An Integrated DVFS Policy Approach for CPU & Memory

Furkan Ercan[1], Neven Abou Gazala[2], Howard David[2]

1. METU NCC, Sustainable Environment and Energy Systems
2. Intel Labs, Intel Corporation

ICEAC 2012

December 4, 2012

Overview

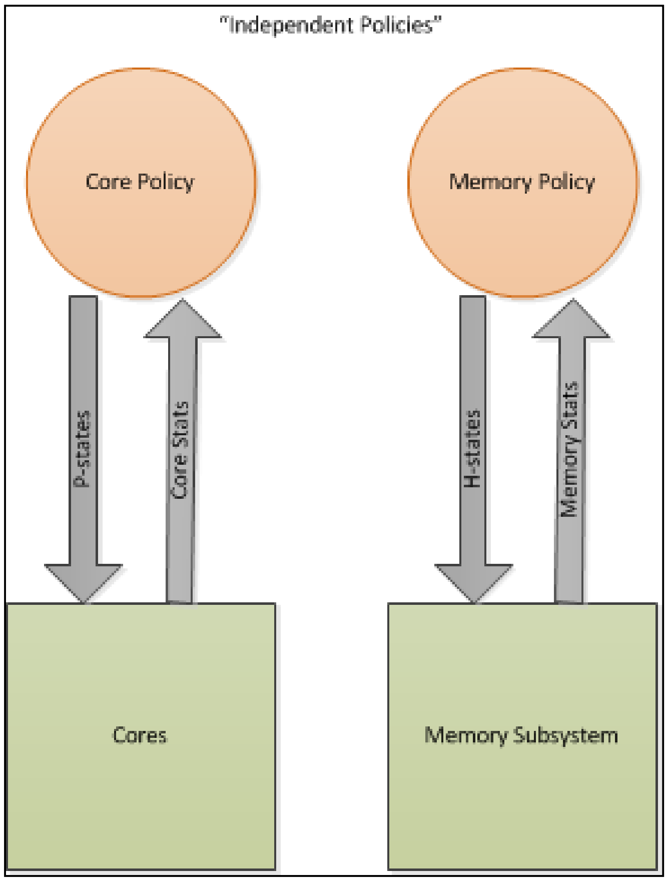
- Introduction
- Core & Memory Power Model
- Integrated Policy
- Experimental Setup
- Results
- Summary

