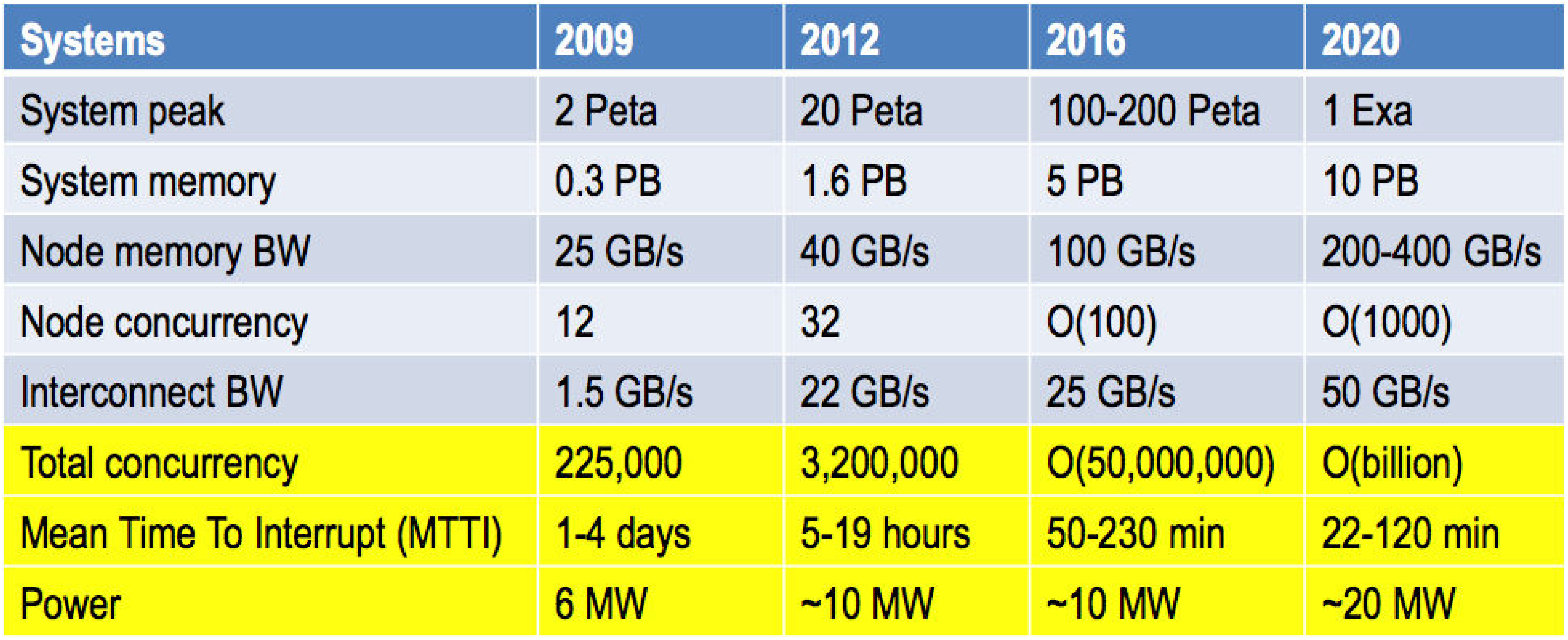
The Interplay between Energy Efficiency and Resilience for Scalable High Performance Computing Systems

Li Tan

PhD Final Defense

Exascale Roadmap

Source: http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/aug12/2012-ECI-ASCAC-v4.pdf

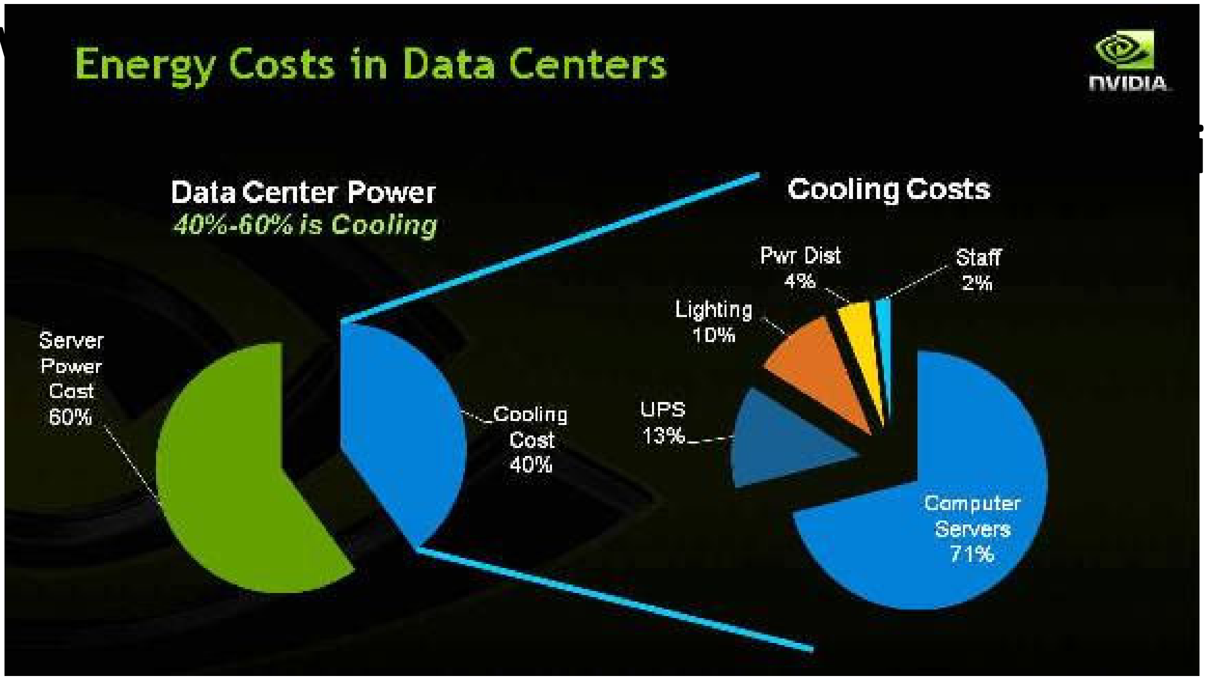


Systems	2009	2012	2016	2020
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa
System memory	0.3 PB	1.6 PB	5 PB	10 PB
Node memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	22 GB/s	25 GB/s	50 GB/s
Total concurrency	225,000	3,200,000	O(50,000,000)	O(billion)
Mean Time To Interrupt (MTTI)	1-4 days	5-19 hours	50-230 min	22-120 min
Power	6 MW	~10 MW	~10 MW	~20 MW

- Explosive increase in parallelism
- *Large increase in the number of failures*
- *Large increase in the power costs*

Energy and Resilience Concerns in HPC

- Power and energy costs of high performance computing systems are a growing severity nowadays → *operating costs* and *system reliability*
  - AvgPwr of top 5 supercomputers (TOP500)→10.1MW
  - 20MW FLOPS)
  - Over ing costs



High Vulnerability of Large-Scale HPC Systems

- Failure Rate Explosion
  - *Small on a single node, increasingly susceptible at scale*
  - Up to 1,700 *ECC memory errors* in 2 months for a 692-node (22,144 cores in total) cluster at PNNL [1]
  - K computer (1st on TOP500 in 2011): *hardware failure* rate of up to 3% + affected by 70 *soft errors* in 1 month [2]
  - A 128,000-node BlueGene/L system: 1 *soft error* in the L1 cache every 4-6 hours due to radioactive decay [3]
  - What about the forthcoming exascale systems in 2020?

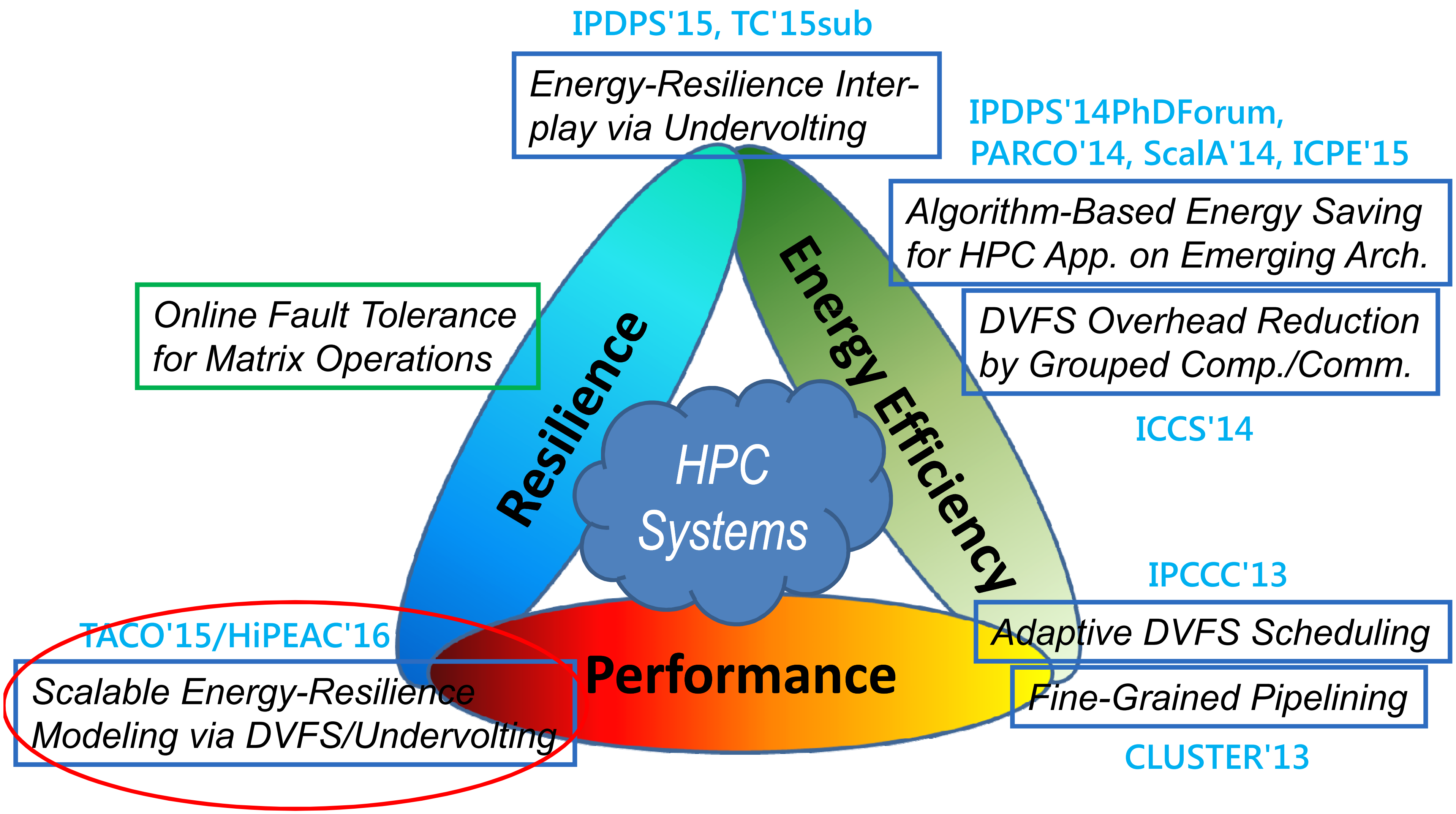
- More Error-Prone Components
  - Memory bit-flips, CPU/GPU logic errors, FPGA soft errors

[1] PIC: Pacific Northwest National Laboratory Institutional Computing. https://cvs.pnl.gov/PIC/wiki/PicCompute.
[2] Keiji Yamamoto et al., The K computer Operations: Experiences and Statistics, in Proc. ICCS, 2014, pp. 576–585.
[3] Greg Bronevetsky and Bronis de Supinski, Soft error vulnerability of iterative linear algebra methods, In Proc. ICS, 2008, pp. 155-164.

