Exploring Cross-layer Power Management for PGAS Applications on the SCC Platform

Author: Marc Gamell

marcgamell@uoc.edu

Advisor: Ivan Rodero

Open University of Catalonia

January 23, 2012

Introduction

Motivation

- Projects like Exascale shows that future HPC systems exposes new challenges:
  - Programming complexity or
  - Extreme energy efficiency
- We try to approach this problems taking in mind that:
  - PGAS programming model offers a simple but powerful parallel programming approach
  - HPC systems will feature large numbers of high-core-count processors (many-cores)

Introduction

Our work

- Profile UPC (PGAS) application kernels on Intel's SCC (many-core)
- Propose a power management middleware that allows:
  - Cross-layer power management via PGAS language extensions
  - Tunning power/performance tradeoffs
- Provide recommendations to support PGAS power management on many-core

Background

Background

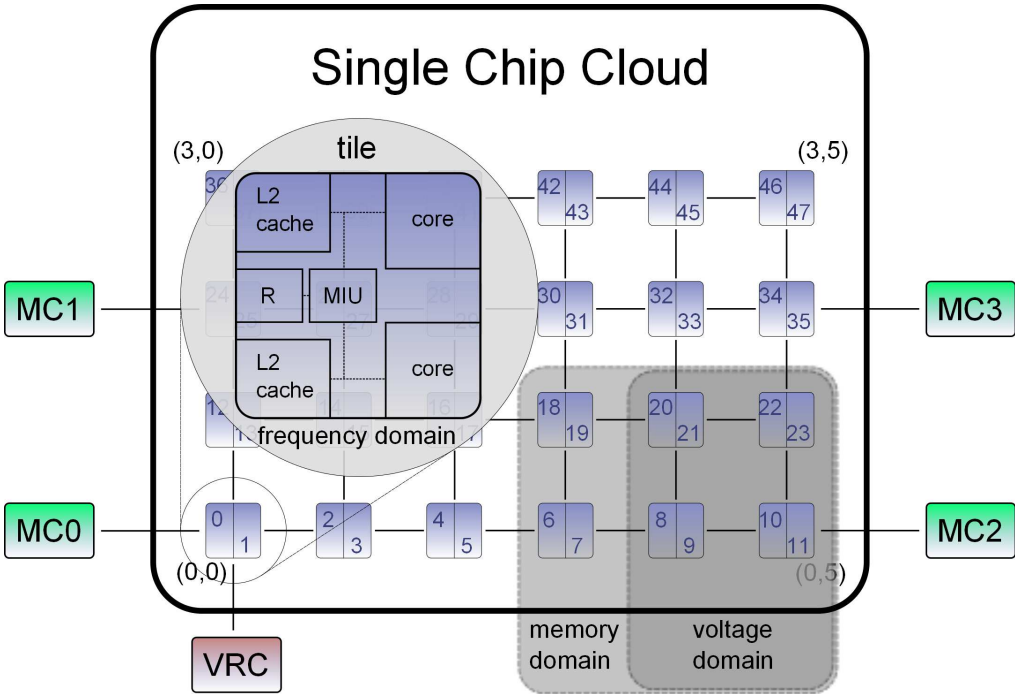
Many-core systems

- Multi-core-like processors with large numbers of simpler cores
- Smaller cores in terms of their die-area
- Have more attractive power/performance ratios
- The throughput of the system increases linearly with the larger number of small cores
- Future HPC processors?

Background

SCC

- Many-core prototype by Intel Labs Tera-scale Program
- 48 x86 P54C Pentium cores
- 24-router on-die mesh network, hardware message-passing
- Per-core 16 KB L1 cache, 256 KB L2 cache



Background

SCC power management tools

- SCC offers fine-grained power management (frequency or voltage scaling):
- Frequency:
  - per-tile (2 core) management
  - write a value in a tile-mapped register
  - 100 to 800 MHz adjusting in 15 steps
  - adjustment takes 20 clock cycles (very fast)
- Voltage:
  - per-voltageDomain (8 cores) management
  - send a command to the system-wide VRC
  - 0.7 to 1.1 V adjusting
  - adjustment takes 40.2 ms, in average (slow)
- Power ranges from 25 W to 125 W