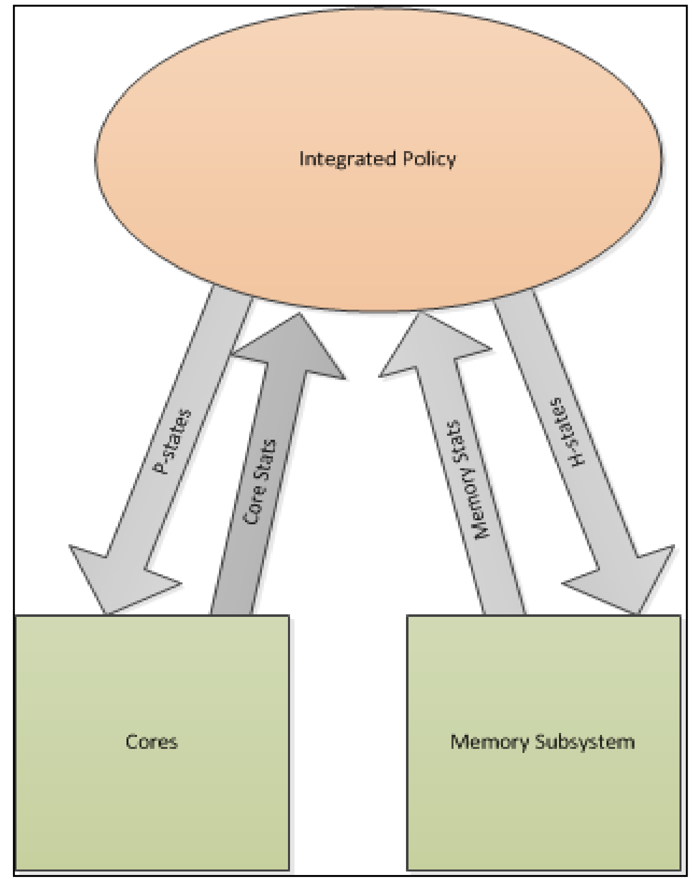
Introduction

- CPU and memory currently have separate policies to control their DVFS
- Existing policies are based on local information and control; what is missing is the interactions between resources
- Our aim is to find an effective way to manage DVFS for both given a *performance loss tolerance*
- The contribution of this work is to manage CPU and memory more efficiently through an integrated policy

Integrated vs. Independent Policy

- **Independent Policies (State-of-the-art):** Considers their respective resources only (CPU or memory), one policy per resource.
- **Integrated Policy (proposal):** Considers both resources at the same time, a better perspective of workload's resource allocation.





Background: Current DVFS Policies

Core DVFS (Linux)

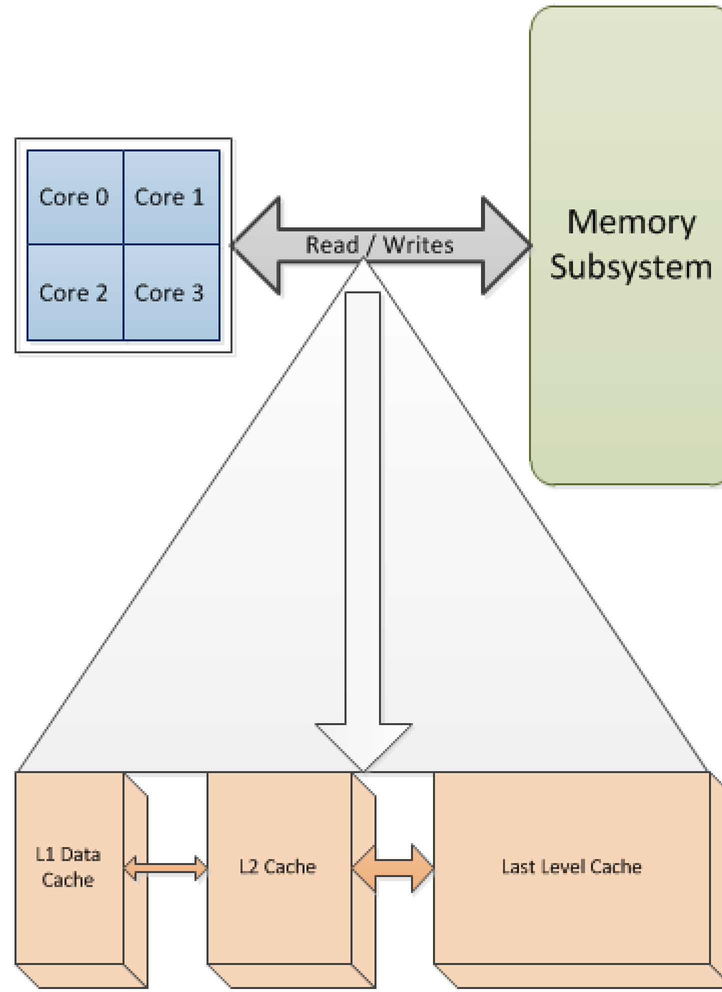
- ondemand: based on system load (mostly used)
  - Does not have a tolerance, will run at max frequency for the workloads as spec web.
- conservative: based on system load with gradually switching among P-states.

