SUPER

INSTITUTE FOR SUSTAINED PERFORMANCE,
ENERGY, AND RESILIENCE

SUPER Energy-efficiency HPC Research

Laura Carrington (lead), Ananta Tiwari
UCSD/PMaC

Rob Fowler
RENCI

Dan Terpstra
UTK

Abstract: SUPER's Energy thrust is charged with understanding how computation and communication patterns affect the overall energy requirements of HPC applications. We then leverage this understanding to design software- and hardware-aware optimization techniques that reduce the DOE's HPC energy footprint. Two focus areas have emerged within this thrust: software solutions that provide fine-grained access to the power measurements and energy efficiency research that utilizes these measurements to develop green optimization strategies. We highlight recent accomplishments in each area and present empirical results that illustrate SUPER's contributions in minimizing DOE's HPC energy requirements.

Energy-Constrained Computation: Measurement and Adaptation

Technological and Commercial Imperatives Are Driving the Problem.

Moore's law → The number of gates/chip grows.

- Vendor compete to build and sell increasingly complex high-end chips.

Denard scaling → If geometry, voltages, and clocks scale co-linearly, power density remains constant.

- While these scaling properties worked, high-end chips could run at increasing speed and still be adequately cooled. (This is why Moore's law translated into faster, not just bigger, computers.)
- In the past decade, gate insulator thickness, voltages stopped scaling. Chips become power/cooling-limited.
- Instruction-level parallel designs became dominated by "housekeeping" overhead.

The multicore (and "System on a Chip") response.

- Add more power-efficient cores and other units (memory controllers, NICS, GPUs) to chips.
- Cores can still run fast, but sustained speed is limited power/cooling of chip package → Sell chips using "de-rated" specs.
  - Intel E5-2680 is nominally 2.7GHz but individual cores run easily at TurboBoost speed of 3.5GHz.
  - AMD "computational sprinting" is similar.
- Entering an era of "dark" or "dim" silicon in which parts of chips are shut down or run slowly.

Impact on HPC of power/cooling constrained computing.

- On chip controllers automatically react to thermal state by adjusting speeds and feeds of major components. Example, Intel's controller uses a "running average power limit" (RAPL) model.
- The performance advantages of hardware adaptation are significant, but
  →Performance is becoming non-deterministic, varies by : socket, core, location in rack, time of day, other programs.
- Measurement, monitoring, and analysis tools are needed to assess the impact.
  - Adaptation requires real time feedback tools.
- System and application code need to interact flexibly with adaptive hardware.

Challenges and Questions

How should power and thermal effects be incorporated in performance experiments? Used in tuning exercises?

How can we use the interfaces to the package controllers to better manage power states?

- Can this be made portable?

Application and library support:

