Moving towards exascale via optimization for application- and hardware-aware execution

Laura Carrington

University of California, San Diego

San Diego Supercomputer Center

Performance Modeling and Characterization Lab (PMaC)

Workshop on Extreme-scale Performance Tools

Nov. 16th 2012

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

PMaC
Performance Modeling and Characterization

What is happening on today's systems

Top 3 systems (from June 2012 Top500) have 700,000 – 1.5 M cores

- Putting together systems with lower power cores to exploit parallelism within application
- Lower the power = less capable – up to the software to run efficiently
- GPUs, BG systems, MIC, & ARM
- More complex programming models
- At 1.5 M cores – reliability and power become major issues

Moving towards exascale

- What is exascale projected to look like
  - 1000 times more compute capability than current systems
  - 1000 X cores = 1000 X reliability issues / faults

    (Mean time to failure inversely proportional to # of components)
  - 1000 X cores = 1000 X power = ~gigawatt!

THIS IS THE POWER WALL

Need new technology to overcome power wall and get to Exascale

Possible exascale technologies

- New exascale technology could mean:
  - 100-1000 X more cores
  - Simpler logic on cores to reduce power draw
  - Different memory hierarchy
  - Different functional units
  - Ability to shut off units or cores to save power

Requires a lot from application developer

What will developers need (how can tool developers help)

- Understand requirements of work and map this efficiently to simpler cores –modeling can help
- Understand how computations use hardware components and optimize to compute in power budget.
- Runtime systems which adapt and avoid hardware errors/failures – preemptive strategies

Towards exascale tool requirements

Runtime system - allows application to interact with hardware and adapt

- Need to know <u>how</u> and <u>when</u> an application is using the hardware components
  - Enable application-aware reliability decisions
  - Enable application-aware energy optimizations
  - Use power and performance models to make multi-objective optimization :

<u>**Power-Performance-Reliability**</u>

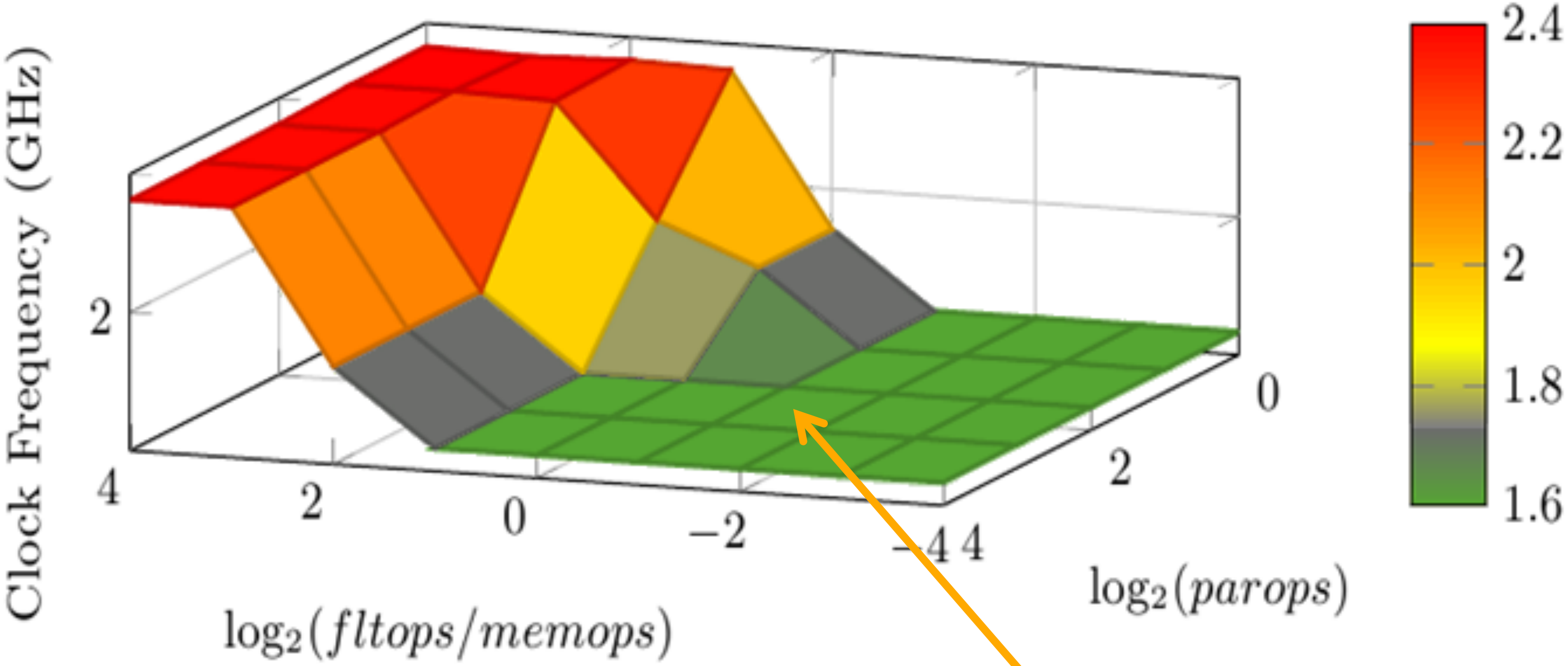
The PMaC's Green Queue Framework
(performance-power requirements)