Memory DVFS

- H-state Policy : Memory utilization
- Smooth transition between H-states

Integrated Policy

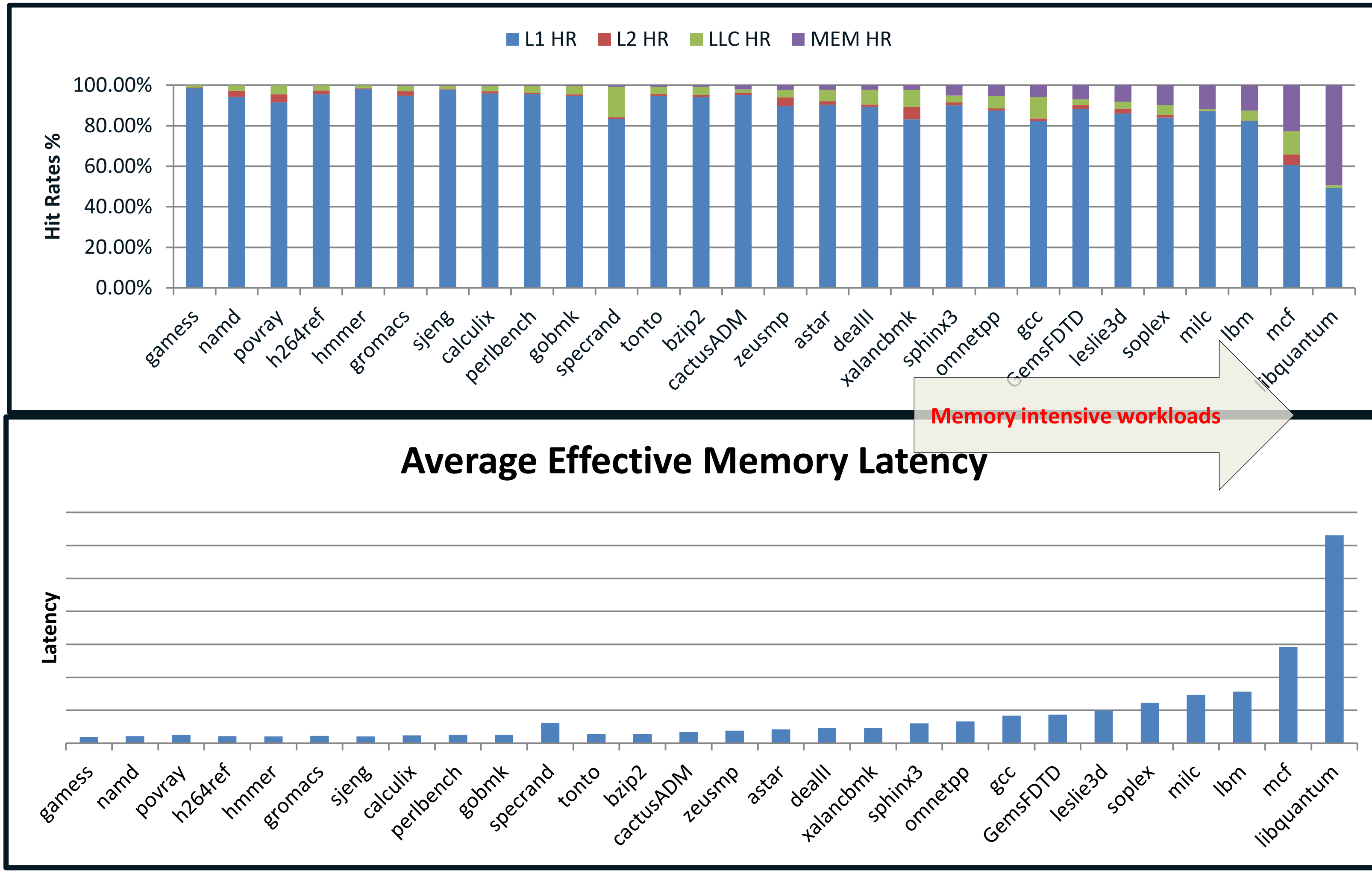
- Given the tolerance that an application can tolerate
- The policy monitors and control CPU, uncore and memory frequency at the same time.
- Dynamically allocate slack based on application's usage of each resource
  - i.e. cache hierarchy hit rates & memory BW
- Tolerance is controlled by adjusting latency of accessing to the memory hierarchy (L1, L2, LLC & memory) using 'Effective Memory Latency' equation