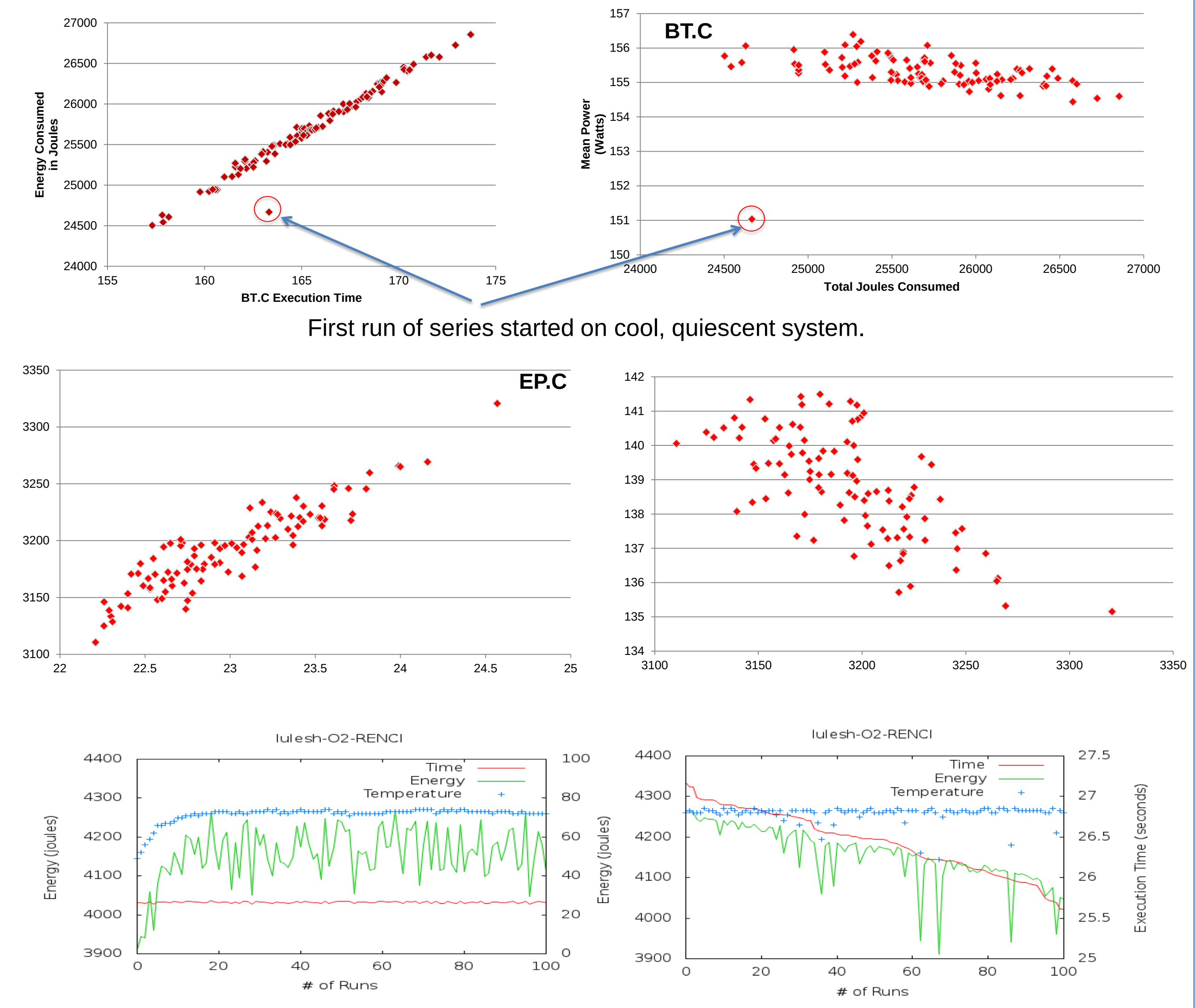
- Thread management mechanisms and policies to improve throughput while saving power?

The V part of DVFS is becoming less effective and is package-wide. Core idling and "clock modulation" (in Intel-speak) is effective on a per core basis. Can we promote this?

Does better (water) cooling at the package level make the problem easier?

Performance and Energy Variability

OpenMP benchmarks run 100 times on dual E52680 blade.
Used GOMP. No application power management.
Power/energy/temperature collected using RAPL.



First run of series started on cool, quiescent system.

