Papers on Performance Optimization with Constrained Power Budget

1. A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems

[Published in ICS'05, Authors from IBM Austin Research Lab & University of Pittsburgh]

- This paper proposes power shifting which means dynamically dividing a system's power budget among its components so as to maximize performance
- Power shifting maintains the invariant that the dynamic component budgets always add up to exactly the system power budget
- Authors consider single-processor systems where they jointly manage the power of the processor and main memory
- Dynamic re-budgeting the available power between the processor and the memory subsystem
- Base power-control mechanism is a throttling scheme that limits the number of operations (a threshold) performed by each subsystem (instruction dispatches and memory accesses) during an interval of time
- On reaching the threshold number of operations, no additional operations are performed until the next interval
- Improperly selecting the interval length may cause power budget violations and unnecessary performance degradation
- Study of workload characteristics and system characteristics to come up with policies for power shifting
- Policies used for power shifting
    - Proportional-by-last-interval (PLI)
        - Assumption is that the activities of the components in the next interval will be the same as in the current interval
        - The estimates are used to determine the required power allocation for each component for the next interval
    - Sliding Window
        - Utilizes information from multiple consecutive intervals, thus averaging out short-term behavior
        - Instead of enforcing a budget for each interval, the sliding window policy effectively enforces an equivalent budget for each window
    - On-demand

        - This policy addresses the estimation inaccuracy that arises when the last interval is not a very good indicator of component activity for the next interval
        - If the total system power is well below its budget, the on-demand policy does not enforce any component power budgets – avoiding unnecessary dampening of the natural activity of workloads from incorrect estimation of requirements for the next interval
        - When the system power approaches its budget, the on-demand policy behaves like the PLI policy
    - Run-to-exhaustion (RTE)
        - It does not attempt to set component budgets at all– instead it allows the system to run normally while monitoring – on a cycle-by-cycle basis– the energy used within the interval
        - If any additional energy consumption would result in the average power for the interval exceeding the system power budget, the processor stops fetching for the remainder of the interval
- Results suggest that dynamic power budgeting performs significantly better than static budgeting by adapting to application-specific and phase-specific power requirements

2. Limiting the Power Consumption of Main Memory

[Published in ISCA'07, Authors from Federal University of Minas Gerais, Brazil & Rutgers University, USA]

- The paper proposes dynamic approaches for limiting the power consumption of main memories to a pre-established power budget
- Memory under consideration
    - RDRAM-based memory subsystems for the ability to control the power state of each memory chip independently
    - Although authors claim that their techniques are also applicable to DDR SDRAM technologies
- Benchmarks
    - MediaBench bench-marks, representing the workloads of hand-held devices
    - SPEC CPU2000 benchmarks, representing the workloads of desktop systems
    - A client-server benchmark, representing the workloads of server systems

- Important table

Power State/Transition	Power (mW)	Delay
Accessing	1167	–
Active	300	–
Standby	180	–
Nap	30	–
Powerdown	3	–
Active → Standby	240	1 memory cycle
Active → Nap	160	8 memory cycles
Active → Powerdown	15	8 memory cycles
Standby → Active	240	+6 ns
Nap → Active	160	+60 ns
Powerdown → Active	15	+6000 ns
Standby → Nap	160	+4 ns
Nap → Powerdown	15	+∼0 ns

Table 1: RDRAM power states, consumptions, and overheads.

- The idea behind proposed techniques is to have the memory controller adjust the power state of the memory devices so that their overall power consumption does not exceed the budget
- Assumption is that the budget has to be high enough that at least one memory device can be accessed at any time
- The proposed technique uses knapsack and greedy algorithms to decide the timings at which memory devices should be transitioned to suitable low-power modes such that the instantaneous power of memory is always within the power budget
- Four techniques have been proposed
    - Knapsack
        - Problem formulation similar as described in mail
        - Initialization of the memory controller assigns the states to each memory device randomly
        - To guarantee that the power budget is not exceeded at run time, Knapsack manages power states dynamically
            - If the memory device to be accessed (target device) is already in active state, no transitioning is done
            - When it is in a low-power state, an active device is selected to transition to the *current power state of the target device*