Interplay: Energy Efficiency and Resilience

- Motivation
  - There exist *entangled effects* between the two: Some HPC parameters involved in both (f/V, number of cores)
  - Improving one does not *necessarily* improve the other
  - *Goal*: The *optimal* HPC configuration that can *balance* the trade-offs (e.g., minimizing energy with resilience)
- Limitations of Related Work
  - Little has been done to investigate this issue both theoretically and empirically
  - Little has been done on this for large-scale HPC systems

Research in PhD


IPDPS'15, TC'15sub
Energy-Resilience Inter-play via Undervolting
IPDPS'14PhDForum, PARCO'14, ScalA'14, ICPE'15
Algorithm-Based Energy Saving for HPC App. on Emerging Arch.
DVFS Overhead Reduction by Grouped Comp./Comm.
ICCS'14
Online Fault Tolerance for Matrix Operations
Resilience
Energy Efficiency
HPC Systems
Performance
IPCCC'13
Adaptive DVFS Scheduling
Fine-Grained Pipelining
CLUSTER'13
TACO'15/HiPEAC'16
Scalable Energy-Resilience Modeling via DVFS/Undervolting

Amdahl's Law and Karp-Flatt Metric

- Two Classic Metrics Quantifying Perf. of Parallel Sys.
  - Amdahl's Law (basic model for processor performance)

$$\text{Speedup}_{\text{a}} = \frac{T_s + T_p}{T_s + \frac{T_p}{P}} = \frac{(1-\alpha)T + \alpha T}{(1-\alpha)T + \frac{\alpha T}{P}} = \frac{1}{1-\alpha+\frac{\alpha}{P}}$$

  - Karp-Flatt Metric (consider parallel overhead eg. comm.)

$$\text{Speedup}_{\text{kf}} = \frac{T_s + T_p}{T_s + \frac{T_p}{P} + T_{comm}} = \frac{(1-\alpha)T + \alpha T}{(1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P)}$$
$$= \frac{1}{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}$$

  - Problem Scope

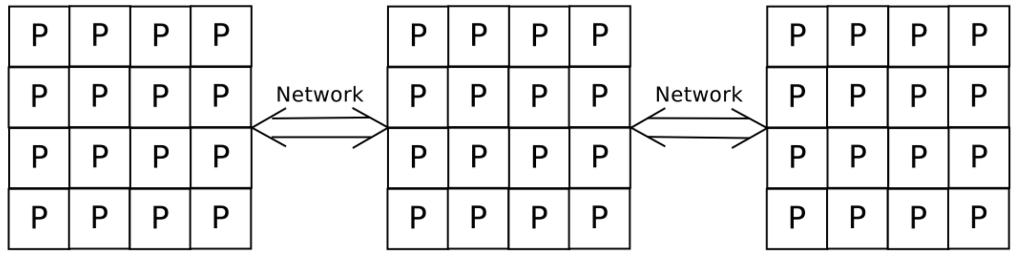


Fig. 2. Investigated Architecture – Symmetric Multicore Processors Interconnected by Networks.

Extended Amdahl’s Law and Karp-Flatt Metric

- Incorporate Power/Energy Efficiency and Resilience
  - Extended Amdahl’s Law for Power Efficiency (CPU)

$$\texttt{Power} = \frac{Energy}{Time} = \frac{\boxed{(Q + (P-1)\mu Q)(1-\alpha)T} + \boxed{PQ\frac{\alpha T}{P}} + \boxed{P\mu Q\kappa(N,P)}}{(1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P)}$$

$$= Q \times \frac{(1+\mu(P-1))(1-\alpha) + \alpha + \mu P\frac{\kappa(N,P)}{T}}{(1-\alpha) + \frac{\alpha}{P} + \frac{\kappa(N,P)}{T}}$$

  - Extended Karp-Flatt Metric for Speedup with Resilience

$$T_{cr} = \frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)} - 1)\frac{T}{\tau}$$ (Checkpoint/Restart)

$$\texttt{Speedup}^{\texttt{cr}}_{\texttt{kf}} = \frac{T_{orig}}{T_{cr}} = \frac{(1-\alpha)T + \alpha T}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)} - 1)\frac{\boxed{(1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P)}}{\tau}}$$

$$= \frac{1}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)} - 1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}}$$

Integrated Energy Efficiency for HPC Systems

- Quantitatively Modeling the Integrated Energy Efficiency with Resilience for Typical HPC Scenarios

$$\frac{\text{Perf}}{\text{Watt}} = \frac{\texttt{Speedup}}{\texttt{Power}}$$

  - HPC Runs + No Faults + No DVFS + No Undervolting
  - HPC Runs + Faults & C/R + No DVFS + No Undervolting
  - HPC Runs + Faults & C/R + DVFS + No Undervolting
  - HPC Runs + Faults & C/R + No DVFS + Undervolting (Increased Failure Rates)

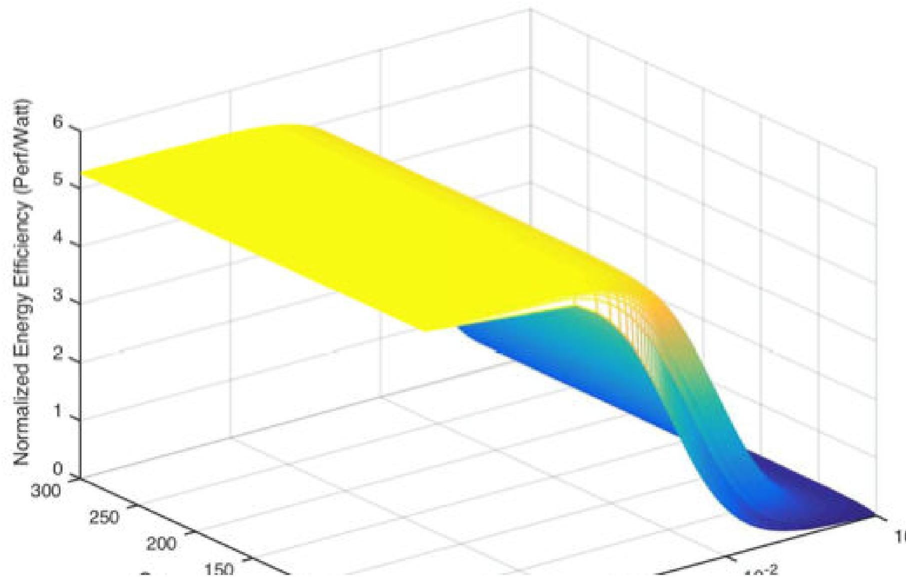
Optimizing Integrated Energy Efficiency

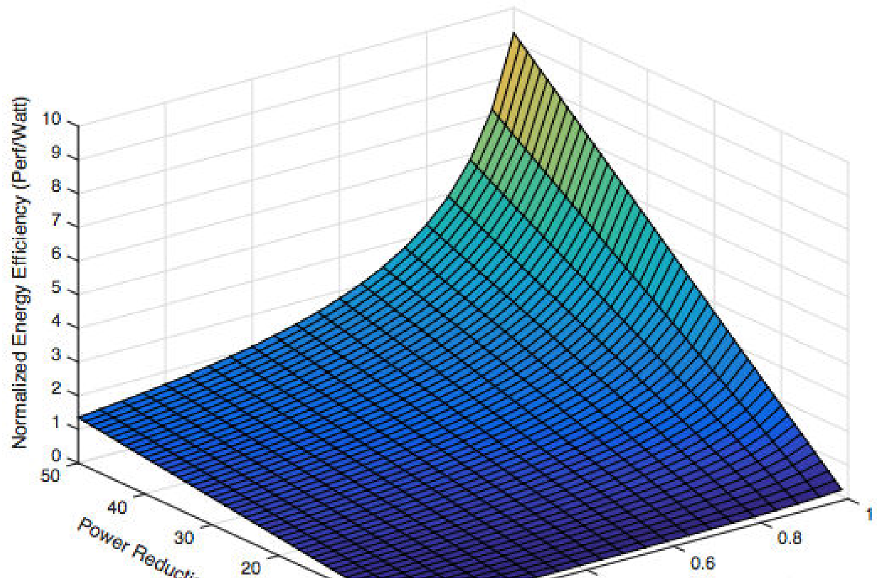
- Objective Function of Optimization

$$\frac{\text{Perf}}{\text{Watt}} = \frac{\texttt{Speedup}}{\texttt{Power}}$$

- HPC Parameters in Concern(Perf/Energy/Resilience)

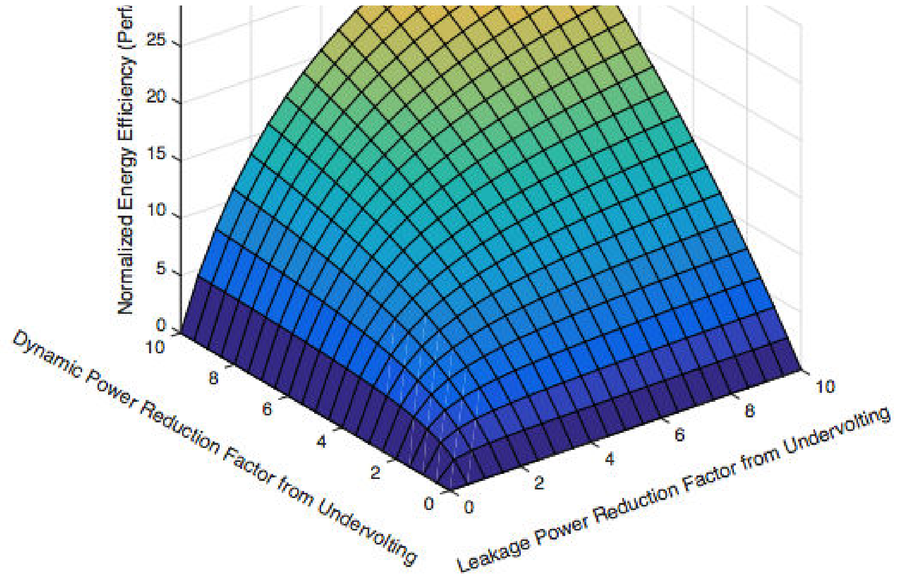
Typical HPC Parameters	Formulation	Dimension in HPC
Checkpoint/Restart Overhead	C / R	Performance/Resilience
Checkpoint Interval	τ	Performance/Resilience
Failure Rate	λ	Performance/Resilience
Number of Cores	P	Performance/Energy
Problem Size	N	Performance/Energy
Frequency/Voltage	f / V	Performance/Energy/Resilience

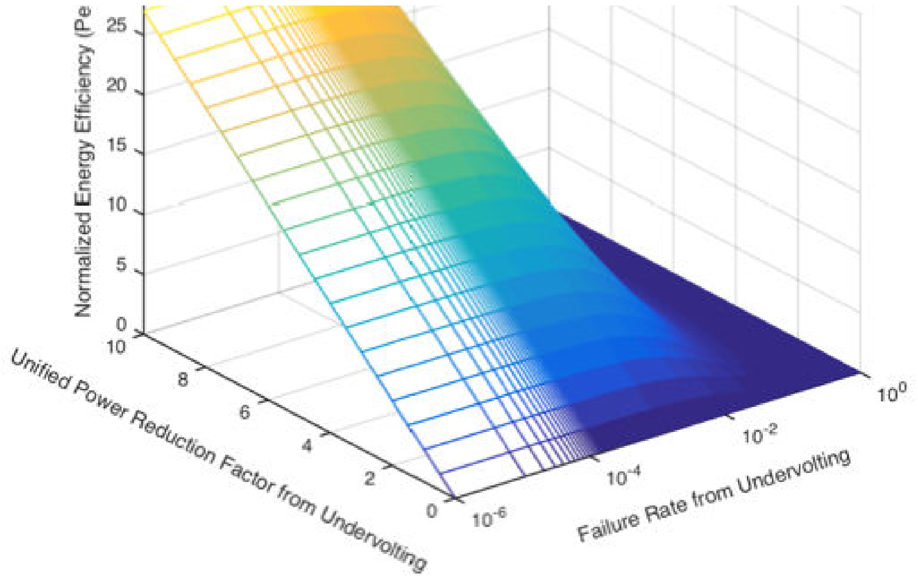




$$\frac{\text{Perf}}{\text{Watt}} = \frac{1}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}} \times \frac{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda(1+\frac{C}{\tau})(C+R)}$$

$$= \frac{1}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda(1+\frac{C}{\tau})(C+R)}$$





Benchmarks and Experimental Setup

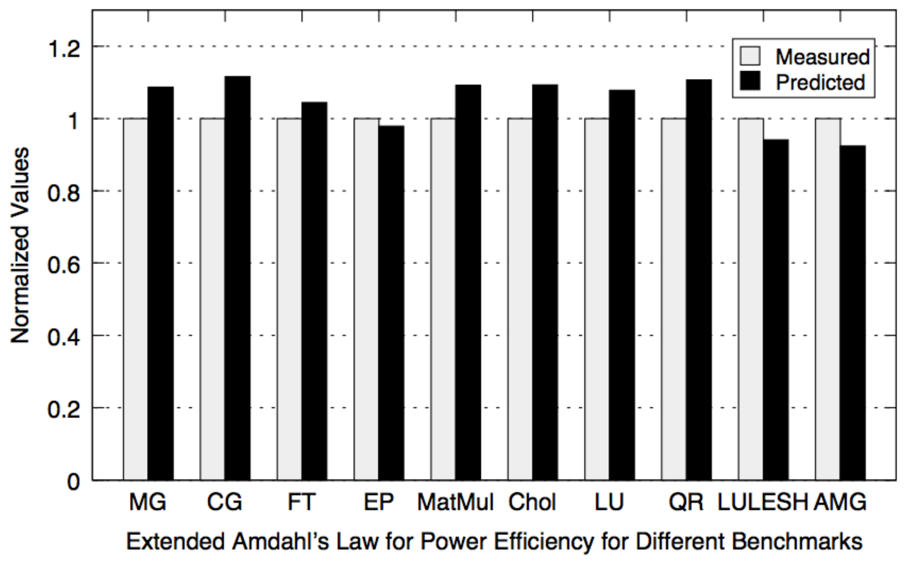
Table I. Benchmark details. From left to right: benchmark name, benchmark suite, benchmark description and test case used, problem domain, lines of code in the benchmark, parallelization system employed, and parallelized code percentage relative to the total.