Sorted by start time

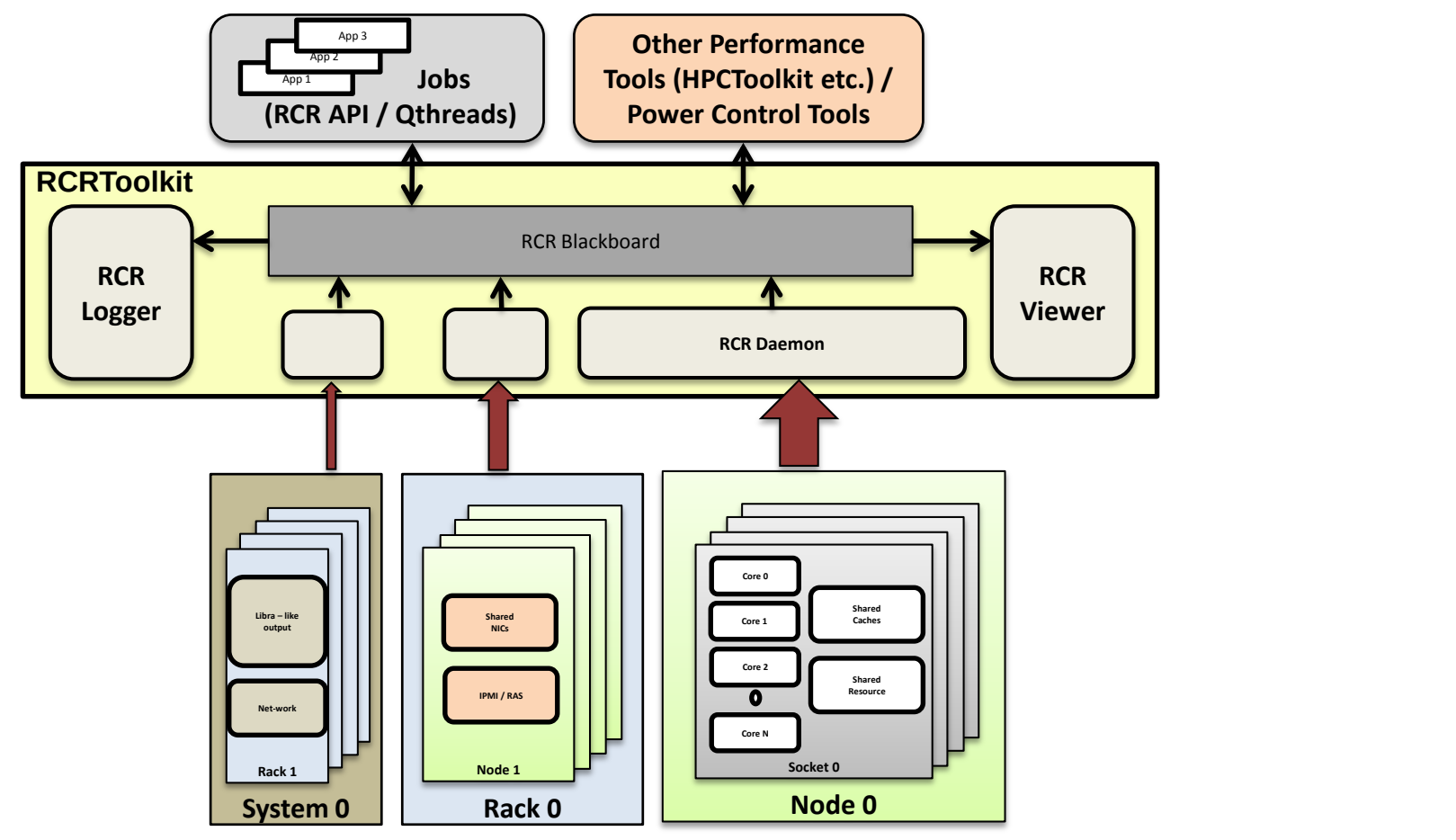
Sorted by elapsed time

Real Time Measurement/Analysis to Support Adaptation.

Resource-Centric Reflection Tools

An infrastructure for real time collection and analysis of node- and system-wide performance information to guide runtime adaptation.

- Capture "uncore" counter information for shared resources.
  - This includes power, energy, and temperature data.
- Capture OS and Network information.
- Share across the entire software stack through a blackboard.



App 3, App 2, App 1, Jobs (RCR API / Qthreads)