                - After this transition occurs, the target device can be activated and accessed
                - Maintains two queue of active device and non-active (other power states) device
    - LRU Greedy
        - This policy tries to keep as many memory devices as possible in active state
        - It involves a single data structure kept by the memory controller, the LRU queue of memory devices
        - When a device is about to be accessed, it is removed from the LRU queue
            - If the target device is active, the controller moves it to the end of the queue and proceeds with the access
            - If the target device is in a low-power state, the controller calculates whether activating it would violate the budget.
                - If not, the controller moves it to the end of the queue, activates it, and allows the access to proceed.
                - If so, LRU-Greedy starts with the LRU memory device, sending it to the shallowest power state that would satisfy the budget. If changing the state of the LRU device alone is not enough, it is left in power down state and the process is repeated for the next device on the queue, and so forth, until the budget is finally satisfied
    - LRU-Smooth
        - This policy tries to keep more devices in shallow low-power states, rather than fewer devices in deeper power states as in LRU-Greedy
        - To accomplish this, LRU-Smooth traverses the LRU queue differently than LRU-Greedy when the target device is in low-power state and activating it would violate the power budget
        - In this policy, controller goes through the LRU queue (from the LRU device to the MRU device) sending each device to the *next lower power state* (and eventually returning to the front of the queue, if necessary) until the set of devices in the queue consumes less power than the budget minus the power consumption of one active device

    - LRU-Ordered
        - This policy addresses the problems of LRU-Greedy and LRU-Smooth at the same time
        - The idea is to assign low-power states evenly (as in LRU-Smooth) but avoid sending active devices to low-power mode if possible (as in LRU-Greedy and Knapsack)
        - This uses two data-structures
            - An additional data structure is a priority queue for the memory devices that are in low-power states
            - The queue is ordered by how shallow the power mode is and devices in shallower states are selected to go to deeper states first
            - The LRU queue is then reserved for active devices only
- Results
    - For all applications, Knapsack and LRU-Ordered degrade performance only slightly
        - less than 3% in all cases and less than 1% in all but one case (bzip2)
        - Knapsack optimizes performance within the available power budget
        - LRU-Greedy and LRU-Smooth degrade performance more substantially; they can exhibit degradations as high as 76% and 67% respectively

3. Optimizing Energy Efficiency of 3-D Multicore Systems with Stacked DRAM under Power and Thermal Constraints

[Published in DAC 2012, Authors from Boston University]

- 3D multicore systems with stacked DRAM provide performance benefits but may cause 3D systems to exceed the power budget or create thermal hot spots
- This paper proposes a runtime optimization policy to maximize performance while maintaining power and thermal constraints
- The proposed policy dynamically monitors workload behavior and selects among low-power and turbo operating modes accordingly
- This paper tries to jointly evaluate and optimize performance, power, and temperature profiles at run-time for logic and DRAM layers in 3D systems simultaneously

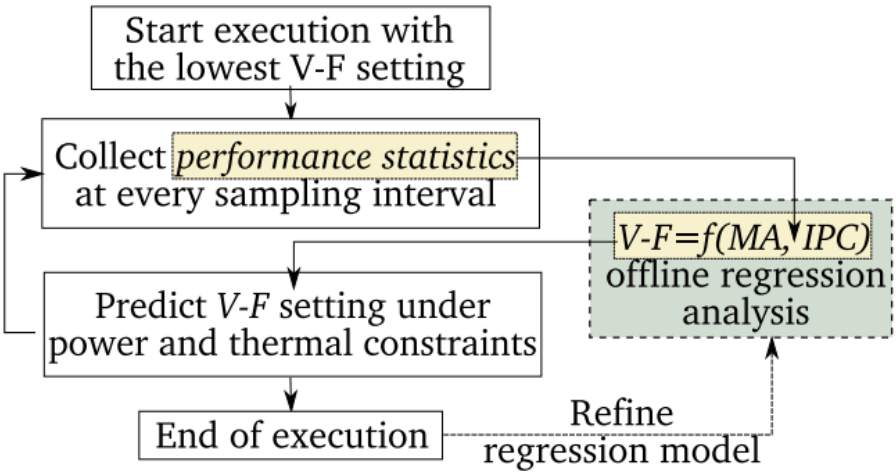
- Authors propose a novel runtime optimization policy for selecting V-F settings to maximize system performance subject to power and thermal constraints.
- For memory-bound benchmarks
    - 3D stacked memories have significant performance improvements when running on 3D systems with on-chip DRAM compared to the 2D baseline with off-chip memory
    - However, power and temperatures on both logic and DRAM layers rise significantly
    - The paper proposes a policy to select a low-power V-F setting to maximize throughput under power and thermal constraints
- For CPU- bound benchmarks
    - 3D stacked memories provide limited performance improvement compared to the 2D baseline
    - However, stacking the DRAM layer with the logic layer provides a temperature slack as the DRAM layer is much cooler than the logic layer and helps maintain low temperature
    - In this case, authors boost system performance using high-frequency turbo modes without creating thermal problems
- The goal is to maximize throughput (instructions per second, IPS) under power and thermal constraints