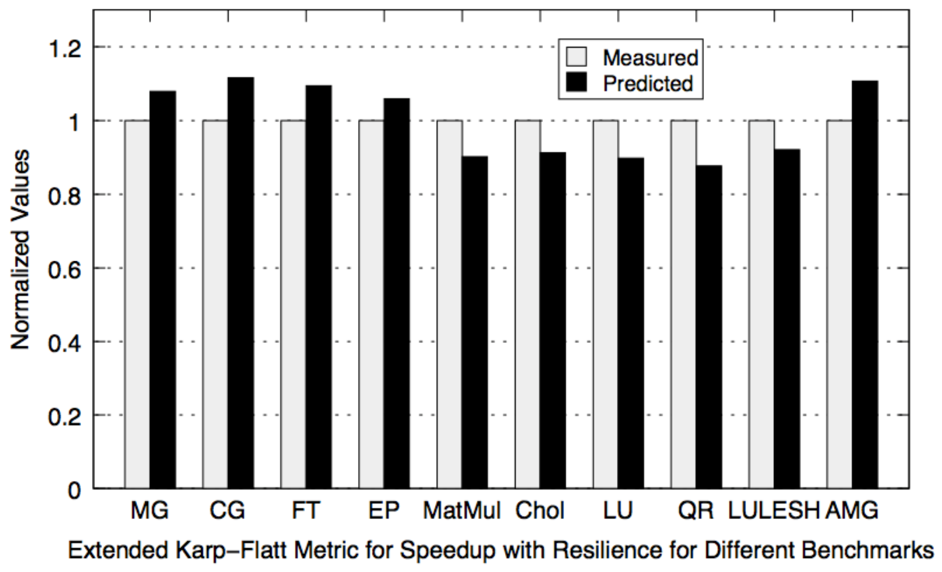
Benchmark	Suite	Description and Test Case	Domain	LOC	Parallelized in	Percentage of Parallelized Code
MG	NPB	Solve a discrete Poisson equation using multigrid method (Class B).	discrete mathematics	2568	OpenMP/MPI	73.0%
CG	NPB	Estimate eigenvalue of a sparse matrix with conjugate gradient method (Class B).	numerical linear algebra	1864	OpenMP/MPI	93.3%
FT	NPB	Solve a partial differential equation using fast Fourier transform (Class B).	numerical linear algebra	2034	OpenMP/MPI	58.7%
EP	NPB	Generate Gaussian random variates using Marsaglia polar method (Class B).	probability theory and statistics	359	OpenMP/MPI	94.7%
MatMul	Self-programed	Matrix multiplication on two 10k×10k global matrices, saving into a third one.	numerical linear algebra	1532	OpenMP/MPI /Pthreads	99.2%
Chol	FT-ScaLAPACK	Cholesky factorization on a 10k×10k global matrix to solve a linear system.	numerical linear algebra	2182	MPI	92.7%
LU	FT-ScaLAPACK	LU factorization on a 10k×10k global matrix to solve a linear system.	numerical linear algebra	2892	MPI	61.6%
QR	FT-ScaLAPACK	QR factorization on a 10k×10k global matrix to solve a linear system.	numerical linear algebra	3371	MPI	76.5%
LULESH	DARPA UHPC	Approximate hydrodynamics equations using 512 volumetric elements on a mesh.	hydrodynamics	6014	OpenMP/MPI	14.6%
AMG	CORAL	An algebraic multigrid solver for linear systems on a 4×4×6 unstructured grid.	numerical linear algebra	3098	OpenMP/MPI	65.1%

Table II. Hardware Configuration for All Experiments.

Cluster	HPCL	ARC
System Size (# of Nodes)	8	108
Processor	2×Quad-core AMD Opteron 2380	2×8-core AMD Opteron 6128
CPU Freq.	0.8, 1.3, 1.8, 2.5 GHz	0.8, 1.0, 1.2, 1.5, 2.0 GHz
CPU Voltage (Undervolting)	1.300, 1.100, 1.025, 0.850 V ($V_h/V_l/V_{safe\_min}/V_{th}$)	N/A
Memory	8 GB RAM	32 GB RAM
Cache	128 KB L1, 512 KB L2, 6 MB L3	128 KB L1, 512 KB L2, 12 MB L3
Network	1 GB/s Ethernet	40 GB/s InfiniBand
OS	CentOS 6.2, 64-bit Linux kernel 2.6.32	CentOS 5.7, 64-bit Linux kernel 2.6.32
Power Meter	PowerPack	Watts up? PRO

Results on Validation of Modeling Accuracy

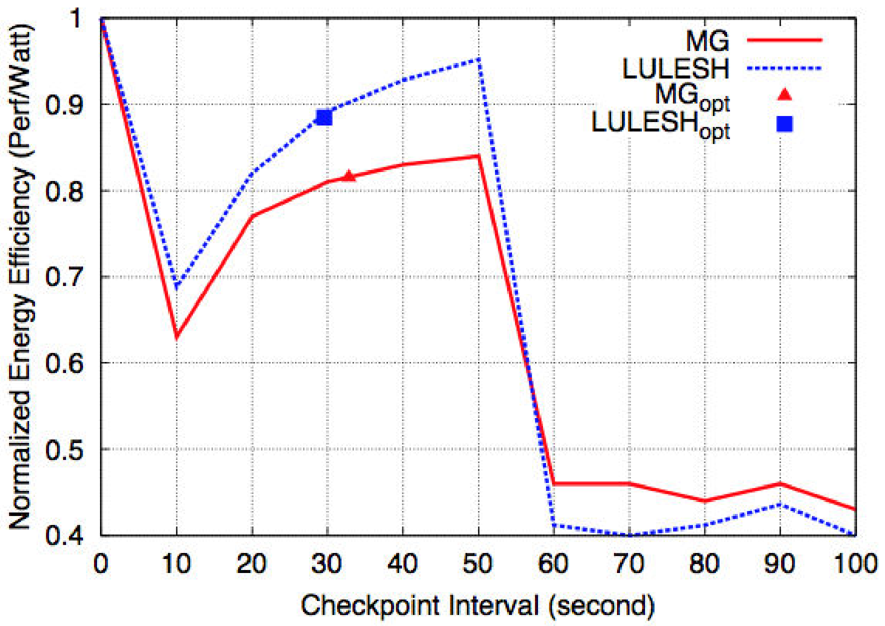


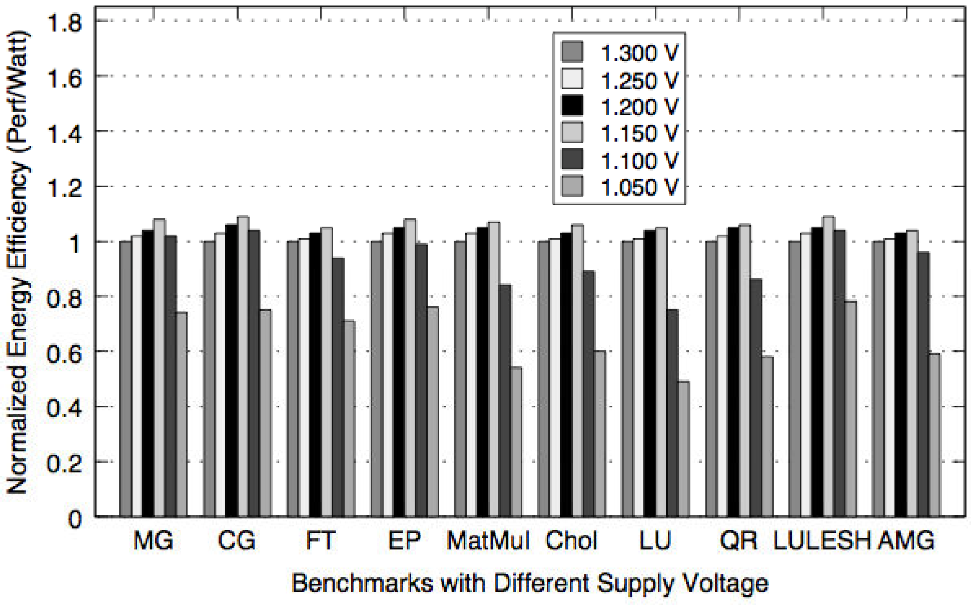


7.7%

9.4%

Results on Effects on EE from HPC Parameters

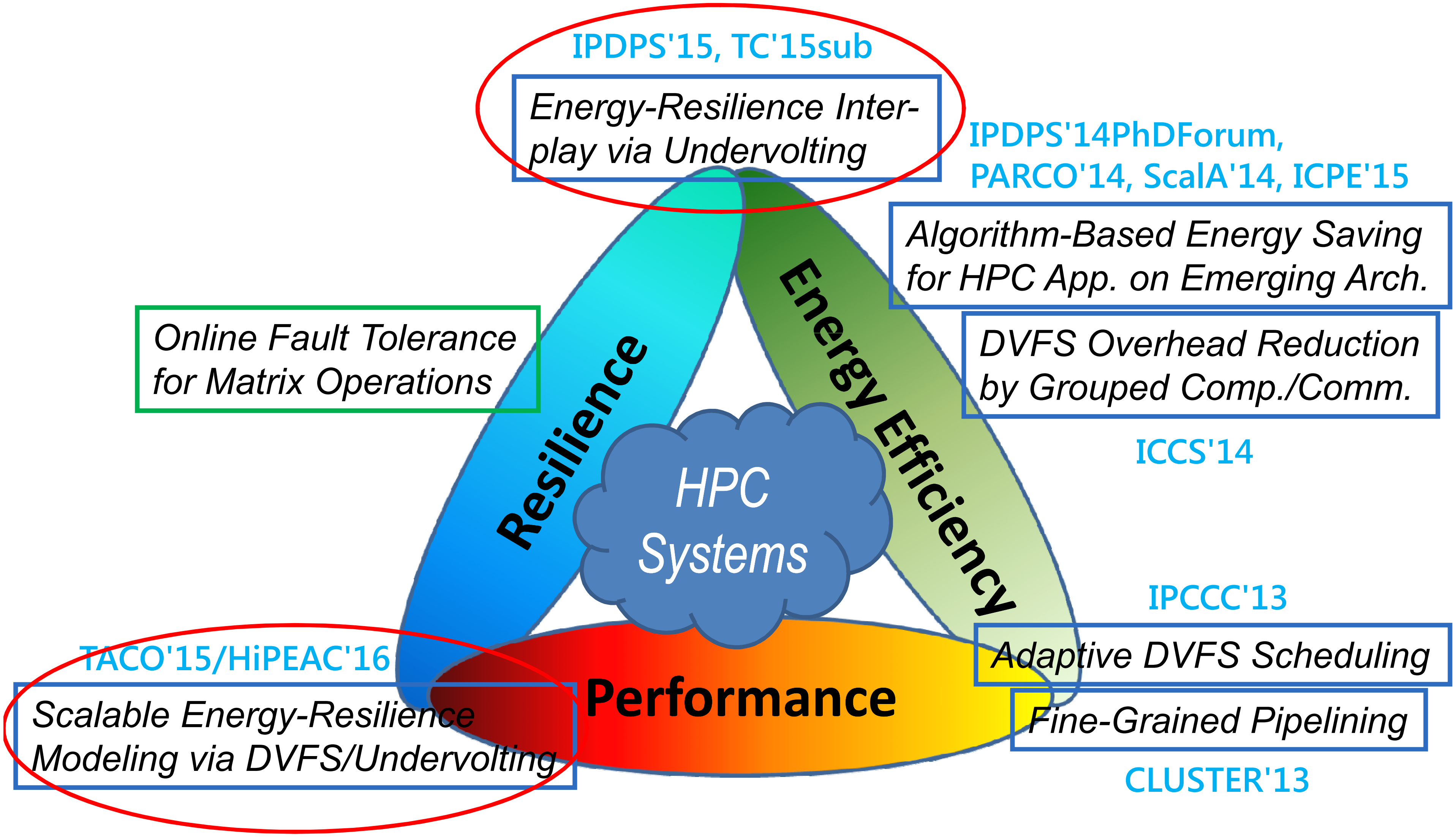




We injected an error at 55th s (T~=100s)