Background

PGAS

- PGAS paradigm:
  - An emerging parallel programming language model
  - Oriented for large-scale systems
  - Offers a shared memory space partitioned among all threads
  - Each portion of the memory is local to one of the processors
- Goals:
  - Improve application performance (thanks to data locality)
  - Ease the parallelization and enhance user productivity:
    - abstract thread synchronization
    - implicit message passing
- Incarnations:
  - UPC, Co-array Fortran, Titanium, Chapel, X10

Background

UPC: Unified Parallel C

- What is UPC?
  - An ISO C 99 extension
  - Supports explicit parallelization and the PGAS model
  - A Single Program Multiple Data (SPMD) computation model
- Goals:
  - Add only simple extensions to C (easy learning curve)
  - Easy-of-use of PGAS paradigm
  - High level control over distributed data
  - Allow static/dynamic shared memory allocations
  - Incremental performance improvements

Related Work

Related work

Layered energy efficiency of CPU subsystem

- OS-level (cpu-freq, ACPI)
- Workload-level (most successful):
  - Overlap computation and communication (MPI)
  - Heuristics data
  - Exploit low power modes when is not in the critical path
  - Reduce power when a task is in a slack
- Application-level (Eon)
- Compiler-level
- Any of these in cross-layer approach!

Related work

Existing PGAS research

- Improvement of UPC collective operations
- Hybrid models to improve performance limitations
- X10 implementation for the SCC
- UPC implementation for Tilera's Tile64 many core

Existing many-core research

- Energy optimization through voltage-island formation
- SCC performance tuning: message passing, application level automatic performance, MPI over RCCE...

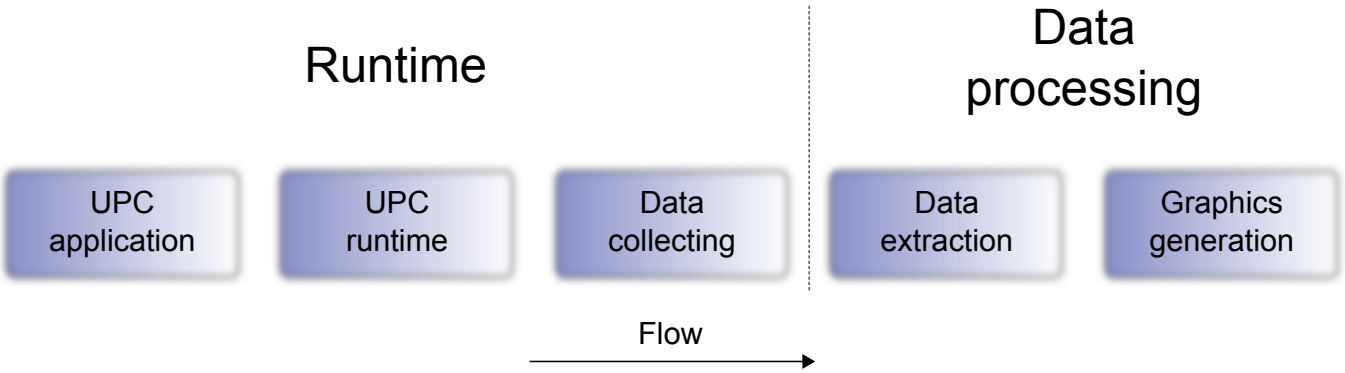
No cross-layer power management
No power management in the PGAS framework
Few power management conclusions for many-cores

Profiling

Profiling

PMI system

- Existing tools would interfere with profiling due the overhead
- Implement a new lightweight instrumentation system: `PMI`



Benchmarks

- NAS UPC (FT class C, MG class C, EP class D)
- Sobel
- Matmul (customizable synthetic application)

FT

- Only memget and wait are good candidates for exploring power management
- Both operations have lots of very-short calls and several med-long operations