Effective Memory Latency

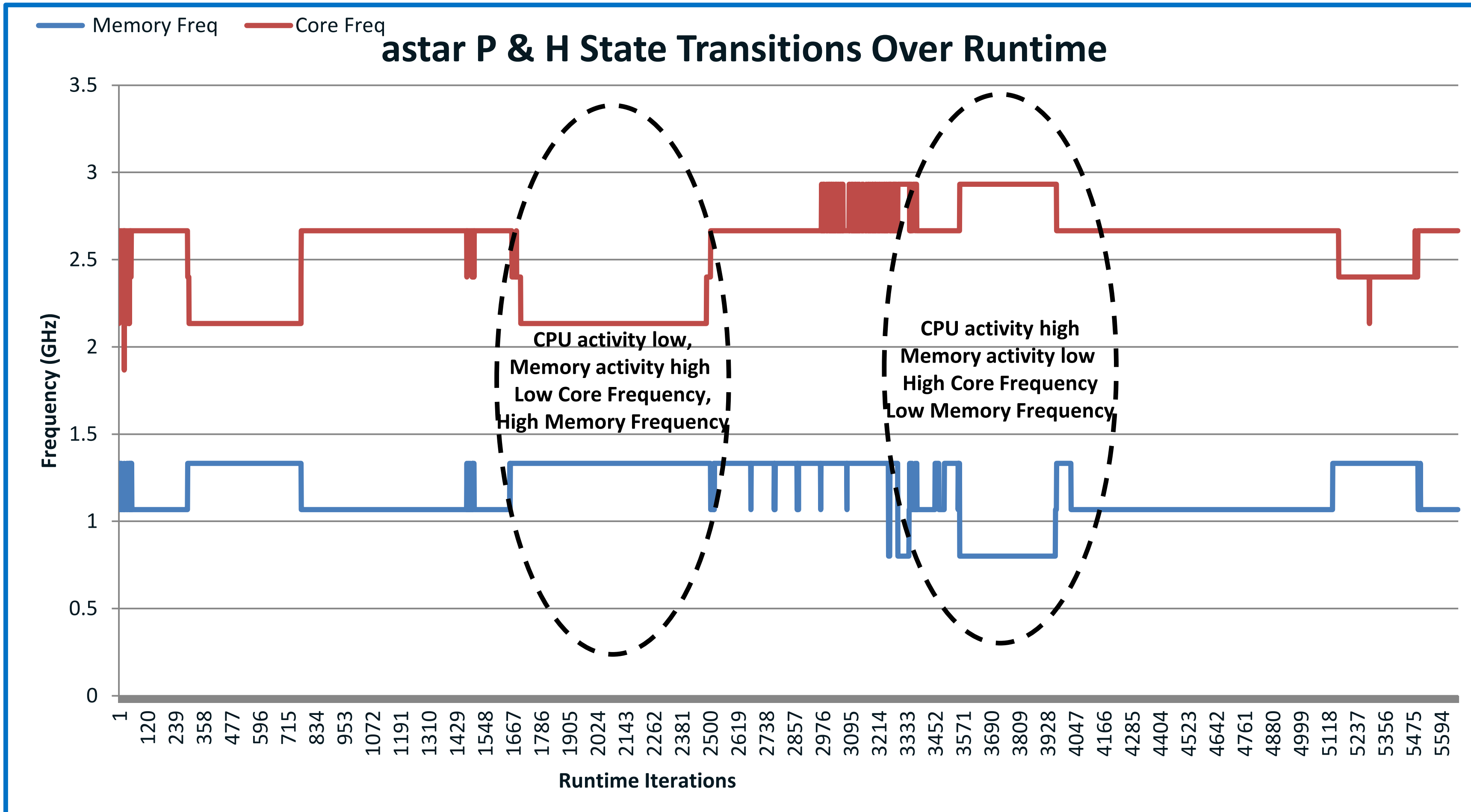
- Workload characteristics are basically considered as CPU intensive and memory intensive (or both) depending on how utilized they use the CPU and memory.
- We define Effective Memory Latency (EML) as: (unit in ns):
- *EML = L1D Read Hit Rate x L1D Access Time +*
- *L2 Read Hit Rate x L2 Access Time +* → Depends on core freq
- *LLC Read Hit Rate x LLC Access Time +* → Depends on uncore freq*
- *Memory Read Hit Rate x Memory Access Time* → Depends on mem freq**
- Memory access time depends on the memory traffic (BW) and the current H-state.
- We rely on ‘read’ hit rates since writes have a separate buffer that does not cause as much stall as reads.