Goal: Develop automated framework that uses power and performance models to make application-aware energy optimizations during execution (now:DVFS future: power gating)

DVFS: Reduce the speed (clock frequency) of CPU in exchange for reduced power consumption

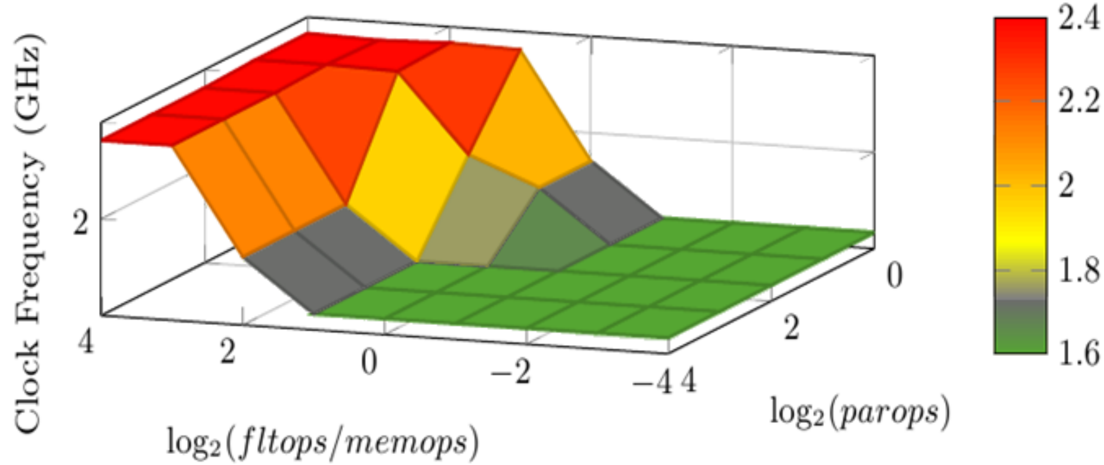
- **Different computations have different power requirements.**
- **For computations where the CPU is waiting for resources the frequency can be reduced to lower power with minimal performance impact.**

Identify the power and performance affects of different computational work



Energy savings via reduce processor frequency – minimal performance impact

Application-aware Energy Efficient HPC



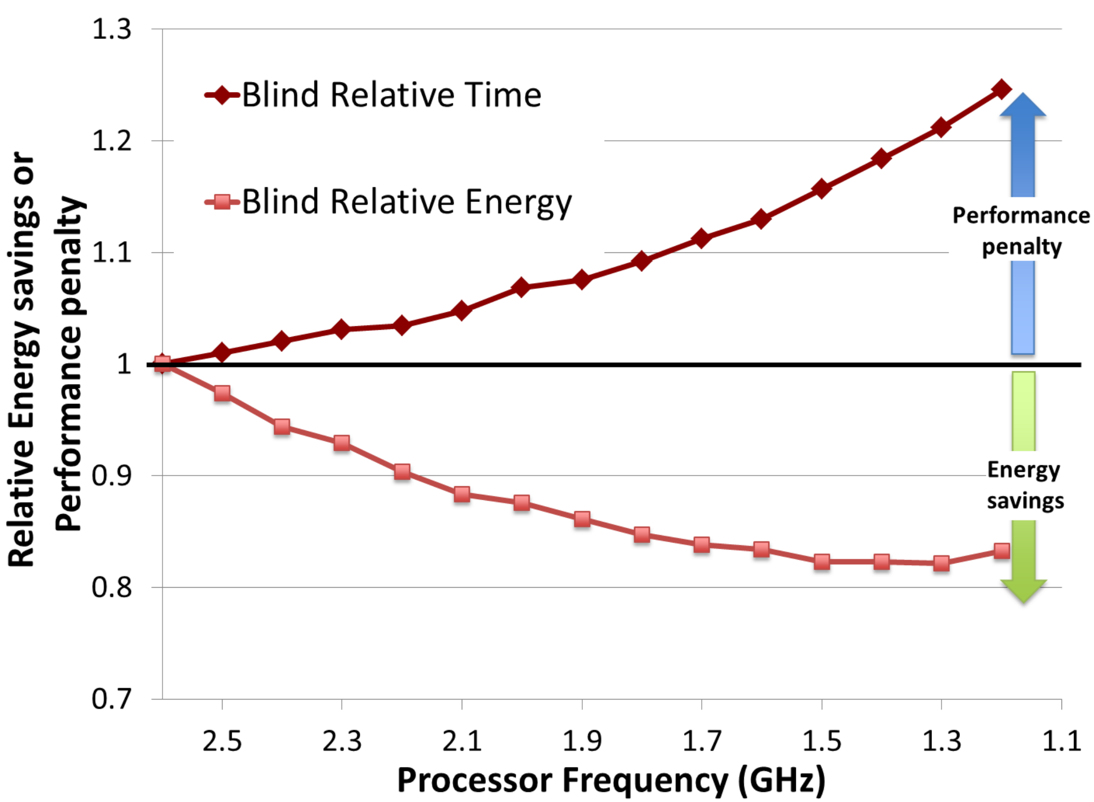
<u>HPC System</u>
Characterize the computational (& communication) patterns affect the overall power draw

