference on Autonomic Computing, ICAC 2011, Karlsruhe, Germany, June 14-18, 2011*, pages 31–40, 2011.
4. Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. Memscale: Active low-power modes for main memory. *SIGARCH Comput. Archit. News*, 39(1):225–238, March 2011.
5. P. J. Denning. Working sets past and present. *IEEE Trans. Softw. Eng.*, 6(1):64–84, January 1980.
6. Bryan Donyanavard, Tiago Mück, Santanu Sarma, and Nikil Dutt. Sparta: Runtime task allocation for energy efficient heterogeneous many-cores. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES '16, pages 27:1–27:10, New York, NY, USA, 2016. ACM.
7. Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 158–169, New York, NY, USA, 2015. ACM.
8. Wen-Yew Liang, Shih-Chang Chen, Yang-Lang Chang, and Jyh-Perng Fang. Memory-aware dynamic voltage and frequency prediction for portable devices. In

The Fourteenth IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2008, Kaohisung, Taiwan, 25-27 August 2008, Proceedings, pages 229–236, 2008.

9. Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt. Runahead execution: An alternative to very large instruction windows for out-of-order processors. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, HPCA '03, pages 129–, Washington, DC, USA, 2003. IEEE Computer Society.
10. Basireddy Karunakar Reddy, Eduardo Weber Wächter, Bashir M. Al-Hashimi, and Geoff V. Merrett. Workload-aware runtime energy management for HPC systems. In *2018 International Conference on High Performance Computing & Simulation, HPCS 2018, Orleans, France, July 16-20, 2018*, pages 292–299, 2018.
11. R. R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997.
12. V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.
13. Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.
14. D. Wu, B. M. Al-Hashimi, and P. Eles. Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems. *IEE Proceedings - Computers and Digital Techniques*, 150(5):262–, Sep. 2003.