Other Performance Tools (HPCToolkit etc.) / Power Control Tools

RCRToolkit

RCR Logger

RCR Blackboard

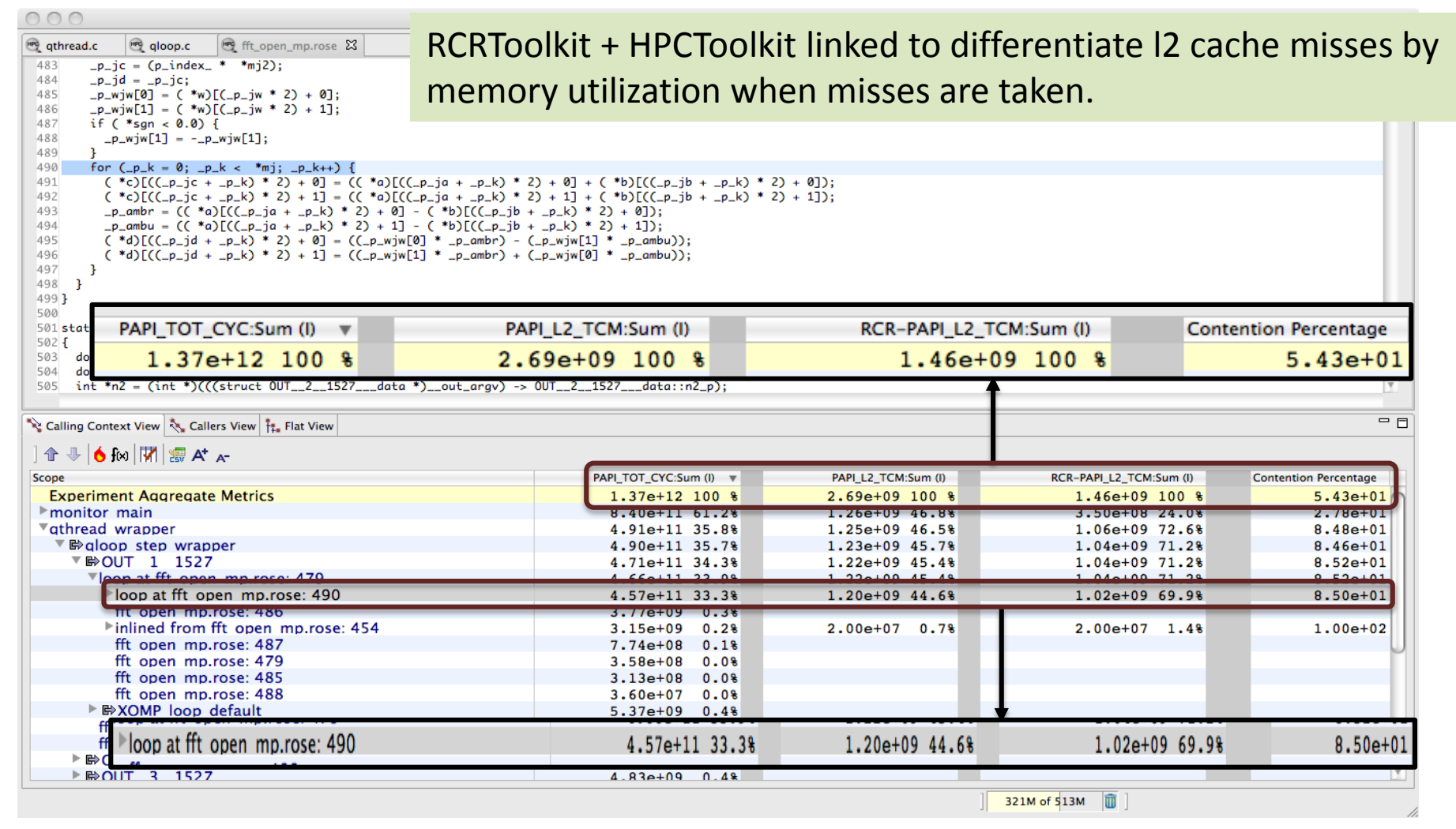
RCR Daemon

RCR Viewer

System 0: Rack 1, Libra – like output, Net-work

Rack 0: Node 1, Shared NICs, IPMI / RAS

Node 0: Socket 0, Core 0, Core 1, Core 2, Core N, Shared Caches, Shared Resource



RCRToolkit + HPCToolkit linked to differentiate l2 cache misses by memory utilization when misses are taken.

```
    _p_jc = (p_index_ *  *mj2);
    _p_jd = _p_jc;
    _p_wjw[0] = ( *w)[(_p_jw * 2) + 0];
    _p_wjw[1] = ( *w)[(_p_jw * 2) + 1];
    if ( *sgn < 0.0) {
      _p_wjw[1] = -_p_wjw[1];
    }
    for (_p_k = 0; _p_k <  *mj; _p_k++) {
      ( *c)[((_p_jc + _p_k) * 2) + 0] = (( *a)[((_p_ja + _p_k) * 2) + 0] + ( *b)[((_p_jb + _p_k) * 2) + 0]);
      ( *c)[((_p_jc + _p_k) * 2) + 1] = (( *a)[((_p_ja + _p_k) * 2) + 1] + ( *b)[((_p_jb + _p_k) * 2) + 1]);
      _p_ambr = (( *a)[((_p_ja + _p_k) * 2) + 0] - ( *b)[((_p_jb + _p_k) * 2) + 0]);
      _p_ambu = (( *a)[((_p_ja + _p_k) * 2) + 1] - ( *b)[((_p_jb + _p_k) * 2) + 1]);
      ( *d)[((_p_jd + _p_k) * 2) + 0] = ((_p_wjw[0] * _p_ambr) - (_p_wjw[1] * _p_ambu));
      ( *d)[((_p_jd + _p_k) * 2) + 1] = ((_p_wjw[1] * _p_ambr) + (_p_wjw[0] * _p_ambu));
    }
  }
}

stat
{
  do
  do
  int *n2 = (int *)(((struct OUT__2__1527___data *)__out_argv) -> OUT__2__1527___data::n2_p);
```

PAPI_TOT_CYC:Sum (I)	PAPI_L2_TCM:Sum (I)	RCR-PAPI_L2_TCM:Sum (I)	Contention Percentage
1.37e+12 100 %	2.69e+09 100 %	1.46e+09 100 %	5.43e+01