Optimal voltage that balances E+ and E-

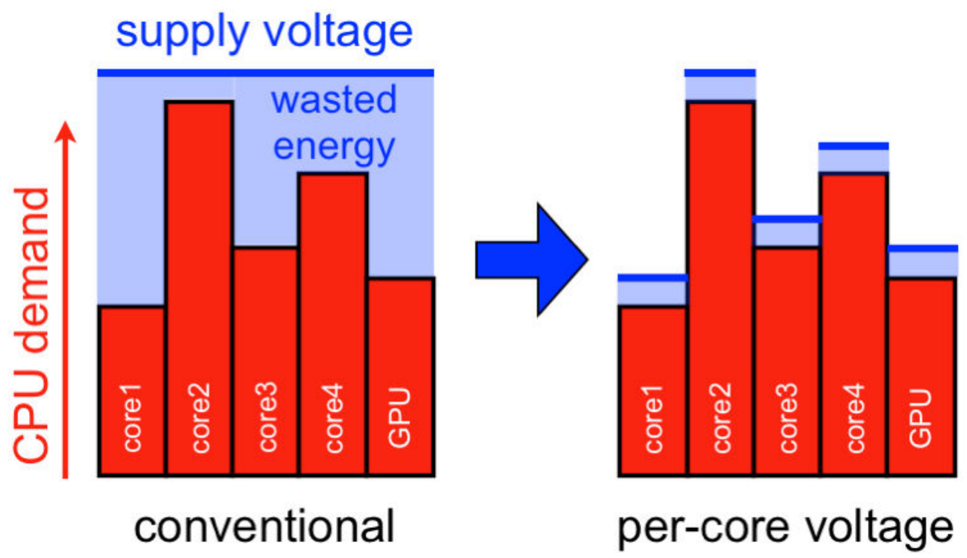
Research in PhD


IPDPS'15, TC'15sub
Energy-Resilience Inter-play via Undervolting
IPDPS'14PhDForum, PARCO'14, ScalA'14, ICPE'15
Algorithm-Based Energy Saving for HPC App. on Emerging Arch.
DVFS Overhead Reduction by Grouped Comp./Comm.
ICCS'14
Online Fault Tolerance for Matrix Operations
Resilience
Energy Efficiency
HPC Systems
Performance
IPCCC'13
Adaptive DVFS Scheduling
Fine-Grained Pipelining
CLUSTER'13
TACO'15/HiPEAC'16
Scalable Energy-Resilience Modeling via DVFS/Undervolting

Solving Power and Energy Concerns in HPC

- Dynamic Voltage and Frequency Scaling (DVFS)
  - CMOS-based components(CPU/GPU/mem.) *dominant*
  - Strategically switch processors to low-power states when the *peak* processor performance is *unnecessary*
  - voltage/frequency ↓ → power ↓ → energy efficiency

$P \propto fV^2$



Beyond DVFS: Undervolting w/ Fixed Frequency

- Basics of Energy Saving DVFS Solutions
  - Power consumption of these components $P \propto fV^2$
  - *Supply voltage* has a positive correlation with (not strictly proportional/linear to) *operating frequency*

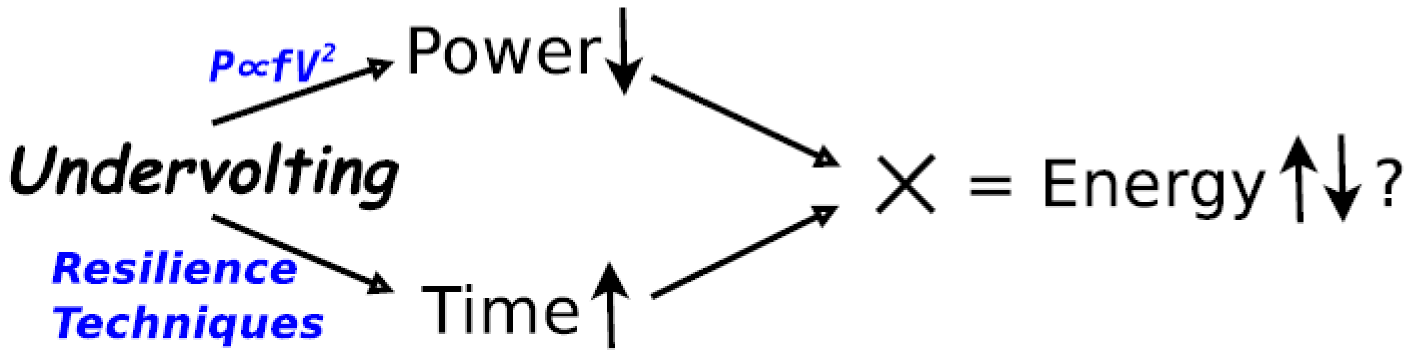
- Limitations of Existing Solutions
  - Most DVFS approaches are *frequency-directed* for slack

- Our Contributions
  - *Undervolting*: For a given frequency, hardware can be supplied w/ a voltage lower than the original *paired* one
    - Hardware throughput is preserved due to *fixed* frequency
    - *Uniformly* applied to both *slack* and *non-slack* of HPC runs

Challenges

- Caused Increasing Failure Rates
  - Both *hard* & *soft* errors may occur during undervolting
  - Energy savings may be *offset*: error detection/recovery
  - Theoretical validation holds or not? Any conditions?
- Hardware Support Constraints
  - Architectural solutions to support reliable undervolting
    - Simulation-based: Intel's Wilkerson *et al.* [ISCA'08, ISCA'11]
    - Real-machine: Bacha *et al.* [ISCA'13] → firmware/software + pre-prod. multicore processor + only studied ECC mem. errors
    - Large-scale HPC systems? → portability + scalability

Our Approach



- Key Points
  - Energy saving undervolting for *HPC systems* by leveraging mainstream resilience techniques
  - *Positive* effects: power reduction
  - *Negative* effects: error detection/recovery overhead

Our Approach (Cont.)

- Goals (A Recap)
  - *Target*: HPC systems consisting of a number of nodes connected by networks based on msg-passing comm.
  - *Q1*: *Trade-off* between *power savings* & *performance loss* at the increased failure rates from undervolting
  - *Q2*: *Theoretically*&*empirically* study if undervolting with a fixed frequency is able to saving energy w/ resilience
  - *Q3*: If it is feasible to save *more* energy than state-of-the-art *frequency-directed* DVFS solutions w/ resilience

Trade-off for Undervolting

- Errors/Faults/Failures in a Computing System
  - Hard errors: *permanent*, e.g., node crash, system abort
  - Soft errors: *transient*, e.g., memory bit-flips, logic errors
  - Concerned error types: hard + soft (↑ by undervolting)
- Error Detection and Recovery
  - We employ resilience techniques that can do both
  - Different techniques have different overhead
  - Trade-off between overhead and generality

Failure Rate Modeling

- Assumption
  - Failures of combinational logic circuits follow a Poisson distribution, determined by frequency and voltage

$$\lambda(f, V_{dd}) = \lambda(f) = \lambda_0 \; e^{\frac{d(f_{max}-f)}{f_{max}-f_{min}}}$$

  - Relationship between frequency and voltage

$$f = \varphi(V_{dd}, V_{th}) = \beta \; \frac{(V_{dd}-V_{th})^2}{V_{dd}}$$

  - By substitution, we get

$$\lambda(f, V_{dd}) = \lambda(V_{dd}) = \lambda_0 \; e^{\frac{d(f_{max}-\beta(V_{dd}-2V_{th}+\frac{V_{th}^2}{V_{dd}}))}{f_{max}-f_{min}}}$$

Failure Rate Modeling (Cont.)

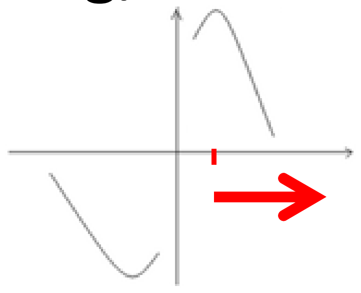
$$\boxed{\lambda(f, V_{dd}) = \lambda(V_{dd}) = \lambda_0 \; e^{\frac{d(f_{max} - \beta(V_{dd} - 2V_{th} + \frac{V_{th}^2}{V_{dd}}))}{f_{max} - f_{min}}}}$$

- We manage to present the average failure rate as a function of supply voltage V_{dd} only
- Let the first derivative of the above equation = 0

$$\frac{\partial \lambda}{\partial V_{dd}} = \lambda_0 \; e^{\mu(V_{dd})} \; \frac{-d\beta(1 - (V_{th}/V_{dd})^2)}{f_{max} - f_{min}} = 0$$

$$\Rightarrow V_{dd} = \pm V_{th}$$

- We thus obtain the monotonic increasing/decreasing domain of $\lambda(V_{dd})$ with regard to V_{dd}



An Example

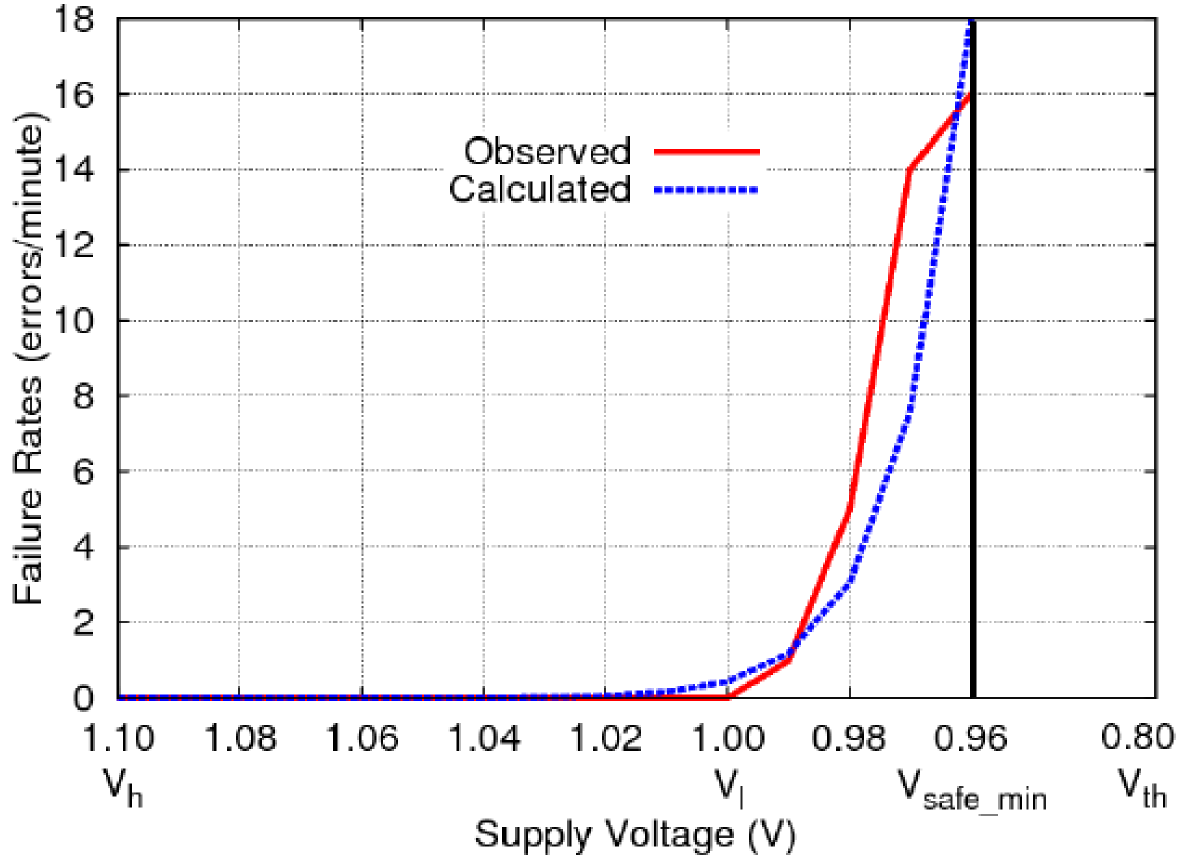
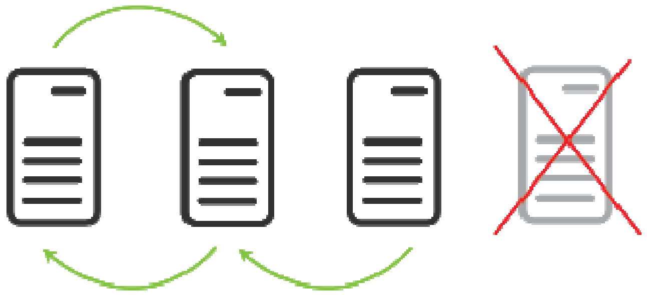
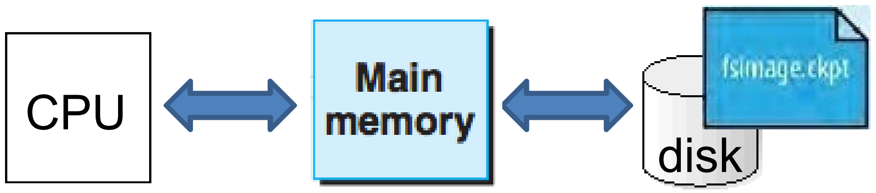


Figure 2. Observed and Calculated Failure Rates as a Function of Supply Voltage for a Pre-Production Intel Itanium II 9560 8-Core Processors (V_h: max volt. paired with max freq., V_l: min voltage paired with min freq., V_{safe_min}: min volt. for pre-production processors, V_{th}: threshold volt.).