<u>HPC Application</u>
Characterize the computational (& communication) behavior of application

Design software- and hardware-aware green optimization techniques to reduce HPC's energy footprint

Fine-grain Application-aware vs. Coarse-grain Application-blind DVFS

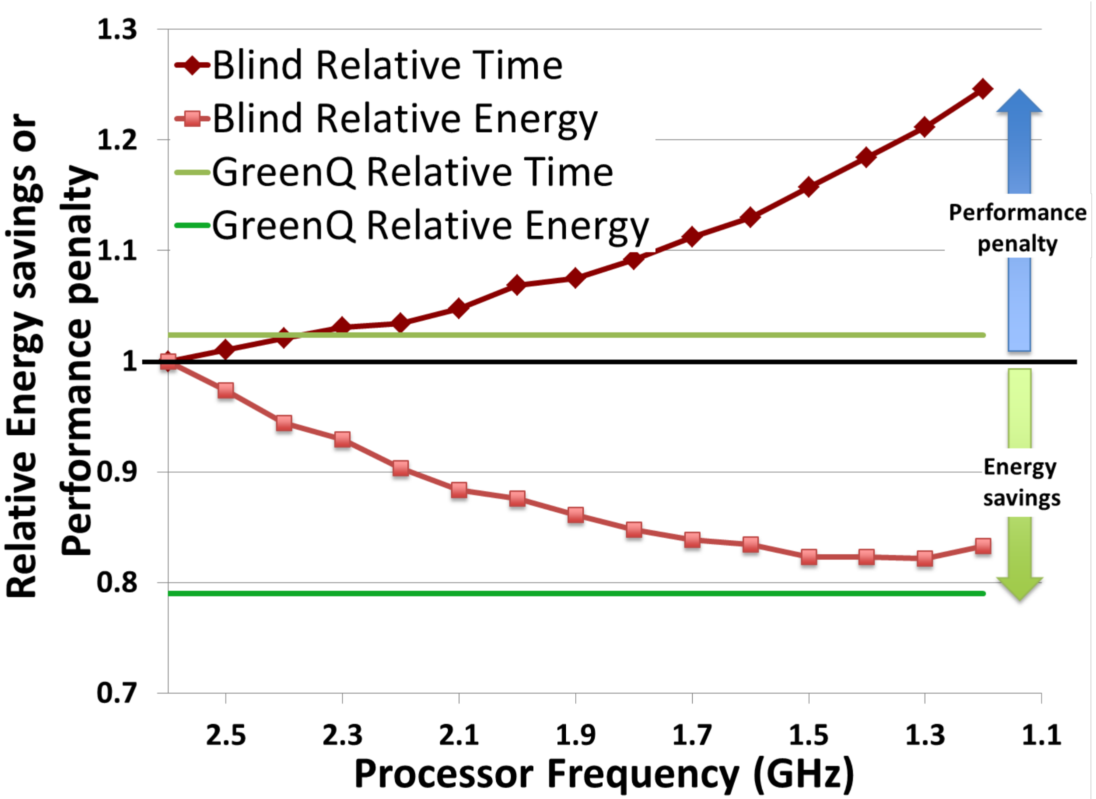
Application-aware fine-grained DVFS shows significantly less impact in performance



Fine-grain Application-aware vs. Coarse-grain Application-blind DVFS

Application-aware fine-grained DVFS shows significantly less impact in performance

2.4% vs. 21%



PMaC's Green Queue Framework
(fine-grained application-aware DVFS strategies)

HPC System
Characterize the computational (& communication) patterns affect the overall power draw

HPC Application
Characterize the computational (&communication) behavior of application

Design software- and hardware-aware green optimization techniques to reduce HPC's energy footprint

PMaC's Green Queue automated framework:

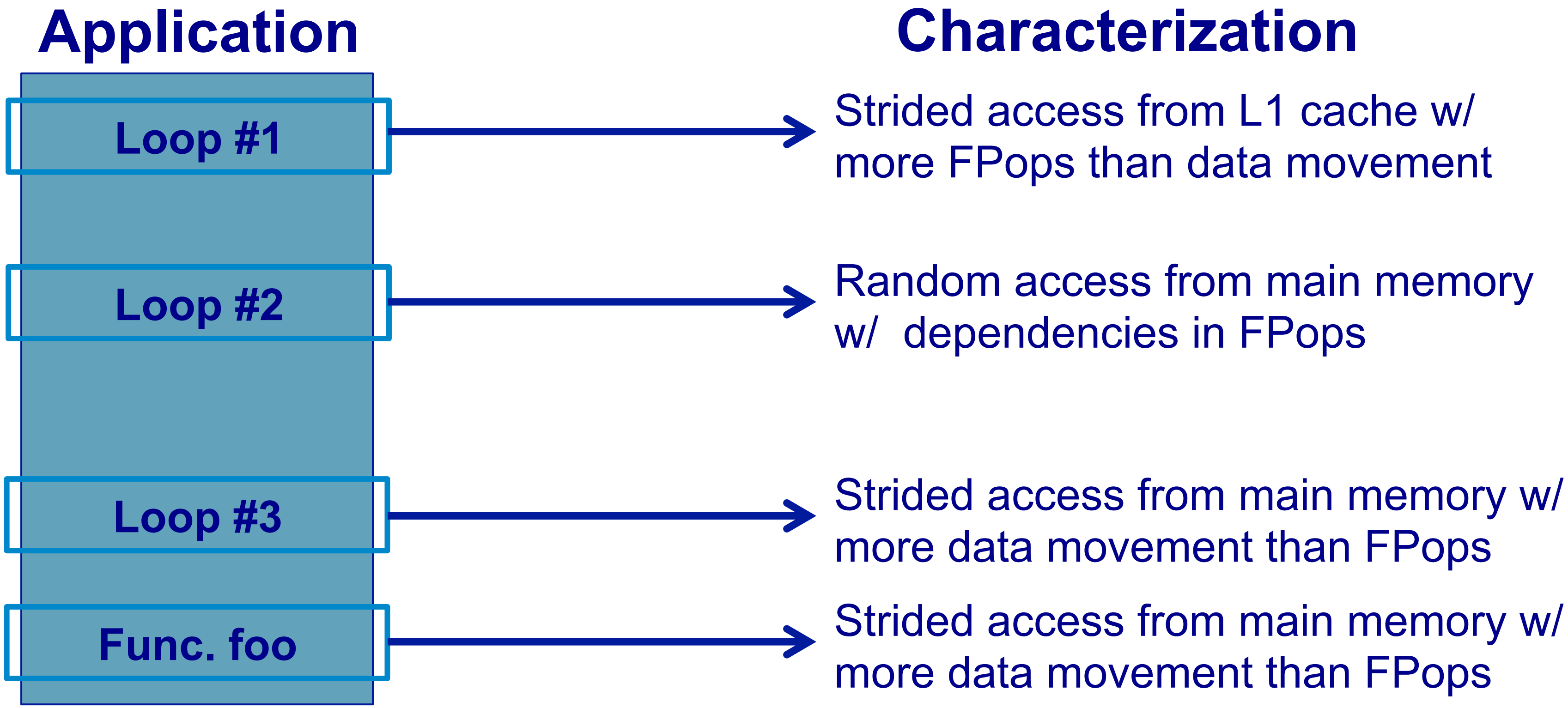
- Characterizes system's power draw behavior by running various computational work and uses to train models
- Characterizes computational work of HPC application
- Creates customize fine-grained DVFS policies for application
  - Inter-node: exploits load imbalances in HPC applications
  - Intra-node: exploits application phases where CPU is stalled waiting for resources

Application Characterization

Application characterization – fine-grained information about the communication & computation behavior of the application

- Low-level details that capture how application uses various hardware components
- Data movement on and off the processor and node
- Data locality and computational dependencies