Scope	PAPI_TOT_CYC:Sum (I)	PAPI_L2_TCM:Sum (I)	RCR-PAPI_L2_TCM:Sum (I)	Contention Percentage
Experiment Aggregate Metrics	1.37e+12 100 %	2.69e+09 100 %	1.46e+09 100 %	5.43e+01
monitor main	8.40e+11 61.2%	1.26e+09 46.8%	3.50e+08 24.0%	2.78e+01
qthread wrapper	4.91e+11 35.8%	1.25e+09 46.5%	1.06e+09 72.6%	8.48e+01
qloop step wrapper	4.90e+11 35.7%	1.23e+09 45.7%	1.04e+09 71.2%	8.46e+01
OUT 1 1527	4.71e+11 34.3%	1.22e+09 45.4%	1.04e+09 71.2%	8.52e+01
loop at fft open mp.rose: 479	4.66e+11 33.9%	1.22e+09 45.4%	1.04e+09 71.2%	8.52e+01
loop at fft open mp.rose: 490	4.57e+11 33.3%	1.20e+09 44.6%	1.02e+09 69.9%	8.50e+01
fft open mp.rose: 486	3.77e+09 0.3%			
inlined from fft open mp.rose: 454	3.15e+09 0.2%	2.00e+07 0.7%	2.00e+07 1.4%	1.00e+02
fft open mp.rose: 487	7.74e+08 0.1%			
fft open mp.rose: 479	3.58e+08 0.0%			
fft open mp.rose: 485	3.13e+08 0.0%			
fft open mp.rose: 488	3.60e+07 0.0%			
XOMP loop default	5.37e+09 0.4%			
OUT 3 1527	4.83e+09 0.4%			

loop at fft open mp.rose: 490	4.57e+11 33.3%	1.20e+09 44.6%	1.02e+09 69.9%	8.50e+01

Capturing power usage via PAPI

Power measurement is essential to all energy efficient research because it helps us understand, exploit and limit hardware policies for transparent and adaptive power control. Such understanding allows the implementation of more effective hardware-software co-managed energy optimizations.

PAPI provides access to hardware performance counters, enabling software engineers to measure relationships between software performance and processor or other hardware events.