Fault Tolerance in HPC

- Resilience Techniques
  - Disk-Based Checkpoint/Restart (DBCR)
    - Checkpoints saved in disk, high I/O overhead
  - Diskless Checkpointing (DC)
    - Checkpoints saved in memory, trade-off (mem. + generality)
  - Triple Modular Redundancy (TMR)
    - Detect and correct one erroneous run within three runs
  - Algorithm-Based Fault Tolerance (ABFT)
    - Leverage algorithmic characteristics to correct errors online





Fault Tolerance in HPC (Cont.)

- Examples (C/R and ABFT only)

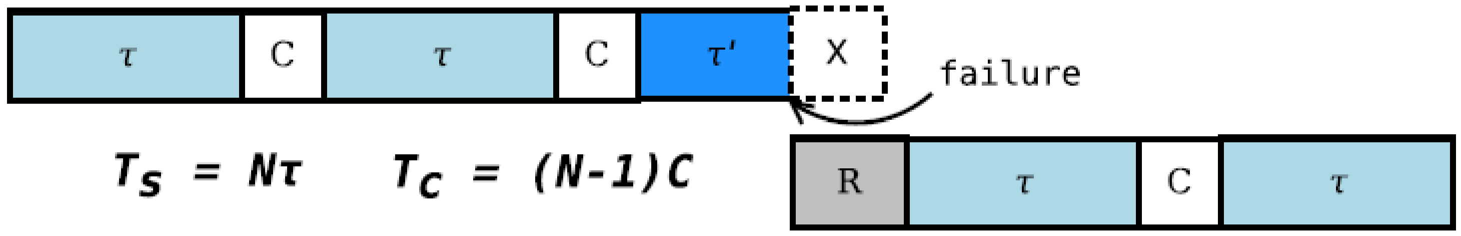


Figure 3. Checkpoint/Restart Execution Model for a Single Process.

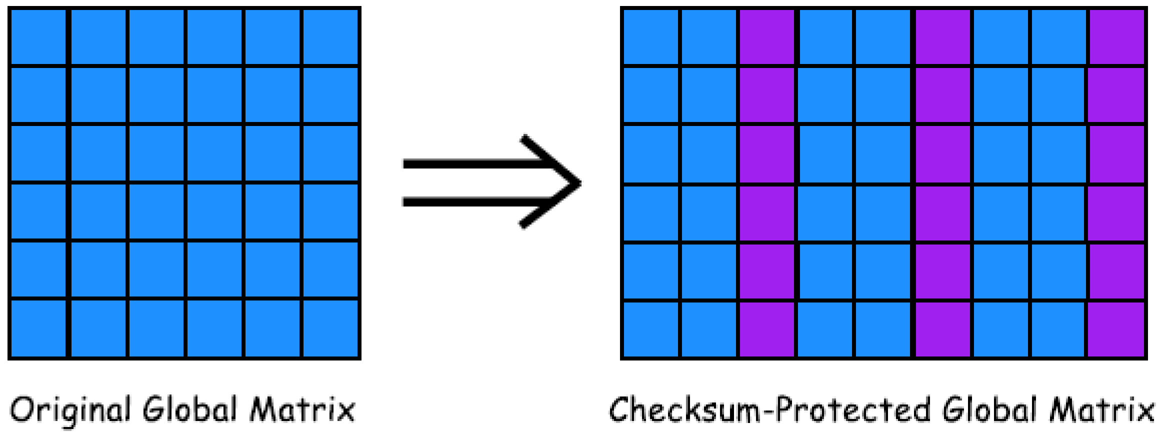


Figure 4. Algorithm-Based Fault Tolerance Model for Matrix Operations.

Performance Modeling

- Checkpoint/Restart (C/R) for General Applications
  - Given a failure rate, there exists an *optimal* checkpoint interval that *minimizes* the total C/R overhead
    - At *nominal* voltage, $\lambda(V_{dd})$ is small (close to zero)

$$\tau_{opt} = \sqrt{2C(\frac{1}{\lambda} + R)} \quad \text{for } \tau + C \ll \frac{1}{\lambda}$$

    - At *further reduced* voltage, $\lambda(V_{dd})$ is raised significantly

$$\tau_{opt} = \begin{cases} \sqrt{\frac{2C}{\lambda}} - C & \text{for } C < \frac{1}{2\lambda} \\ \frac{1}{\lambda} & \text{for } C \geq \frac{1}{2\lambda} \end{cases}$$

  - Performance breakdown:

$$T_{cr} = T_s + (\frac{T_s}{\tau} - 1)C + \phi(\tau + C)n + Rn$$

Power Consumption Modeling

- With Undervolting and Resilience Techniques
  - Use C/R as an example for model building
  - Study homogeneous HPC systems w/o accelerators
  - For a cluster of compute nodes, a nodal power model

$$\begin{aligned} P &= P_{dynamic}^{CPU} + P_{leakage}^{CPU} + P_{leakage}^{other} \\ &= AC'fV_{dd}^2 + I_{sub}V_{dd} + I'_{sub}V'_{dd} \end{aligned}$$

  - Consider three power patterns for a node doing C/R

$$\begin{cases} P_h &= AC'f_hV_h^2 + I_{sub}V_h + P_c \\ P_m &= AC'f_hV_{safe\_min}^2 + I_{sub}V_{safe\_min} + P_c \\ P_l &= AC'f_lV_{safe\_min}^2 + I_{sub}V_{safe\_min} + P_c \end{cases}$$

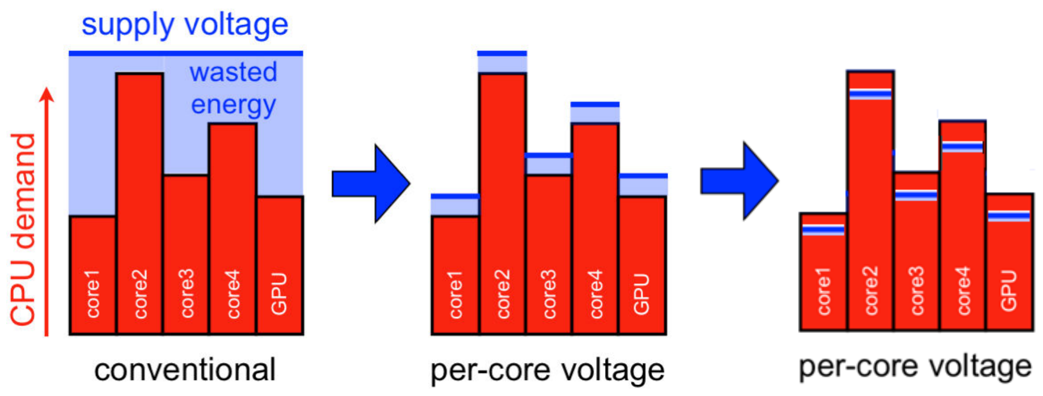
Energy Consumption Modeling

- With Undervolting and Resilience Techniques
  - For an HPC run, we have three variants
    - A *baseline* run with *nominal* frequency and voltage
    - A run with undervolting in the *absence* of failures
    - A run with undervolting in the *presence* of failures
  - Integrating three power patterns, energy cost models

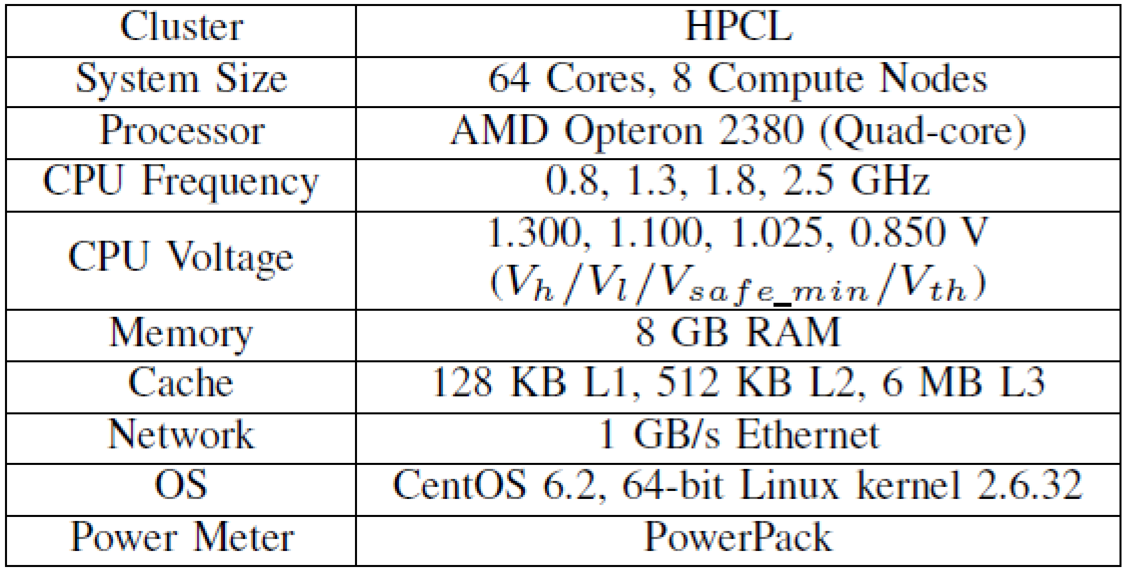
$$
\begin{cases}
E_{base} = P_h T_s \\
E_{uv}^{\overline{err}} = P_m T_s + P_l\left(\frac{T_s}{\tau} - 1\right)C \\
E_{uv}^{err} = P_m(T_s + \phi\tau n) + P_l\left(\left(\frac{T_s-\tau}{\tau} + \phi n\right)C + Rn\right)
\end{cases}
$$

Energy Savings over State-of-the-art DVFS

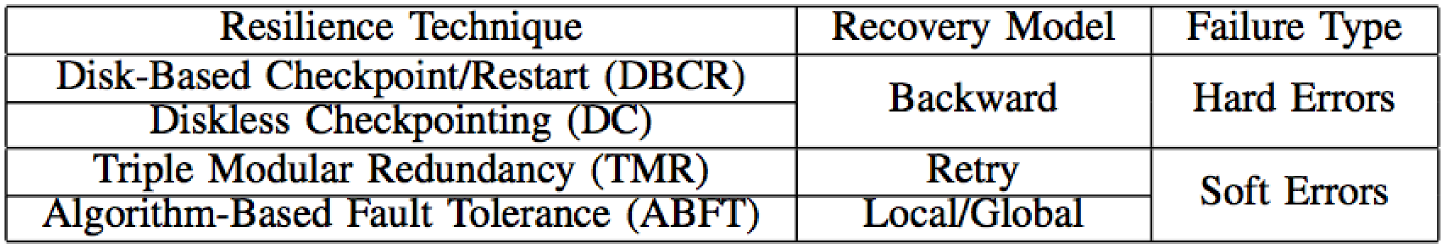
- Frequency-directed DVFS Approaches
  - Processors equipped with a range of frequencies
  - Predict and apply *appropriate* freq./volt. during slack
    - Accurate workload prediction, frequency approximation, etc.
    - Employ a state-of-the-art DVFS solution Adagio [ICS'09]
  - Can we *further* save energy *beyond* DVFS?
    - Continue undervolting further per selected appropriate F/V
    - Also leverage resilience techniques to guarantee correctness