Example of Application Characterization

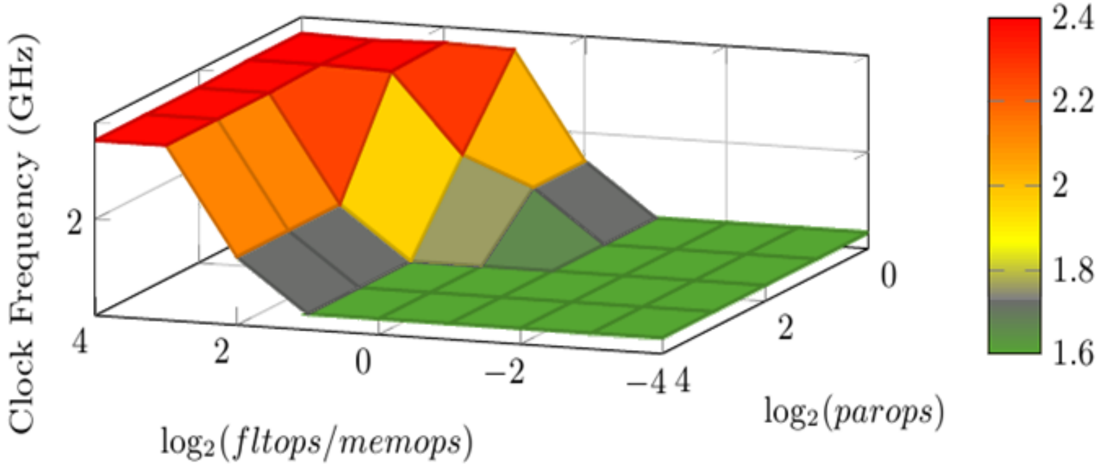


- Application characterization can be dependent on the system it is running on as well as the input set

Collecting Application Characterization Data

- Computation characterization – based on PEBIL (PMaC's Efficient Binary Instrumentor for Linux)
  - Static Analysis
    - Memory, FP operation counts
    - Operation parallelism
    - Program structure (e.g., function and loop boundaries)
  - Dynamic (runtime) analysis
    - Data locality
    - Working set size
    - Execution counts
- MPI communication characterization– based on PSiNSTracer
  - Behavior about communication

System Characterization



System characterization:

- Determine the most energy efficient frequency for range of computational work.
- Computational work focusing on-node.
- Computational work behavior that spans all HPC applications

Performance and Power Benchmarking framework

PMaC's Performance Power benchmark (P^3)

- Generates computational test loops to measure performance and power for computational space of HPC application.
- Test loops measured at different frequencies
- Test loops designed to vary different characteristics of the loop (e.g. working set size or data locality)

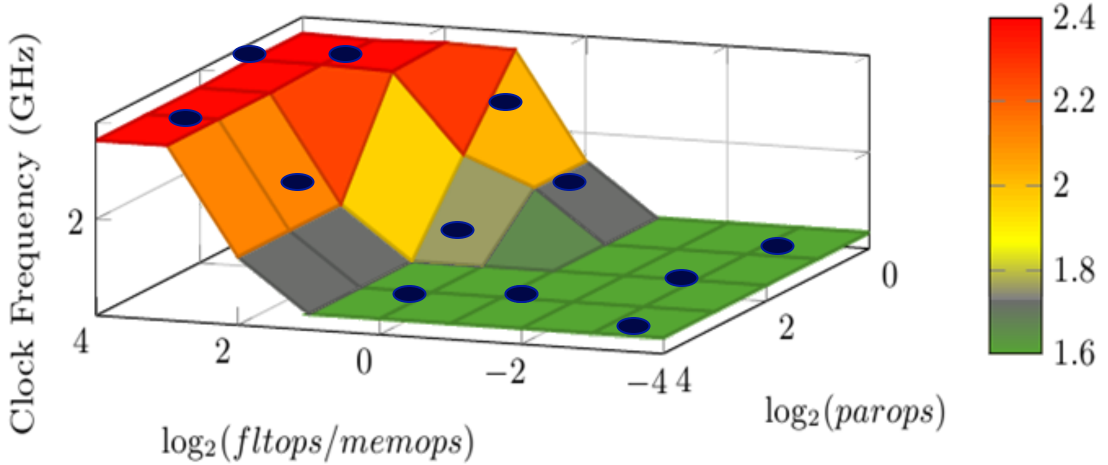
Testing space can grow to over 100K tests - weeks to run

Performance and Power models can save time

Power draw = func(computational behavior)

Why Power Models?

- Reduce the number of pcubed benchmark tests that we need to run: >100K → 3K
  - Reduces runtime from weeks to hours
- Use sampling of test runs to model remaining computation space.



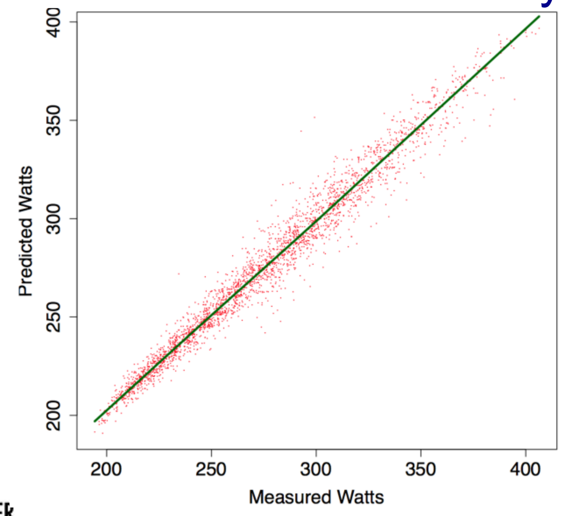
Power draw = func(computational behavior)

Developing Power Models

Power models – relate the relevant properties of a computation to the system's power response.