$$\begin{aligned} \underset{(f,v)\in(F,V)}{\text{maximize}} \quad & IPS(f,v) \\ \text{subject to} \quad & power(f,v) \leq P_{cap},\ temperature(f,v) \leq T_{thld}. \end{aligned}$$

    - Pcap is the power budget of the target system
    - Tthld is the peak temperature threshold to ensure reliable operation
    - Pcap of 200W, Tthld at 85 degree C
- Construct a regression-based model for selecting the V-F settings
    - Choose instructions per cycle (IPC) and memory access per instruction (MA)
    - IPC is a good indicator of the power of the logic layer and MA is a good indicator of the power of the DRAM layer
    - Power densities on both layers affect chip peak temperature on the 3D system
    - V-F prediction model is in the form of

$$VF = c0 + c1 \cdot MA + c2 \cdot IPC + c3 \cdot MA * IPC$$

- Different coefficients used in the model depending on the current V-F setting, as MA and IPC vary with the V-F setting



4. RAPL: Memory Power Estimation and Capping

[Published in ISLPED'10, Authors from Intel Corporation]

- This paper describe a new approach for measuring memory power and demonstrate its applicability to a novel power limiting algorithm
- Authors evaluate their approach in the modern servers and show that they achieve up to 40% lower performance impact when compared to the state-of-art baseline across the power limiting range
- The reference power measurement used in this paper employed a memory power riser to directly measure the DIMM power with an Agilent 3470 data acquisition system at a 10ms cadence
- Activity based power model
    - Measuring memory power using a power model based on activity counters and predefined weights
    - Pre-compute the weights offline for each activity based on the information provided by different DRAM vendors

ACTIVITY	WEIGHTS	Units
Activate (A)	36.3	nj/Activate
Read (R)	22.8	nj/Read
Write (W)	21.7	nj/Write
CKE=HIGH adder	2112	mW
CKE=LOW baseline	1663	mW

Table 2: Memory Power Model Weights

- These calibrated weights and activity counters are fed into a power model to estimate memory power (along with baseline power K) during system operation:

$$\begin{aligned} Power &= W_{act} \cdot A + W_{read} \cdot R + W_{write} \cdot W \\ &+ W_{CKE\_H} \cdot CKE\_H_{adder} + K \end{aligned}$$

- Running Average Power Limit Algorithm
  - RAPL maintains an average power limit over a sliding time window
  - RAPL algorithm determines the power budget based on memory bandwidth, specified power limit and time window

$$PwrBudget_{hard} = N \cdot PwrLimit - \sum_{i=1}^{N-1} MemoryPwr_i$$

- It maintains an energy budget and a history of power
- consumption over the time window
- Starting at each interval, it computes the budget by subtracting the power consumed over the last N-1 intervals from the available power budget
- Upon computing power budget, RAPL searches M-state table to determine the state that will not exceed available power budget
- Algorithm maintains two types of limits
  - Soft and hard limits

    - Hard limits use the entire window N of past power consumption history and allocate all available budget to the next time interval
    - Soft limits shift the time window by M intervals to capture the most recent workload behavior and smooth the effects of power limiting by predicting average bandwidth demand over the next M time intervals

$$PwrBudget_{soft} = \frac{N \cdot PwrLimit - \sum_{i=M}^{N-1} MemoryPwr_i}{M}$$

- The fundamental difference between the RAPL and baseline algorithm is dynamic selection of the MPL state in the RAPL algorithm vs. a static MPL state for the baseline algorithm