Experimental Setup



Cluster	HPCL
System Size	64 Cores, 8 Compute Nodes
Processor	AMD Opteron 2380 (Quad-core)
CPU Frequency	0.8, 1.3, 1.8, 2.5 GHz
CPU Voltage	1.300, 1.100, 1.025, 0.850 V (V_h/V_l/V_{safe_min}/V_{th})
Memory	8 GB RAM
Cache	128 KB L1, 512 KB L2, 6 MB L3
Network	1 GB/s Ethernet
OS	CentOS 6.2, 64-bit Linux kernel 2.6.32
Power Meter	PowerPack



Resilience Technique	Recovery Model	Failure Type
Disk-Based Checkpoint/Restart (DBCR)	Backward	Hard Errors
Diskless Checkpointing (DC)		
Triple Modular Redundancy (TMR)	Retry	Soft Errors
Algorithm-Based Fault Tolerance (ABFT)	Local/Global	

Benchmarks

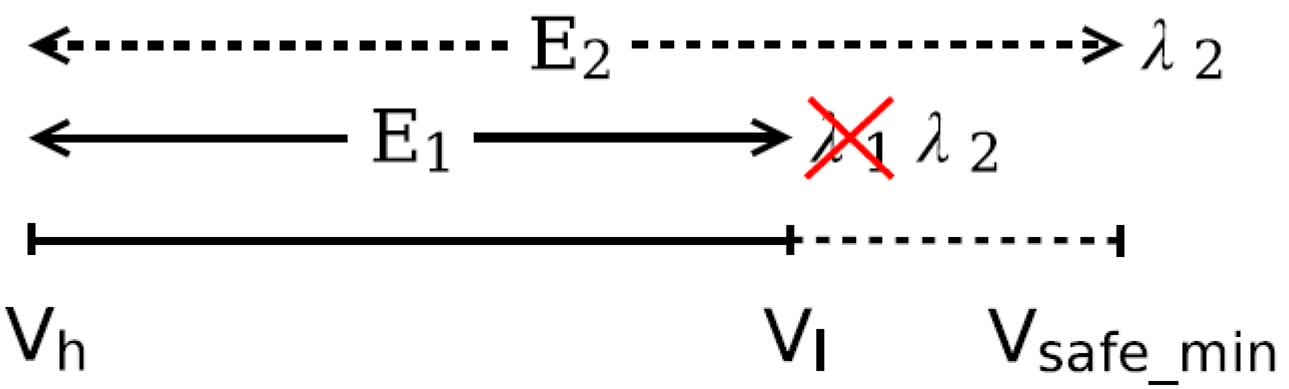
- NASA-concerned HPC Benchmarks
  - MG, CG, and FT from the NPB benchmark suite
- DOE-concerned HPC Benchmarks
  - LULESH
  - AMG
- Widely-used Numerical Linear Algebra Libraries
  - Matrix multiplication
  - Cholesky factorization
  - LU factorization
  - QR factorization

Implementation

- Undervolting Production Processors
  - Modify the northbridge/CPU FID and VID control reg.
    - Voltage reg. values are altered via *Model Specific Register*
  - This approach needs careful detection of the upper and lower bounds of supply voltage of the processor
    - Hardware-damaging issues may arise
  - Different from the undervolting approach in [ISCA'13]
    - Software/firmware control
    - Pre-production processor is required
    - ECC memory support is necessary

Implementation (Cont.)

- Error Injection
  - Minimum voltage we can undervolt to is V_l
    - No errors will be observed due to *close-to-zero* failure rates
  - Based on the failure rates between V_l and V_{safe_min} we inject errors to emulate the erroneous scenarios
    - Hard errors: manually kill an arbitrary MPI process
    - Soft errors: modify values of matrix elements randomly



Implementation (Cont.)

- Energy Cost Estimation
  - Energy consumption cannot be experimentally measured when undervolting to V_{safe_min}
  - Apply the previous *emulated scaling* method to estimate the energy costs at V_{safe_min}

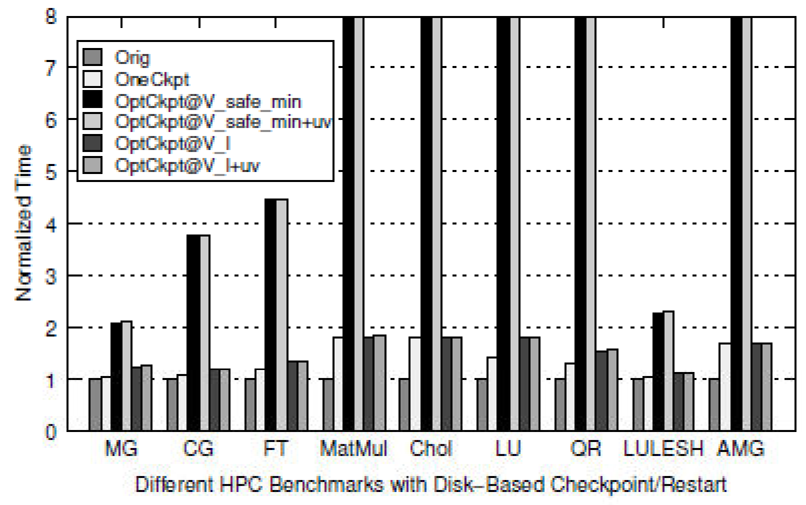
$$\begin{cases} P_h &= AC' f_h V_h^2 + I_{sub} V_h + P_c \\ P_m &= AC' f_h V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \\ P_l &= AC' f_l V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \end{cases}$$

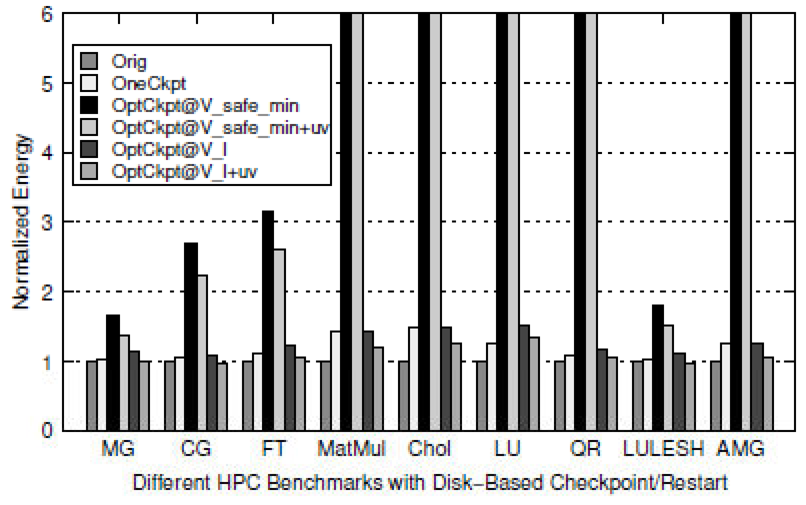
$$\begin{cases} E_{base} = P_h T_s \\ E_{uv}^{\overline{err}} &= P_m T_s + P_l (\frac{T_s}{\tau} - 1) C \\ E_{uv}^{err} &= P_m (T_s + \phi \tau n) + P_l \left((\frac{T_s - \tau}{\tau} + \phi n) C + Rn \right) \end{cases}$$

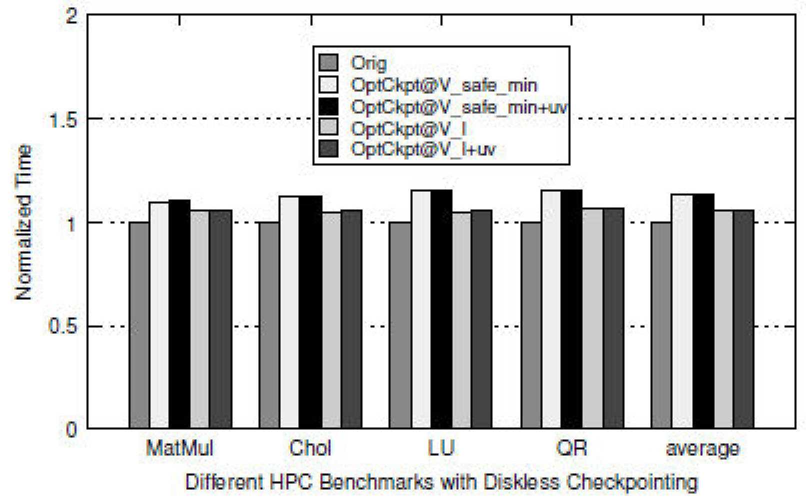
Evaluation

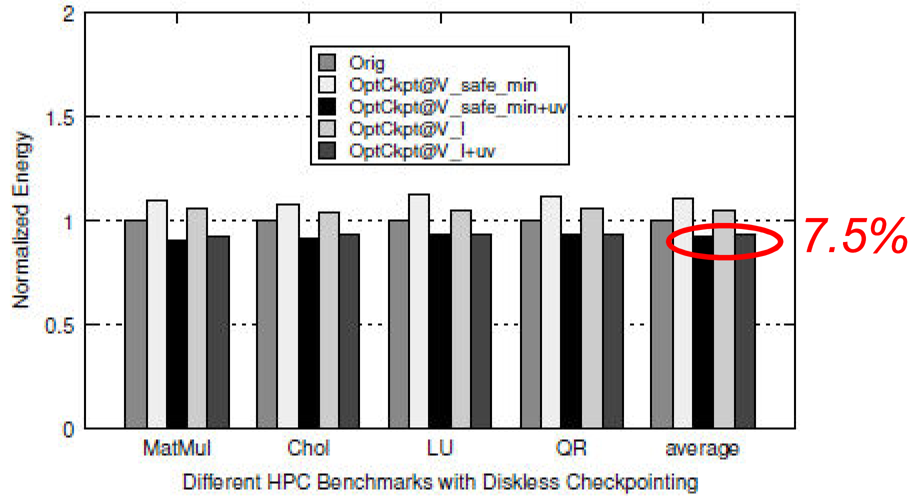
- Test Scenarios
  - Checkpoint-kind resilience techniques (DBCR/DC)
    - OneCkpt: Checkpoint/restart is only performed *once*
    - OptCkpt@Vx: Checkpoint/restart is performed with the *optimal* checkpoint interval at Vx
    - OptCkpt@Vx + uv: OptCkpt@Vx + undervolting
  - Non-checkpoint resilience techniques (TMR/ABFT)
    - By nature, fault tolerant actions are performed at a *fixed* frequency, not affected by failure rates
    - Simply apply undervolting at different voltage levels
  - Energy efficiency over Adagio
    - Adagio: predicted frequency + paired voltage
    - Adagio + uv: predicted frequency + undervolting

Experimental Results (DBCR vs. DC)