- Use power and performance measurements for set of benchmarks tests
- Use corresponding characterization data (data locality, data footprint, etc.) for the benchmarks
- Machine learning (Gradient Boosting Method) for constructing the power models

Power Model Accuracy

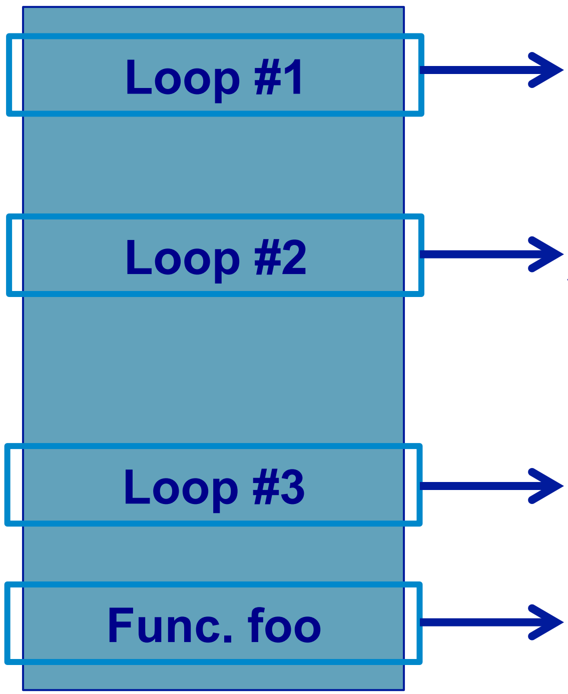


Model accuracy for power estimation: 2.2% absolute mean error

Uses for Power Models

Map the application characterization data to system characterization

Application | **Characterization**

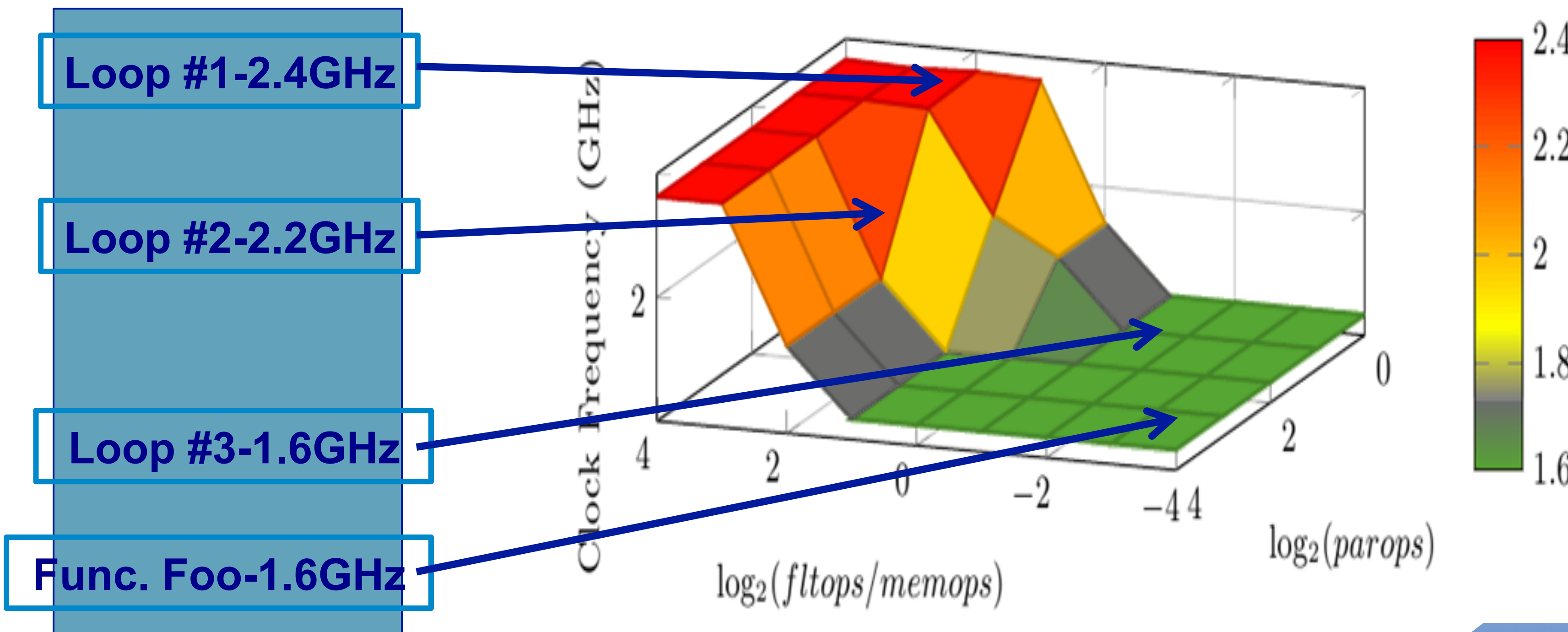


- Loop #1 → Strided access from L1 cache w/ more FPops than data movement
- Loop #2 → Random access from main memory w/ dependencies in FPops
- Loop #3 → Strided access from main memory w/ more data movement than FPops
- Func. foo → Strided access from main memory w/ more data movement than FPops