5. An Analysis of Efficient Multi-Core Global Power Management Policies:Maximizing Performance for a Given Power Budget

[Published in Micro 2006, Authors from IBM Research and Princeton University]

- This paper proposes a global power manager
    - Provides a hierarchical feedback-control based mechanism to sense the per-core power and performance "state" of the chip at periodic intervals
    - Sets the operating power level or mode of each core to enforce adherence to known chip-level power budgets
- The main motivation in architecting a global power manager is to exploit the known variability in demand and characteristics of the input workloads
    - The different regions of execution are commonly referred to as phases of application behavior
- In a multi-core setting
    - Global power manager must observe phase states and transitions across all the cores
    - Next it takes appropriate mode-setting actions with the objective of enforcing a chip-level power budget
- Each core has multiple operating modes called power modes

    - Modes can be set independently by the hierarchical controller, depending on workload behavior, overall budget constraints and mode-shift constraints
- Two types of controllers are being used
    - A local controller within each core without interference from the global controller
        - Baseline dynamic power management techniques like clock-gating, fetch-gating, etc. may be used to provide a baseline power-efficient core functionality
        - Authors assume the presence of on-core current sensors
        - On-core performance monitoring counter hardware used for performance-related information
        - Local current sensors provide the core power consumption information, and the local performance counters provide the core performance information (i.e. retired instructions per sampling period) to the global controller
    - A global controller for all cores
        - The local controllers provide periodic per-core power-performance data to the global manager
        - The global power manager periodically monitors the power and IPC of each core and sets the operating modes of the cores for the next monitored interval
        - The global management layer provides per-core mode-setting directives, in coordination with higher-level scheduling and load-balancing directives provided by system software

Mode	Power Savings	Performance Degradation
Turbo	None	None
Eff1	15%	5%
Eff2	45%	15%
General Target	**3X**	**1X**

Table 3. Target $\Delta Power : \Delta Performance$ ratios for different power modes.

- Policy proposed for global management
    - Priority
        - Assigns different priorities to different tasks/cores
        - For a four-core CMP, core4 has the highest priority and core1 has the lowest priority
        - Tries to run the fourth core as fast as possible, while preferring to slow down the first core first in case of a budget overshoot
    - PullhiPushLo
        - Balances the power consumption of each core, by slowing down the core that has the highest power in case of a budget overshoot, and by speeding up the lowest power core when there is available power slack
    - MaxBIPS
        - Targets at optimizing the system throughput, by predicting and choosing the power mode combination that maximizes the throughput at each explore time
        - This policy predicts the corresponding power and BIPS values for each possible mode combination. Afterwards, it chooses the combination with the highest throughput that satisfies the current power budget
    - Chip-Wide DVFS
        - All cores transition together into Turbo, Eff1 or Eff2 modes at each explore time based on budget constraints
- Best policy
    - For all the given budgets, MaxBIPS performs significantly superior for performance as expected.
    - For each power budget, MaxBIPS achieves the least over all performance degradation in comparison to other policies

6. PCP: A Generalized Approach to Optimizing Performance Under Power Constraints through Resource Management

[Published in ICAC 2014, Authors from University of Chicago]

- Developing systems that operate within power budgets is a constrained op-timization problem
    - Configuring the components within the system to maximize performance while maintaining sustainable power consumption

- Prior approaches address these challenges by fixing a set of components and designing a power budgeting framework that manages only that one set of components
- This paper presents PCP, a general solution to the power budgeting problem that works with arbitrary sets of components, even if they are not known at design time or change during runtime
    - To demonstrate PCP, authors implement it in software and deploy it on a Linux/x86 platform
- PCP is Runtime Control Framework with three main components
    - System
        - Describes system under control, including the available components, or Knobs that affect power and performance
    - Controller
        - Measures power consumption and produces a generic control signal indicating available overhead in the power budget
    - Estimator
        - Dynamically tunes the controller to the specific application and system under control and accounts for approximations in the initial model
    - Translator
        - Turns the generic control signal into a performance-optimal component configuration that achieves the power budget