Average Hit Rates of spec CPU 2006 workloads

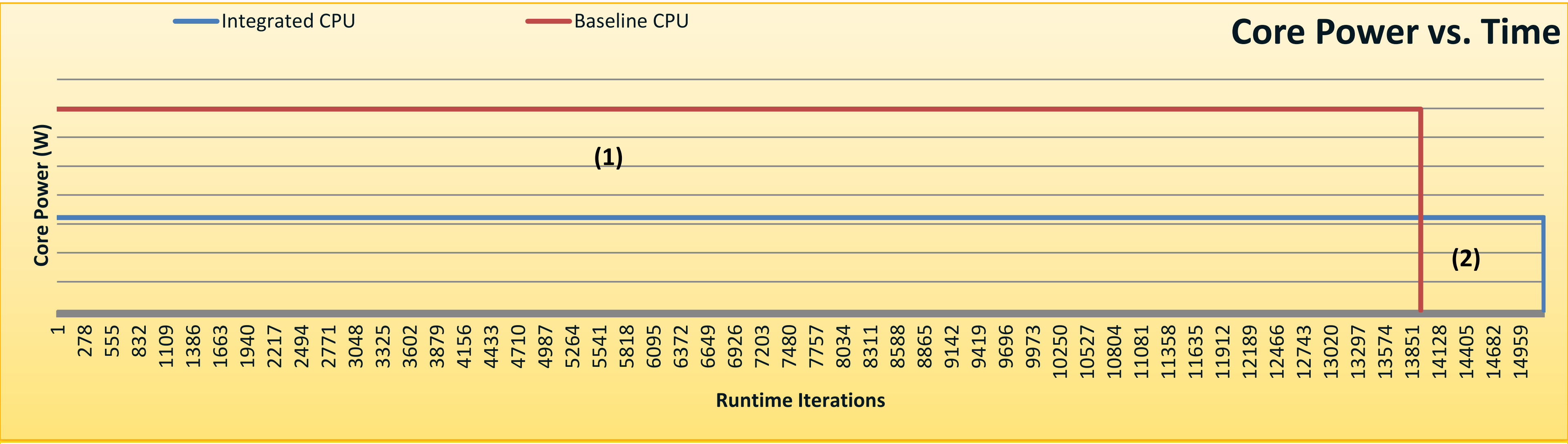


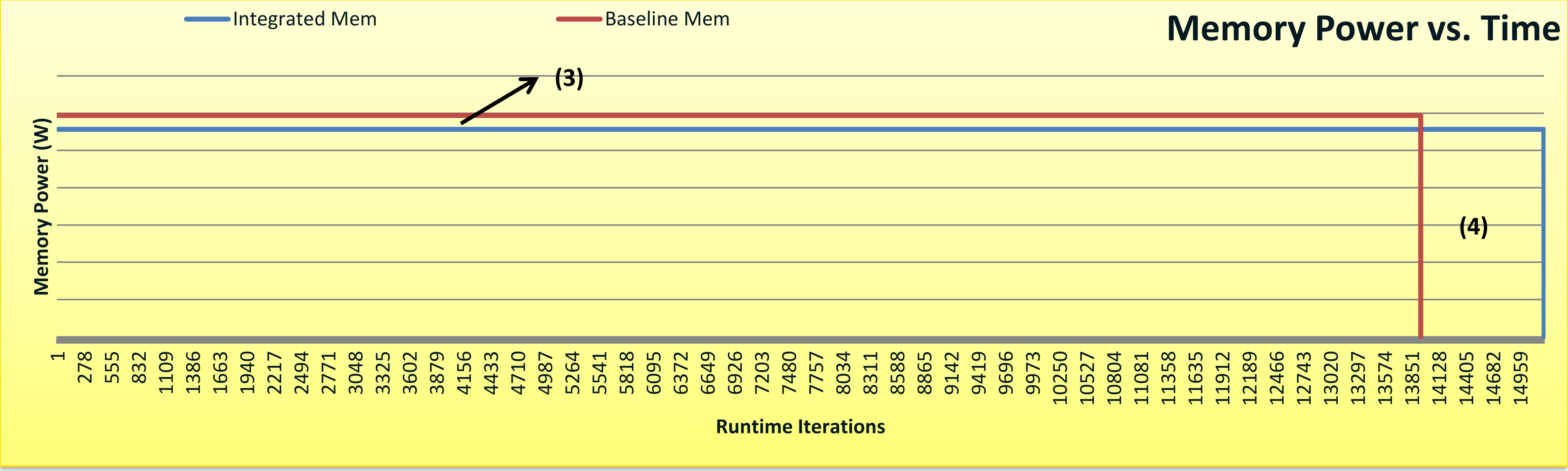
December 4, 2012

Integrated Policy: Dynamic



CPU versus Memory Efficiency: libquantum





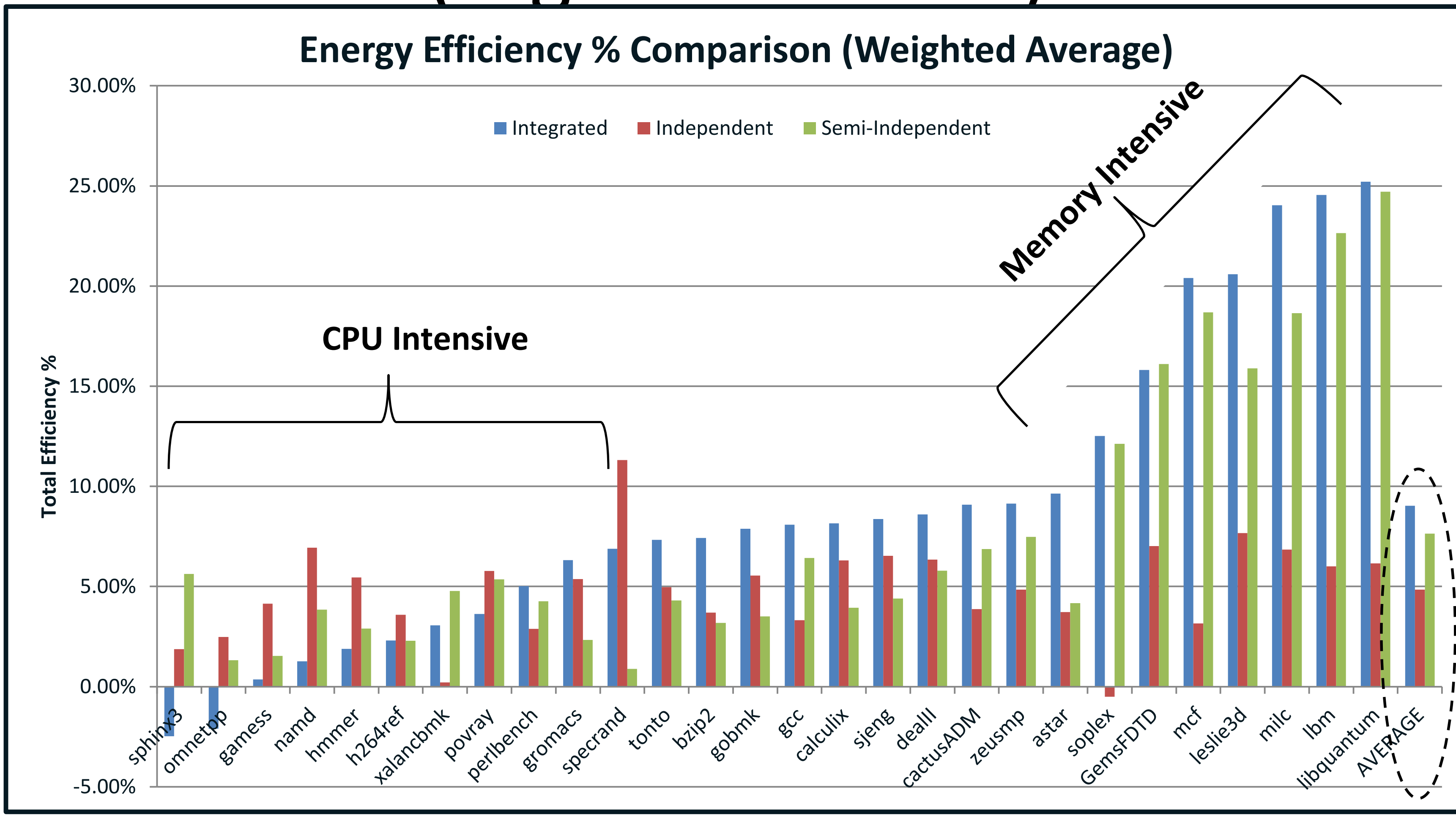
Experimental Setup

- NHM Greencity Server System
- Two C0 stepping NHM packages
- 2 DIMMs per channel, 48 GB (DDR3-1333 dual rank by 8)
- Red Hat ™ Enterprise Linux OS 6.0
- Extended H-state Tool
  - H-state Tool developed by Intel Labs controls H-states w.r.t BW
  - Developed over original H-state Tool to monitor and control CPU and memory with an acceptable performance loss.
- SPEC CPU 2006 workloads
- P-state range : 1.6 GHz to 2.93 GHz with 0.267 GHz stepping
- H-state range: 0.8 GHz to 1.33 GHz with 0.267 GHz stepping