Uses for Power Models

Map the application characterization data to system characterization

Application



PMaC's Green Queue Framework
(fine-grained application-aware DVFS strategies)

HPC System
Characterize the computational (& communication) patterns affect the overall power draw

HPC Application
Characterize the computational (&communication) behavior of application

Design software- and hardware-aware green optimization techniques to reduce HPC's energy footprint

PMaC's Green Queue automated framework:

- Characterize systems power draw behavior when running various computational work using models
- Characterizes computational work of HPC application
- Creates customize fine-grained DVFS policies for application
  - Inter-node: exploits load imbalances in HPC applications
  - Intra-node: exploits application phases where CPU is stalled waiting for resources

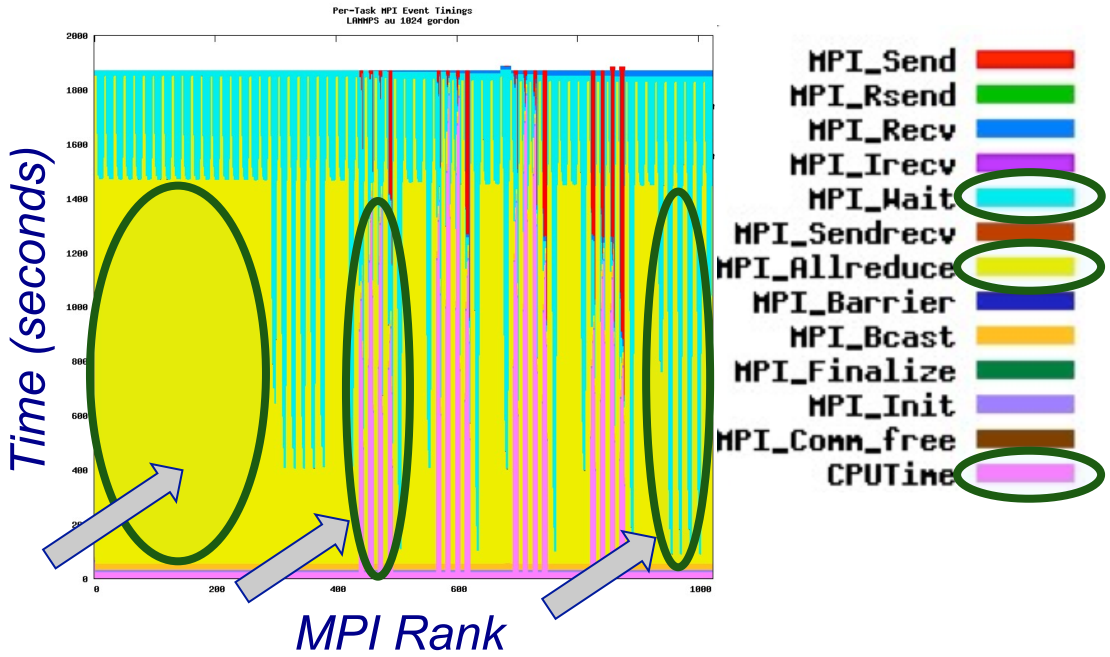
Inter-node Technique

(Focusing on load imbalance in application due to work distribution)

- MPI load imbalance: a subset of MPI processes have less work to do and wait for others thereby wasting energy
  - Could arise due to inherent nature of the problem/dataset
- Large body of research on remedying load imbalance and on exploiting the same to save energy
- Green Queue's approach is simple but we apply it at scale

Inter-node Technique

- Green Queue captures and quantifies load imbalance by profiling all MPI communications and core-level computations



- Measure the “idleness” for each core by taking a simple ratio of its computation time to the computation time of the busiest core