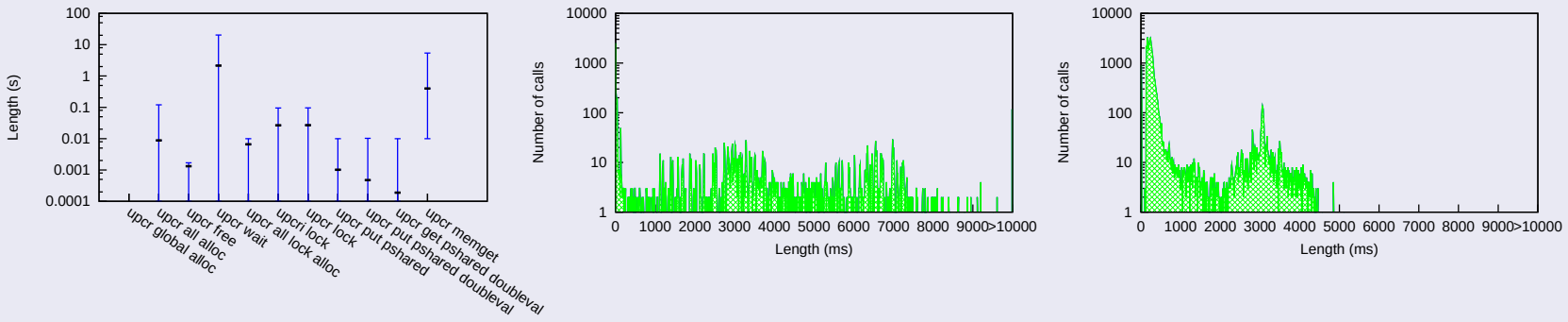


Figure: UPC used operations, wait and memget call length histogram

Profiling

MG and EP – not good candidates

- Long wait's only in the initialization phase
- The rest of execution is **well balanced**
- Both are **cpu-intensive**

Sobel

- Large initialization phase, performed only by core 0
- Very interesting for it's big imbalance (**long wait periods**)
  - In the initialization phase
  - Between every Sobel iteration (sync.)
- We can try to save energy during initialization phase and sync. periods

Synthetic Matmul

Matmul is useful to study the potential regarding **imbalance**

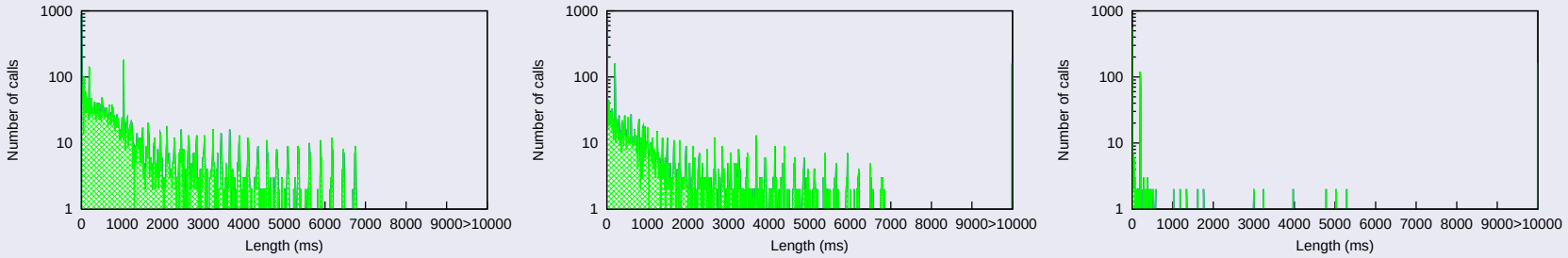


Figure: Matmul's wait length histogram (3, 51, 97 % of imbalance).
Note that, the more imbalance the application, the longer wait calls are.