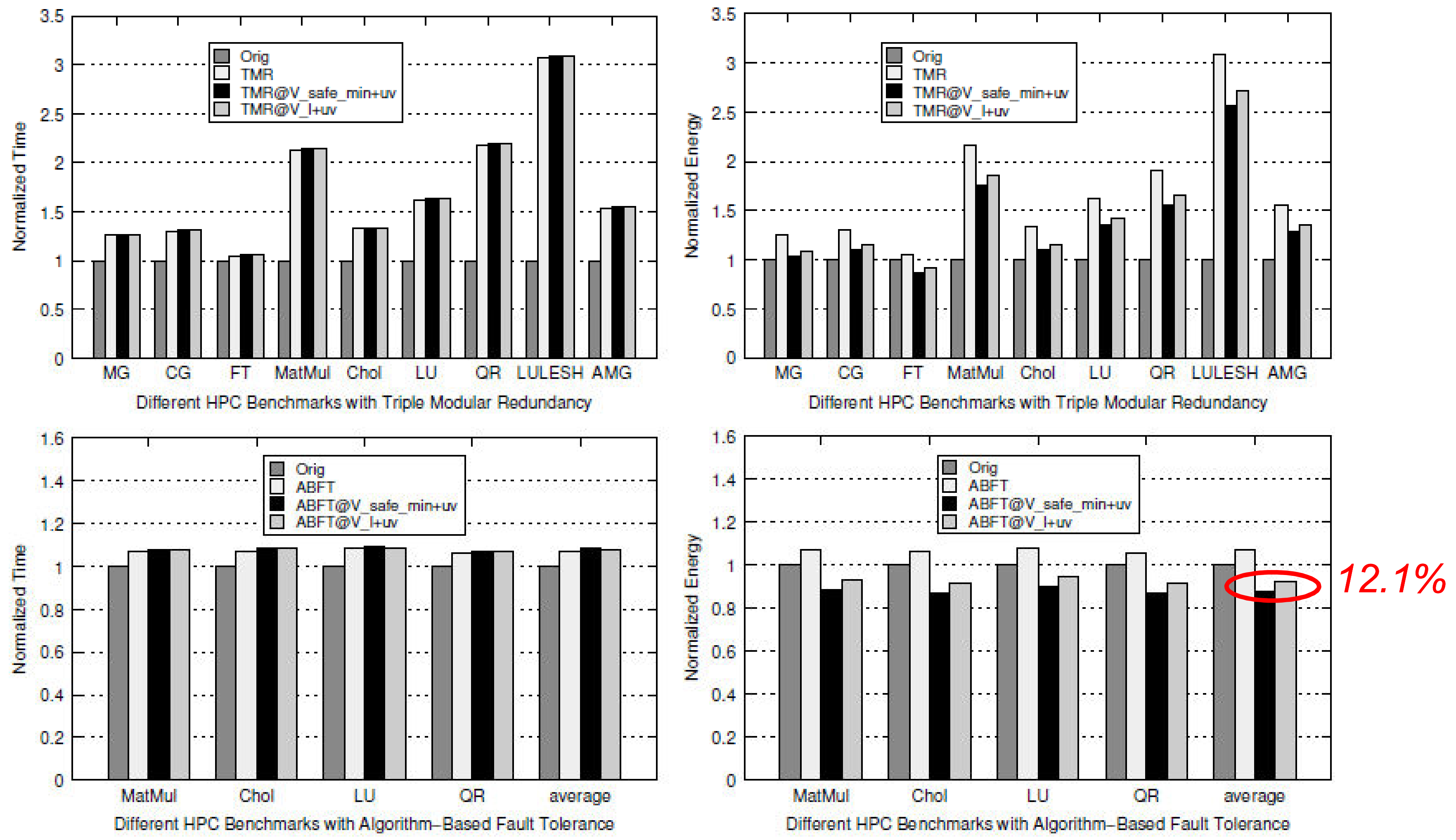




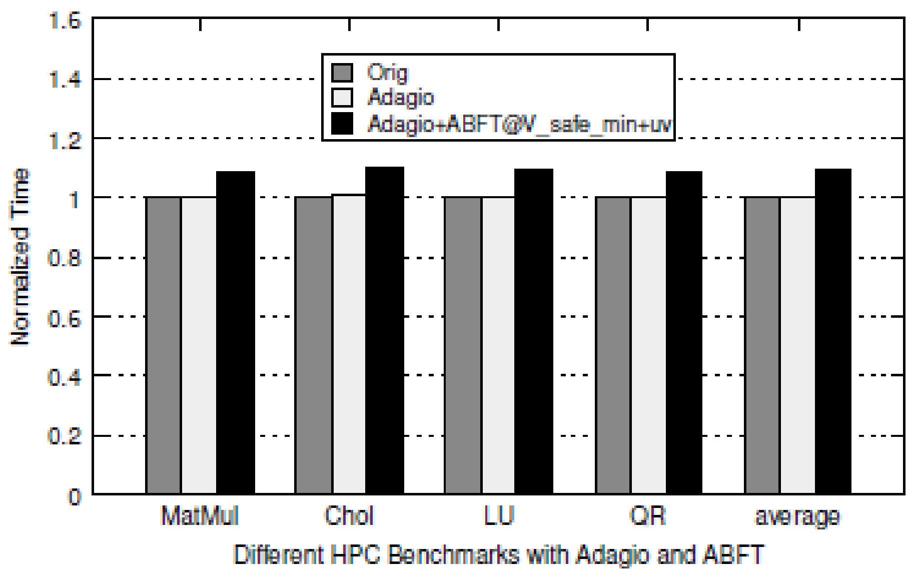


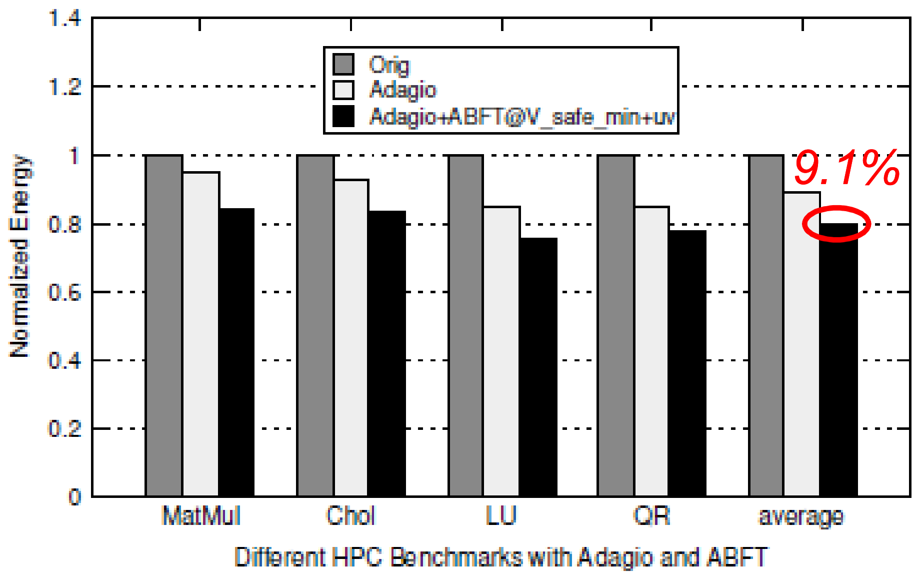


Experimental Results (TMR vs. ABFT)

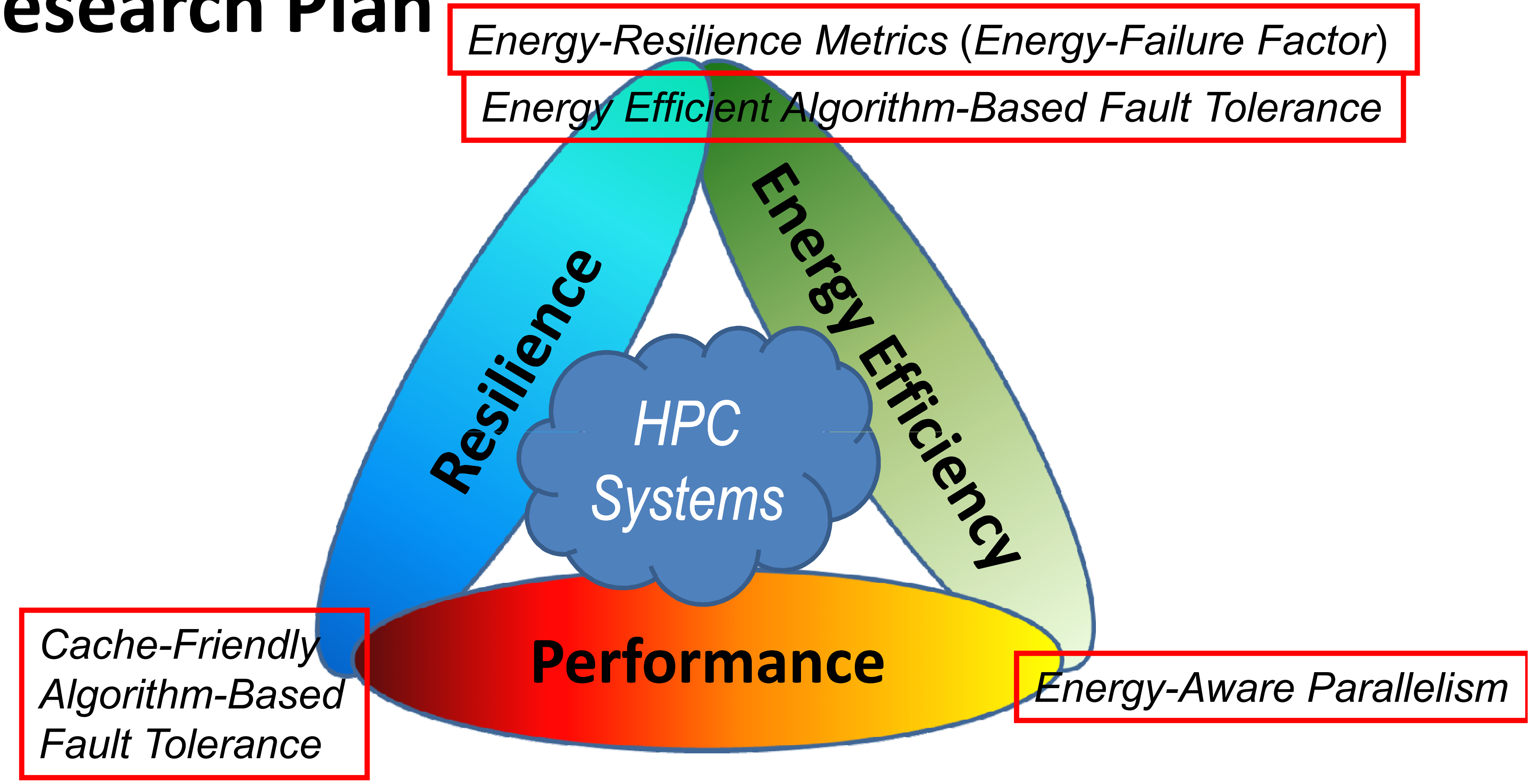


Experimental Results (Adagio + Undervolting)





Research Plan

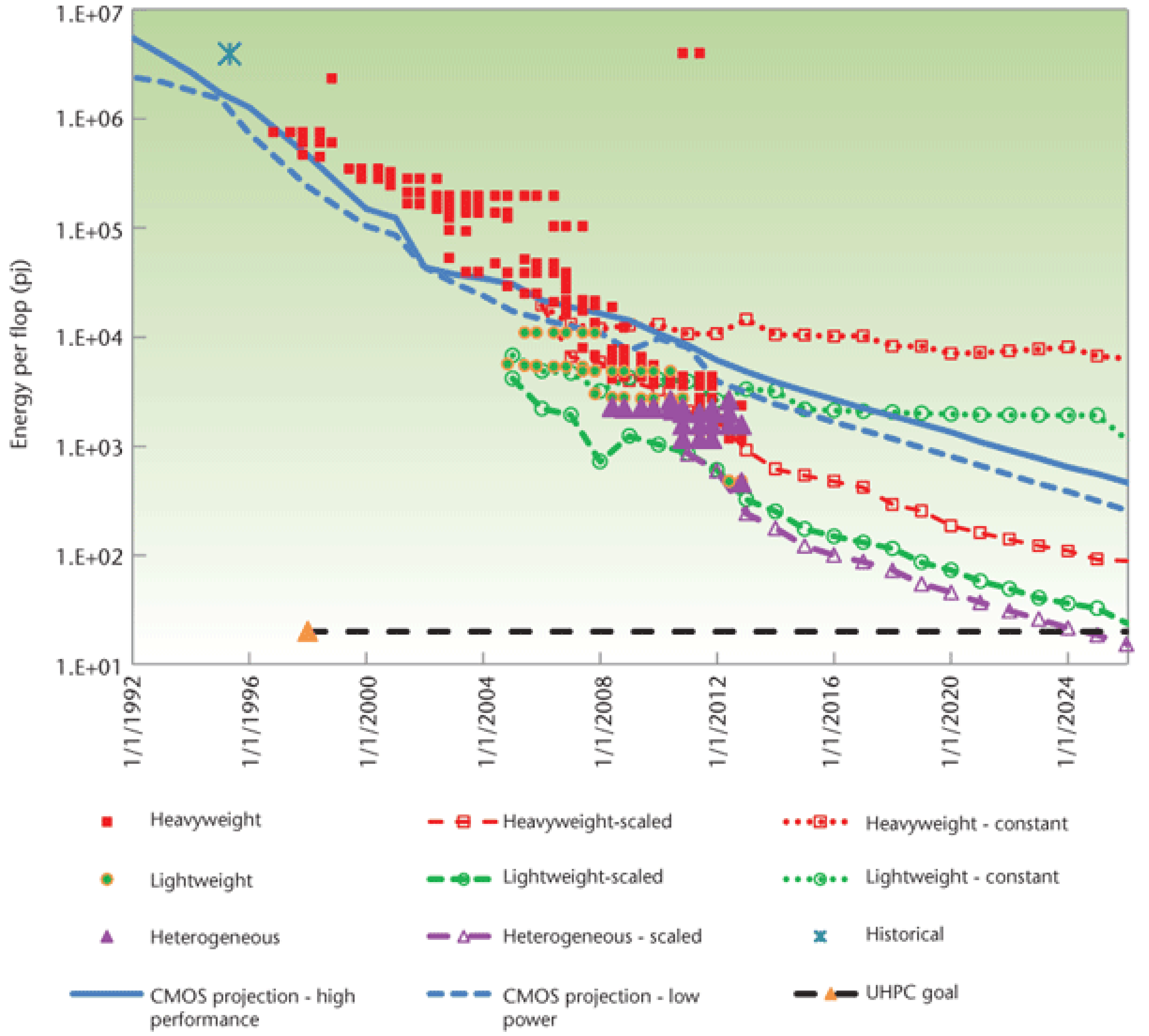

Energy-Resilience Metrics (Energy-Failure Factor)
Energy Efficient Algorithm-Based Fault Tolerance
Resilience
Energy Efficiency
HPC Systems
Performance
Cache-Friendly Algorithm-Based Fault Tolerance
Energy-Aware Parallelism