Intra-node Technique

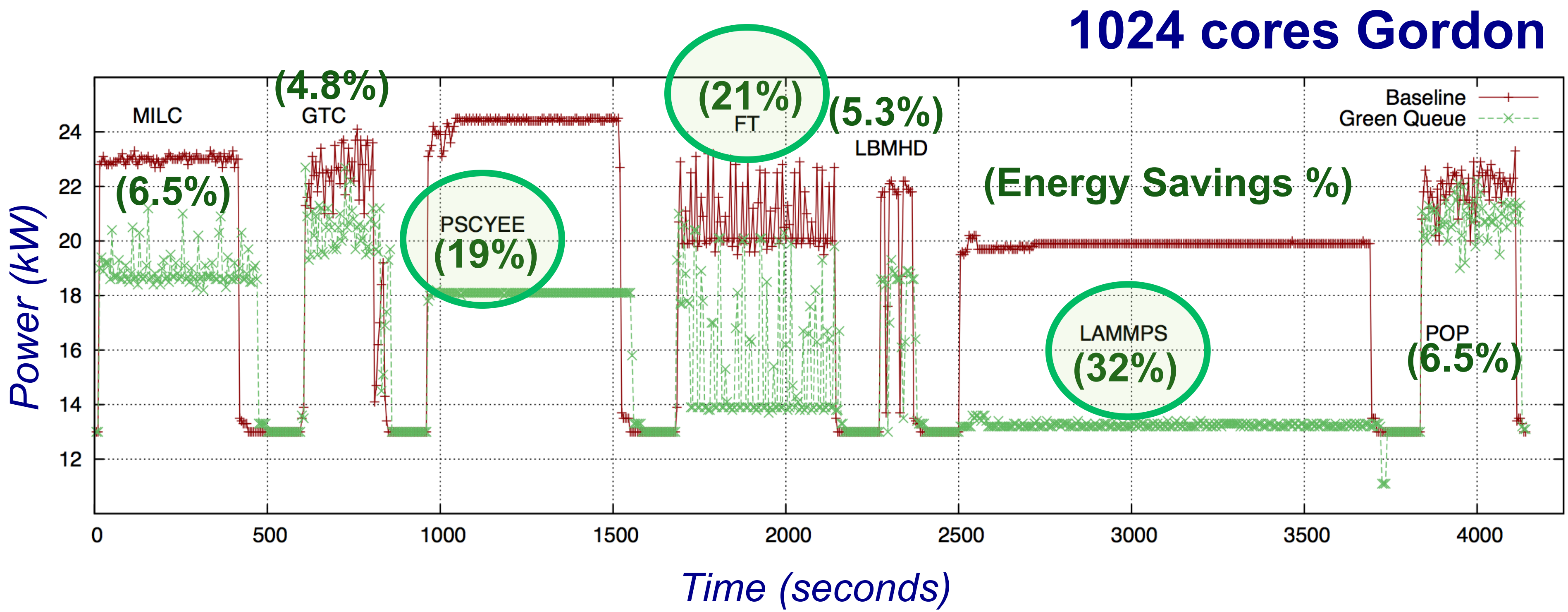
(Focusing on work done on processor in between communication events)

- Memory subsystem's performance is often the bottleneck for node-level performance
  - CPU may stall while the hardware satisfies memory requests from off-chip (e.g., L3 cache or main memory)
  - Lower the clock frequency during the phases where these stalls are significant
- *Phase* is a path through the program's control flow graph which exhibits uniform runtime behavior while on that path
- Green Queue uses the **structure of the application** to identify *all phases*
  - Phase detection mechanism crosses loop and function boundaries

Results – Experimental Setup

- Gordon, an Intel Sandybridge based supercomputer
  - Dual socket nodes. 8-core processor on each socket. Each socket independently can be set to run on one of the 15 available clock frequencies
  - Nodes configured as a 3D torus. QDR Infiniband network
- Experiments run using a single rack of Gordon (1024 cores)
  - Not a limitation of this work
- Rack-level power measurement obtained from PDUs
- Large scale applications and benchmarks – MILC, SWEEP3D, GTC, LBMHD, LAMMPS, POP, WRF, HYCOM, CG, FT, MG

Results – Overall & Discussion



- Ongoing work
  - Merge inter and intra node techniques

Contributions & Conclusions

- Phase detection based on the structure of the program
- Optimal frequency assignment for *all* phases in an application
- Framework deployed at scale on current generation supercomputer

Tiwari A, Laurenzano M, Peraza J, Carrington L, Snavely A: **Green Queue: Customized Large-scale Clock Frequency Scaling**. *CGC 2012* 2012.

Peraza J, Tiwari A, Laurenzano M, Carrington L, Snavely A: **PMaC's Green Queue: A Framework for Selecting Energy Optimal DVFS Configurations in Large Scale MPI Applications**. *Concurrency and Computation: Practice and Experience* 2012.

For details on PMaC Lab's recent energy efficiency work, please visit: http://www.sdsc.edu/pmac/

Or e-mail: lcarring@sdsc.edu

Looking Ahead

- Green Queue start for application- and hardware-aware runtime system (power-performance)
- Extensions to reliability required for exascale
- Need APIs to access the more hardware information like errors, power, etc.
- Runtime system –Support for fine-grained software-driven management– give more control to the software
  - DVFS
  - Power gating-power planes
  - more control of hardware

Questions ?