Middleware

Middleware

Goals

- Reduce energy footprint by exploiting application's slack periods
- Support all PGAS implementations (UPC, CAF...)
- Provide user-level interface to easy-tune the runtime power management ("hints")

Middleware

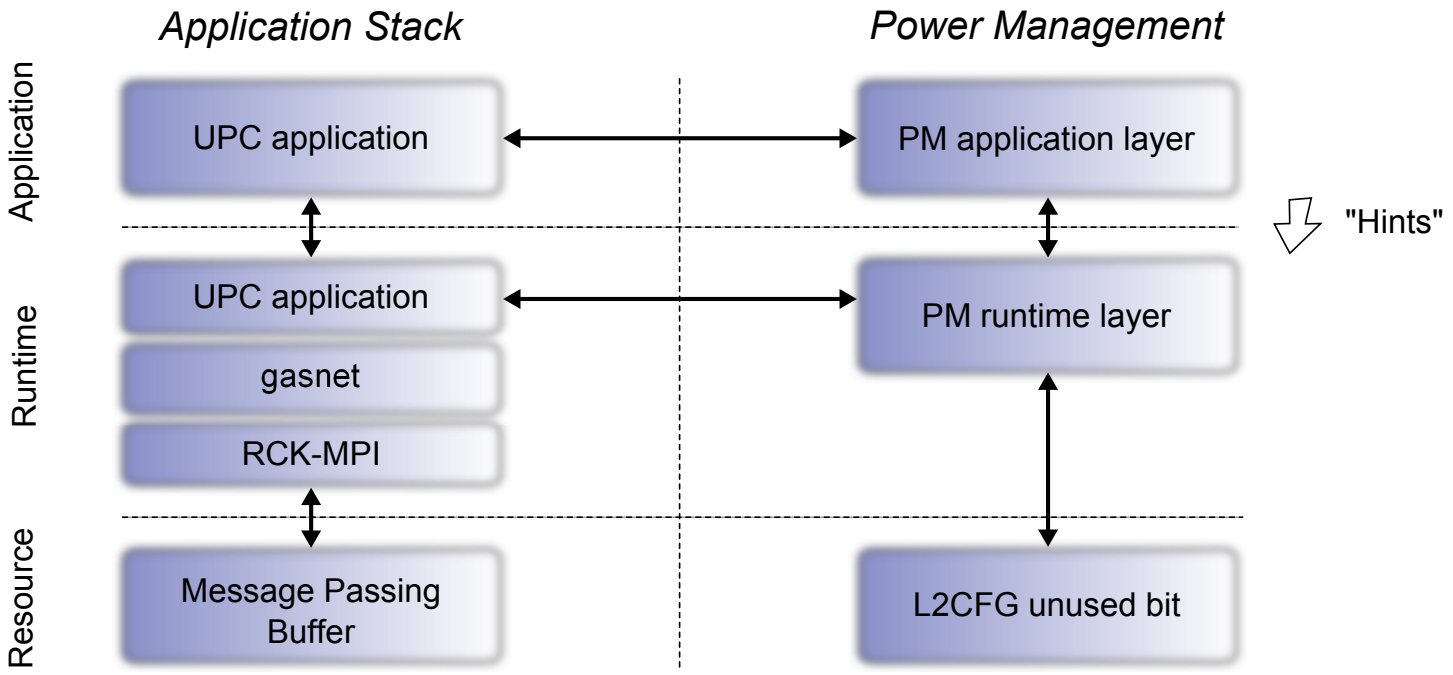


Figure: Cross-layer architecture

Middleware

Architecture

- Modular architecture
- External requests via unix sockets

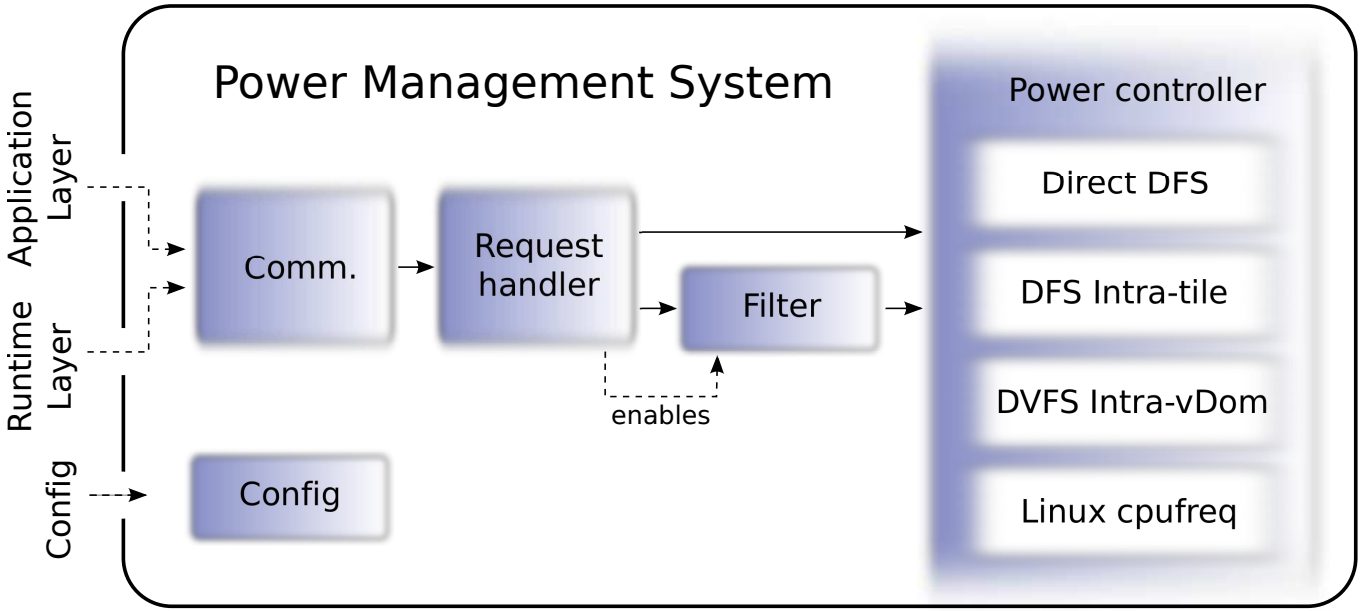


Figure: Power management system architecture

Middleware

User hints

- PM_PERFORMANCE, maximum performance
  full power
- PM_CONSERVATIVE, balance power/performance
  low power modes during slack
- PM_SAVE_ENERGY, minimum power, limited delay penalty
  low voltage level, regardless application's slack periods
- PM_AGGRESSIVE_SAVE_ENERGY, maximum energy save
  lowest voltage and frequency levels

Middleware

Filtering

Aim:

- Eliminate shortest calls (i.e. the zero-length cluster)

Policies:

- **thresholds**:
  waits an amount of time before accepting a request
- **moving average**:
  recalculates the threshold based upon the historical call length
- **mixed**:
  limits the value of moving average with specified max and min

Middleware

Power adjuster

- Standard CPU-freq interface not available on the SCC
- Can't use RCCE for power management:
  - RCCE uses MPB (standard SCC tile-to-tile communication)
  - MPB is being used by RCK-MPI!
- Therefore, need to implement tools that access the hardware:
  - **Direct DFS**:
    change tile frequency
  - **DFS Intertile synchronization**:
    frequency adjustments if whole tile (2 cores) agree
  - **DVFS**:
    frequency+voltage adjustments if whole vDom (8 cores) agree

Experimental Results

Experimental results

Environment

- SCC prototype given by Intel Labs
- Berkeley UPC runtime
- RCKMPI
- Per-core Linux