- Entangled effects among three critical challenges in HPC
- Improving one or more w/o degrading the others (much)
- Balance/trade-off among the three is worth digging into

Publications in PhD (11 Peer-review 1st Author)

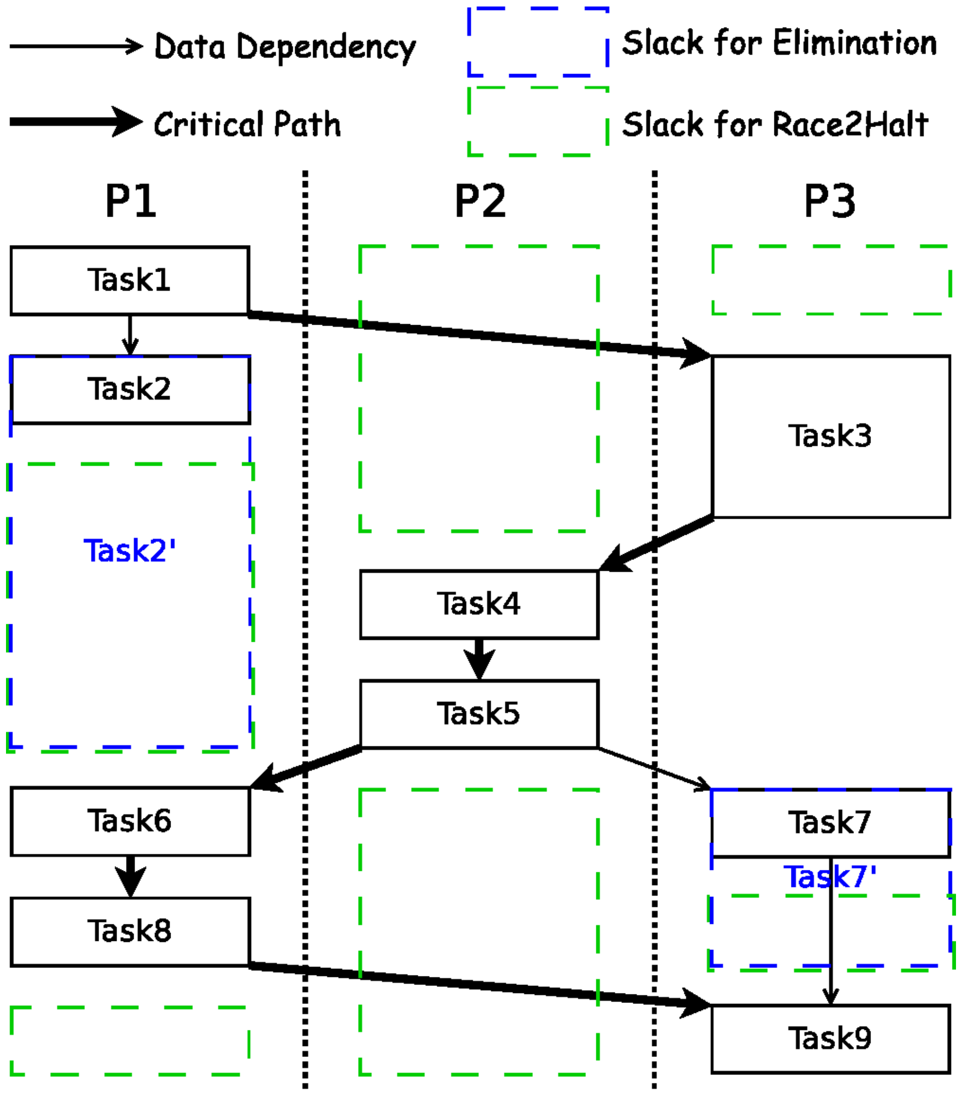
TACO'15/ HiPEAC'16	**Li Tan**, Zizhong Chen, and Shuaiwen Leon Song. "Scalable Energy Efficiency with Resilience for High Performance Computing Systems: A Quantitative Methodology", ACM Transactions on Architecture and Code Optimization (TACO), Vol. 12, No. 4, Article 35, Oct. 2015.&International Conference on High-Performance Embedded Architectures and Compilers (HiPEAC) 2016.
TC'15sub	**Li Tan**, Shuaiwen Leon Song, PanruoWu, Zizhong Chen, Rong Ge, and Darren J. Kerbyson. "Investigating the Interplay between Energy Efficiency and Resilience in High Performance Computing", submitted to IEEE Transactions on Computers, 2015.
IPDPS'15	**Li Tan**, Shuaiwen Leon Song, Panruo Wu, Zizhong Chen, Rong Ge, and Darren J. Kerbyson. "Investigating the Interplay between Energy Efficiency and Resilience in High Performance Computing", in Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 786-796, Hyderabad, India, May 2015.
PARCO'14	**Li Tan**, Shashank Reddy Kothapalli, Longxiang Chen, Omar Hussaini, Ryan Bissiri, and Zizhong Chen. "A Survey of Power and Energy Efficient Techniques for High Performance Numerical Linear Algebra Operations", Journal of Parallel Computing (PARCO), Vol. 40, No. 10, pp. 559-573, Dec. 2014.
CLUSTER'13	**Li Tan**, Longxiang Chen, Zizhong Chen, Ziliang Zong, Rong Ge, and Dong Li. "Improving Performance and Energy Efficiency of Matrix Multiplication via Pipeline Broadcast", in Proceedings of the 15th IEEE International Conference on Cluster Computing (CLUSTER), pp. 1-5, Indianapolis, Indiana, USA, Sept. 2013.
CGO'13	**Li Tan**, Min Feng, and Rajiv Gupta. "Lightweight Fault Detection in Parallelized Programs", in Proceedings of the 11th ACM/IEEE International Symposium on Code Generation and Optimization (CGO), pp. 1-11, Shenzhen, China, Feb. 2013.
ICCS'14	**Li Tan**, Longxiang Chen, Zizhong Chen, Ziliang Zong, Rong Ge, and Dong Li. HP-DAEMON: "High Performance Distributed Adaptive Energy-efficient Matrix-multiplicatiON", in Proceedings of the 14th International Conference on Computational Science (ICCS), pp. 599-613, Cairns, Queensland, Australia, June 2014.
IPCCC'13	**Li Tan**, Zizhong Chen, Ziliang Zong, Rong Ge, and Dong Li. "A2E: Adaptively Aggressive Energy Efficient DVFS Scheduling for Data Intensive Applications", in Proceedings of the 32nd IEEE International Performance Computing and Communications Conference (IPCCC), pp. 1-10, San Diego, California, USA, Dec. 2013.
ICPE'15	**Li Tan** and Zizhong Chen. "Slow Down or Halt: Saving the Optimal Energy for Scalable HPC Systems", in Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE), pp. 241-244, Austin, Texas, USA, Feb. 2015.
ScalA'14	**Li Tan** and Zizhong Chen. "TX: Algorithmic Energy Saving for Distributed Dense Matrix Factorizations", in Proceedings of the 5th International Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), pp. 23-30, New Orleans, Louisiana, USA, Nov. 2014.
IPDPS'14 PhDForum	**Li Tan** and Zizhong Chen. "Optimizing Energy Efficiency for Distributed Dense Matrix Factorizations via Utilizing Algorithmic Characteristics", in PhD Forum of the 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Phoenix, Arizona, USA, May 2014.

Backup Slides



Source: Peter Kogge and John Shalf, *Exascale Computing Trends: Adjusting to the "New Normal"' for Computer Architecture*, IEEE Computing in Science & Engineering, Vol. 15, No. 6, pp. 16-26, 2013

Two Classic Energy Saving DVFS Solutions



Critical Path Aware Slack Reclamation

Race-to-halt/idle

Performance Modeling (Cont.)

- Algorithm-Based Fault Tolerance (ABFT) for Matrix Operations (Cholesky/LU/QR factorization)
  - In C/R, checkpoints are periodically *saved*
  - While in ABFT, checksums are periodically *updated*
    - Interval of updating checksums is *fixed* and not affected by the variation of failure rates → more cost-efficient
  - Performance breakdown:

$$T_{abft} = \frac{\mu C_f \mathbb{N}^3}{\mathbb{P}} t_f + \frac{\mu C_v \mathbb{N}^2}{\sqrt{\mathbb{P}}} t_v + \frac{C_m \mathbb{N}}{nb} t_m + T_d + T_l + T_c$$

- Performance modeling for other resilience techniques is conceptually similar

Energy Savings over DVFS (Cont.)

- Our Strategy (A Recap)
  - Use the frequency Adagio predicted for eliminating slack and *further lower* the voltage paired with it
  - We thus employ the following power patterns

$$\begin{cases} P_{Adagio}^{slack} & = AC' f_m V_m^2 + I_{sub} V_m + P_c \\ P_{Adagio}^{non-slack} & = P_h \\ P_{uv}^{slack} & = AC' f_m V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \\ P_{uv}^{non-slack} & = P_m \end{cases}$$

  - Theoretical energy savings over baseline runs

$$\begin{aligned} \Delta E & = E_{base} - E_{uv} \\ & = (P_h - P_m) T_s \oplus (P_h - P_{uv}^{slack}) T_{slack} - \\ & \left(P_m \phi \tau n + P_l \left(\left(\frac{T_s - \tau}{\tau} + \phi n \right) C + Rn \right) \right) \end{aligned}$$

Energy Saving Conditions over Baseline

- Given Platform-dependent Parameters (c_1, c_2, c_3, AC', I_{sub}, f, V, P_C)
  - Before Model Relaxation

$$T_s > \frac{c_3 C}{c_3\left(\frac{\lambda C}{\sqrt{2\lambda C}-\lambda C}+\frac{1}{2}\lambda C+R\lambda\right)-c_1+c_2(\sqrt{2\lambda C}-\lambda C)}$$

  - After Model Relaxation (N-$1 \approx N$)

$$R < \frac{c_1}{c_3\lambda}-\frac{c_2}{c_3}\left(\sqrt{\frac{2C}{\lambda}}-C\right)-\left(\frac{1}{\sqrt{2\lambda C}-\lambda C}+\frac{1}{2}\right)C$$

Implementation

- Failure Rate Calculation
  - *Limitation*: HPC production machines do not allow further voltage reduction beyond V_l
    - No real failures can be observed on our platform
  - Estimate failure rates between V_l and V_{safe_min}
  - Use the equation below to calculate the failure rates
    - High accuracy shown in the illustrated example

$$\boxed{\lambda(f, V_{dd}) = \lambda(V_{dd}) = \lambda_0 \; e^{\frac{d(f_{max} - \beta(V_{dd} - 2V_{th} + \frac{V_{th}^2}{V_{dd}}))}{f_{max} - f_{min}}}}$$

Implementation (Cont.)

- NB/CPU FID/VID control register format and formula

Bits (64 bits in total)	*Description*
63:32, 24:23, 21:19	Reserved
32:25	Northbridge Voltage ID, Read-Write
22	Northbridge Divisor ID, Read-Write
18:16	P-state ID, Read-Write
15:9	Core Voltage ID, Read-Write
8:6	Core Divisor ID, Read-Write
5:0	Core Frequency ID, Read-Write

- frequency = 100 MHz * (CPUFid + 10hex)/(2^CPUDid)

E.g.: 0x30002809 -> frequency = 100 * (9+16)/2^0 = 2.5 GHz

- voltage = 1.550 V – 0.0125 V * CPUVid

E.g.: 0x30002809 -> voltage = 1.550 - 0.0125 * 0010100b = 1.300 V

Conclusions and Future Work

- Undervolting can be beneficial to energy efficiency
  - At the cost of *increased* failure rates (detection + recovery)
  - *Lightweight* resilience techniques only incur *minor* perf. loss on error detection/recovery → energy savings
  - Undervolting is practical for future HPC systems
  - Feasible to save energy beyond classic DVFS solutions

- Ongoing Directions
  - Migrate undervolting to more types of hardware (eg, GPU)
  - Undervolting w/ fixed freq. VS. overclocking w/ fixed volt.
  - Is the other way around possible? → Improving resilience or performance at the cost of energy efficiency