Policies to be Compared & Latency Constraint Allocation

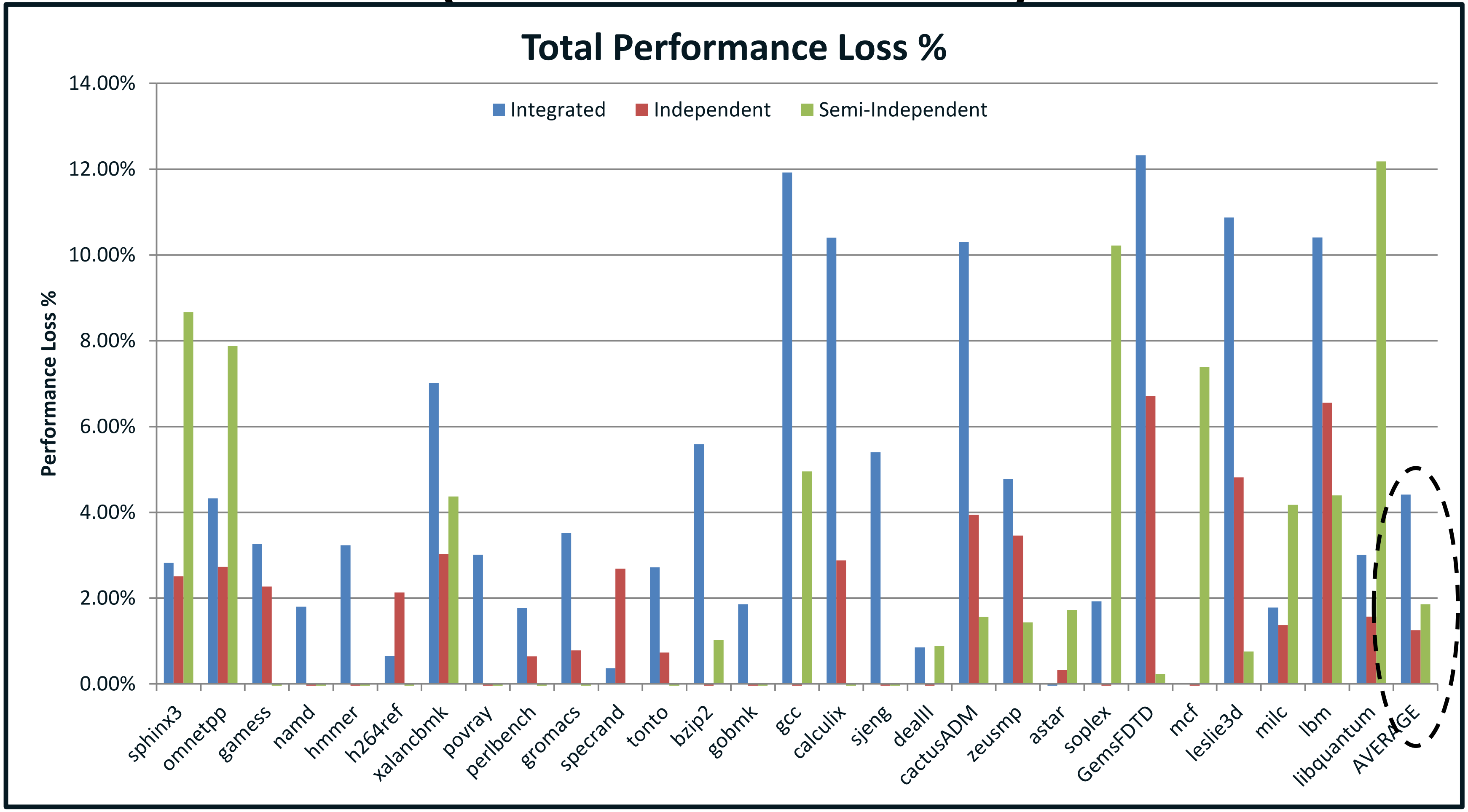
	Explanation	**Latency Constraint (S)**
Integrated Policy	Single mechanism that controls both P & H states subject to a performance loss tolerance *S.*	Dynamically allocated to core and memory
Independent Policy	Two separate policies for P & H states subject to tolerance *S1 and S2.*	S1=2/3 S to Core S2=1/3 S to Memory
Semi-Independent Policy	Similar to Integrated without accounting for interactions. Assuming other subsystem running at max freq. Subject to *S1 and S2.*	S1=2/3 S to Core S2=1/3 S to Memory

Total Energy Efficiency % Comparison (Higher is Better)



December 4, 2012

Total Performance Loss % Comparison (Lower is Better)



December 4, 2012

Summary

– An Integrated Policy with a defined performance loss tolerance has a better profile in terms of energy efficiency over an Independent Policy.
– Semi-Independent Policy profile is closer to Integrated Policy, yet it is limited with pre-distribution of the tolerance to the resources, thus achieves lower energy efficiency.
– Integrated Policy saves energy mostly from memory-intensive workloads which does not utilize the CPU as much as a regular (or CPU intensive) workload.

Thank You