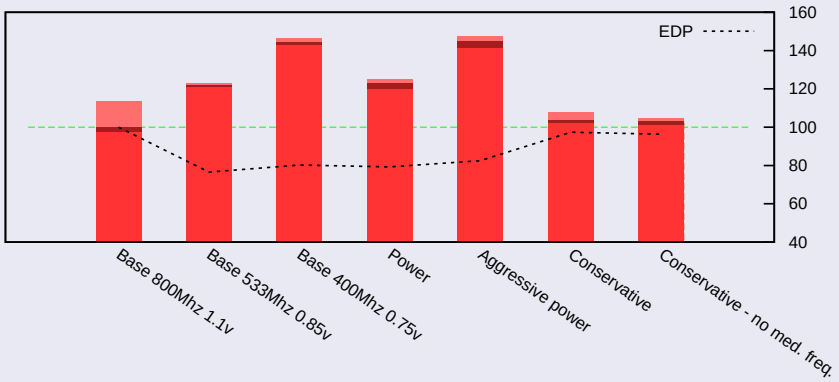
Test suite

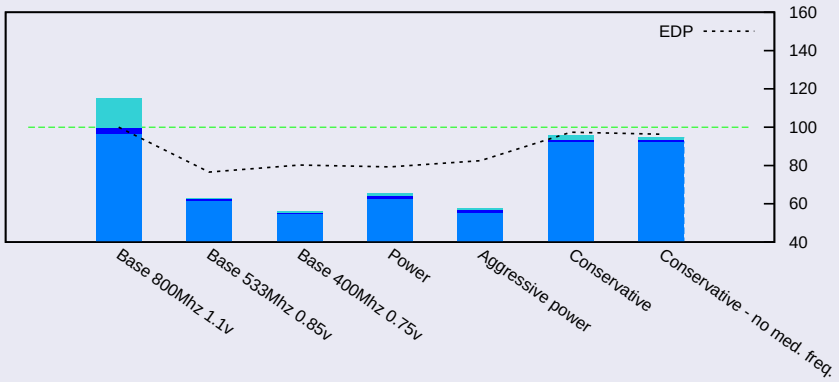
- NAS Parallel Benchmarks: FT class C, EP class D and MG class C
- Sobel edge detector kernel
- Customizable synthetic matmul application

Experimental results

FT

- Base tests power modes:
  - High (800MHz - 1.1V) ← **main base test**
  - Intermediate (533MHz - 0.85V)
  - Low (400MHz - 0.75V)
- Power managed tests:
  - Wait and memget operations
  - Using different application-level policies (whole execution)
  - Using DVFS or DFS with three power modes (low, med, high)
  - Filtering calls with threshold or moving average policies





Experimental results

FT (cont.)

- Significant results:
  - PM_CONSERVATIVE lowest time penalty (0.4%): DVFS, wait+memget, threshold 20-20, 1000-2000
  - PM_CONSERVATIVE highest energy savings (7%): DVFS, wait+memget, threshold 20-20, 300-1000
  - Other *hints* can obtain higher energy savings (45%) at higher time penalty cost
- Other conclusions:
  - No memget: lower time penalties, but lower energy savings
  - Thresholds: energy reduction, little time penalty
  - Moving average: larger energy savings $\rightarrow$ higher time penalties
  - 'Hints' allow definition of energy/performance tradeoff