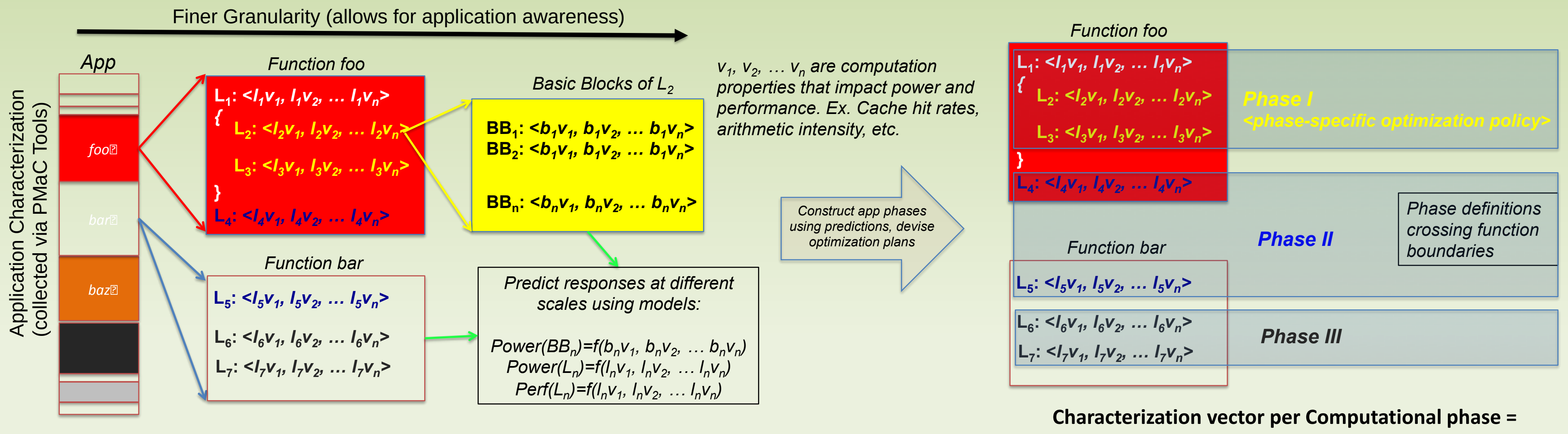
Component PAPI can be extended to allow energy and power measurements without any changes to core infrastructure or measurement tools. Current users of PAPI on HPC systems can now analyze power and energy with little additional effort. Recent and ongoing component development supports measuring energy and power usage via a variety of sources including:

- MIC-side power measurement component
- Host-side MIC power measurement component
- PowerMon2 : RENCI's real-time power monitoring card
- RAPL: Intel's Running Average Power Limit model
- NVML: NVIDIA's Management Library for Tesla cards
- Watts Up?: AC line voltage and current monitor

In discussion with IBM for power measurement via EMON2 & PAPI on BG/Q

Optimizing energy usage via models for the performance & power of computational phases of HPC applications

PMaC's Green Queue Framework



Finer Granularity (allows for application awareness)

Application Characterization (collected via PMaC Tools)

App: foo(), bar(), baz()

Function foo
L_1: $<l_1v_1, l_1v_2, \ldots l_1v_n>$
{
L_2: $<l_2v_1, l_2v_2, \ldots l_2v_n>$
L_3: $<l_3v_1, l_3v_2, \ldots l_3v_n>$
}
L_4: $<l_4v_1, l_4v_2, \ldots l_4v_n>$

Function bar
L_5: $<l_5v_1, l_5v_2, \ldots l_5v_n>$
L_6: $<l_6v_1, l_6v_2, \ldots l_6v_n>$
L_7: $<l_7v_1, l_7v_2, \ldots l_7v_n>$

Basic Blocks of L_2
BB_1: $<b_1v_1, b_1v_2, \ldots b_1v_n>$
BB_2: $<b_1v_1, b_1v_2, \ldots b_1v_n>$
BB_n: $<b_nv_1, b_nv_2, \ldots b_nv_n>$

$v_1, v_2, \ldots v_n$ are computation properties that impact power and performance. Ex. Cache hit rates, arithmetic intensity, etc.