Experimental results

Sobel

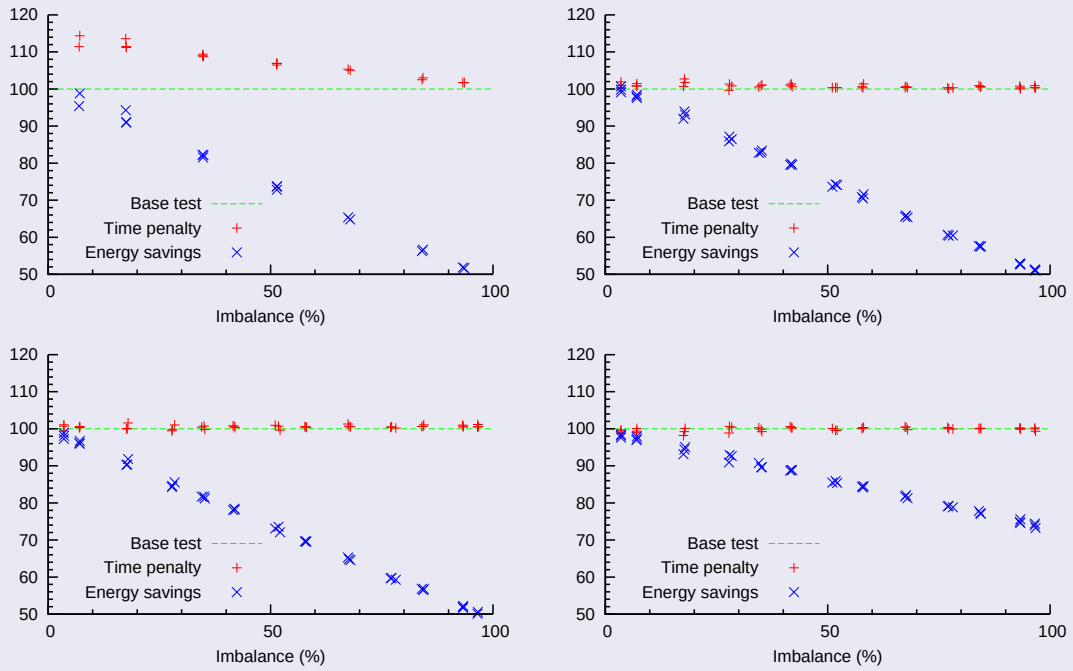
- Tests (useful for study application-level policies –hints):
  - Using the same policy during the whole application execution
  - Policy-driven *intelligent* extensions:
    - Added per-thread hints during the initialization phase
    - Common policy during the iterative phase
- Results and conclusions:
  - Default config. → 24% energy savings, 18% time penalty
  - Policy-driven → 26% energy savings, 1.5% time penalty
  - Hints avoid time delay and mantains energy savings

Test	Default			Policy-driven		
description	**Delay %**	**Energy %**	**EDP %**	**Delay %**	**Energy %**	**EDP %**
Base 800Mhz - 1.1v	100.0	100.0	100.0	100.0	100.0	100.0
Performance	100.7	100.0	100.8	N/A	N/A	N/A
Save energy	**147.6**	75.0	110.7	**101.8**	79.4	80.8
Aggressive save energy	**193.8**	72.8	141.0	**101.9**	73.7	75.2
Conservative	**118.0**	75.7	89.4	**101.5**	73.7	74.9

Experimental results

Matmul

- Tests (useful for study impact of load imbalance):
  - Different power management strategies
  - Different application-level policies (hint)
  - Imbalance ranging from 3% to 97%



Runtime PM and DVFS (no filtering and filtering), hints (DVFS and DFS), respectively.

Experimental results

Matmul (cont.)

- Results and conclusions:
  - Maximum imbalance $\rightarrow$ 50% energy savings, no time penalty
  - Energy savings are proportional to the load imbalance
  - Filtering module helps reducing time penalty
  - Energy savings with DVFS are higher than with DFS
  - Results using runtime PM and application PM are similar $\rightarrow$ runtime PM works efficiently with a properly tuned filter

Conclusion

Conclusion

Work summary

- Explore application-aware cross-layer PM for PGAS on SCC
- Design, implement and evaluate a runtime PM middleware

Main conclusions

- Certain PGAS operations (wait and memget) can provide:
  - Large energy savings, if large
  - Both energy and delay penalties, if short
- Therefore, need to distinguish short and long calls:
  - If they cluster by length $\rightarrow$ direct identification of short calls
  - If they don't $\rightarrow$ intermediate power mode helps energy savings
- Imbalanced applications allows large energy savings
- Significant energy savings can be obtained during memory access. This is surprising, as memory is shared!

Conclusion

Main conclusions (cont.)

- Cross-layer power management allows:
  - Wide range of energy and performance behaviors
  - Selection of the appropriate energy/performance tradeoff
- Hardware power management limitations:
  - SCC per-tile frequency scaling is fast, but small energy savings
  - SCC per-vDom voltage scaling is slow, but large energy savings
  - Ideal power management: per-core DVFS
    - This require a large amount of the die
    - This increases the per-core power requirements

Future work

- Explore other PGAS models
- Implement cross-layer optimizations in compiler level
- Use per-core performance counters to profile in runtime
- Extend PGAS runtime libraries to access RAM directly

The end