Predict responses at different scales using models:
$Power(BB_n)=f(b_nv_1, b_nv_2, \ldots b_nv_n)$
$Power(L_n)=f(l_nv_1, l_nv_2, \ldots l_nv_n)$
$Perf(L_n)=f(l_nv_1, l_nv_2, \ldots l_nv_n)$

Construct app phases using predictions, devise optimization plans

Function foo
L_1: $<l_1v_1, l_1v_2, \ldots l_1v_n>$
{
L_2: $<l_2v_1, l_2v_2, \ldots l_2v_n>$
L_3: $<l_3v_1, l_3v_2, \ldots l_3v_n>$
}
L_4: $<l_4v_1, l_4v_2, \ldots l_4v_n>$

Phase I
<phase-specific optimization policy>

Function bar
L_5: $<l_5v_1, l_5v_2, \ldots l_5v_n>$
L_6: $<l_6v_1, l_6v_2, \ldots l_6v_n>$
L_7: $<l_7v_1, l_7v_2, \ldots l_7v_n>$

Phase II

Phase definitions crossing function boundaries

Phase III

Characterization vector per Computational phase =

$$<P_1v_1, P_1v_2, \ldots P_1v_n>$$

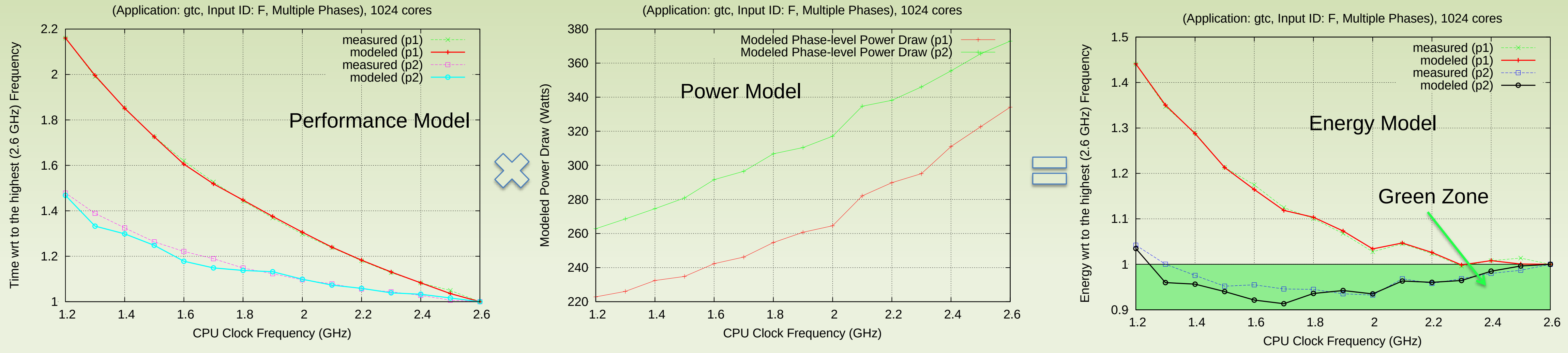
A fully automated framework that utilizes fine-grained application characterizations and power and performance models to devise and deploy energy efficient policies

An application's computational behavior is captured by a series of characterization vectors; these vectors are inputs to power and performance models.

Phase Characterization

- *Collecting data at various scales (from function to loop to basic blocks) allows flexible definition and construction of phases. A computational phase is defined as a window in the execution of a program where some relevant characteristics of the computation remain fairly unchanged. Phase boundaries can change for different types of optimizations*
- *Identify phases based on some behavior of interest:*
  Data footprint, L3 Misses, Power draw, Accelerator performance, vectorization

Case Study: Modeling Two Computational Phases of GTC (1024 core run; Gordon)



These plots show the measured and modeled behavior of two different computational phases of GTC on 1024 cores. We use the phase-level characterization data to predict performance and power responses of the two phases. Predictions for power and performance are then combined to predict energy. We note that for this graph, we normalize the energy required to run each phase at all available frequencies with respect to the energy required to run the phase at the highest frequency. A ratio of less than 1 for a given frequency/phase pair means that we can conserve energy for that phase by running it at that frequency compared to running that phase on the default system frequency. The green zone marks those frequency selections that provide energy savings and illustrate how the models enable fine-grained customized DVFS settings for an application's individual computational phases.

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics).