Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

Investigating the Interplay between Energy Efficiency and Resilience in High Performance Computing

Li Tan[1], Shuaiwen Leon Song[2], Panruo Wu[1], Zizhong Chen[1], Rong Ge[3], Darren J. Kerbyson[2]

[1]*University of California, Riverside*
[2]*Pacific Northwest National Laboratory (PNNL)*
[3]*Marquette University*

29th IEEE International Parallel & Distributed Processing Symposium (IPDPS'15)
May 25-29, Hyderabad, INDIA

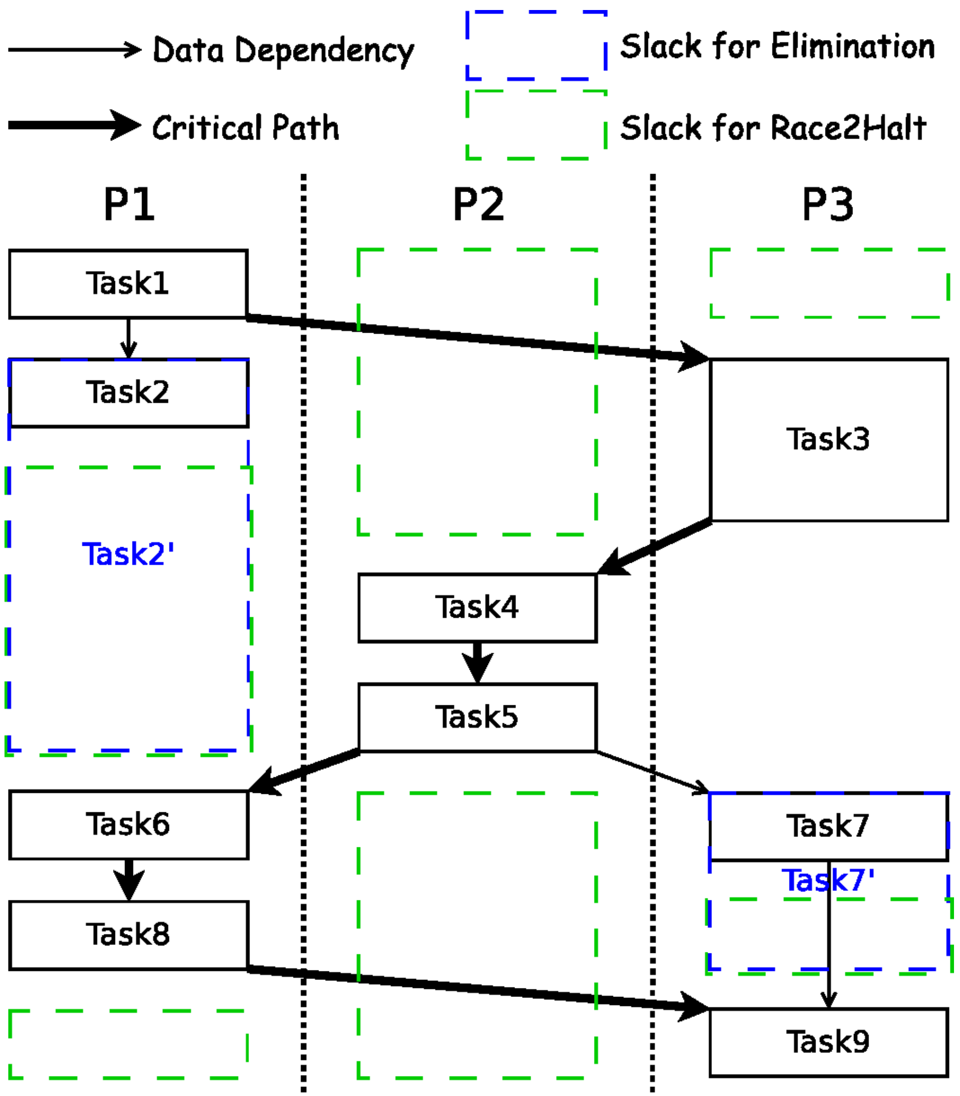
U.S. DEPARTMENT OF ENERGY

Power and Energy 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 *power-wall* by DOE for exascale (10^{18} FLOPS)
  - ❖ Overheat problems (aging/failures) and cooling costs

- 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

Two Classic Energy Saving DVFS Solutions



- *Critical Path Aware Slack Reclamation*
- *Race-to-halt/idle*

Beyond DVFS: Undervolting w/ Fixed Frequency

- Basics of the employed techniques

  - 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 techniques are *frequency-directed*
  - *Undervolting*: For a given frequency, hardware can be supplied w/ a voltage lower than the original *paired* one
    - Original part of the throughput can be preserved due to *fixed* frequency
    - *Uniformly* applied to both *slack* and *non-slack* of HPC runs (using the same DVFS techniques to find appropriate frequencies for each time interval, but with further reduced voltage).

Challenges

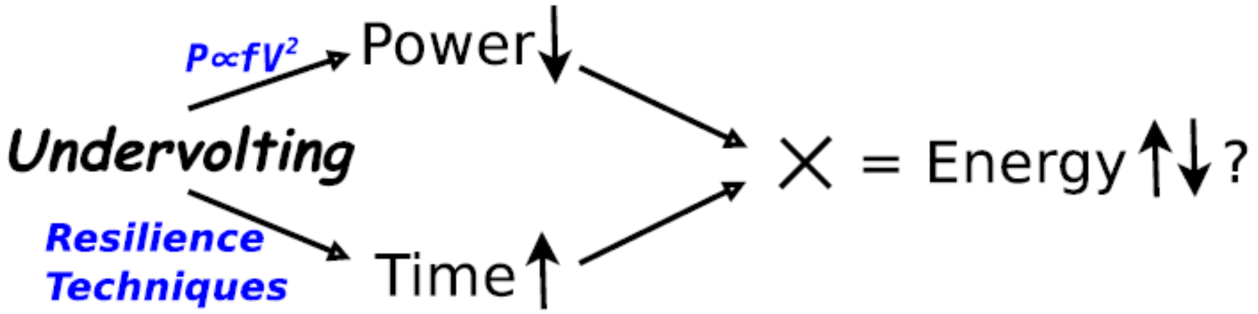
- ➢ Caused Increasing Failure Rates
  - ❖ Both *hard* & *soft* errors may occur during undervolting
  - ❖ Energy savings may be *offset*: error detection/recovery
  - ❖ Theoretical validation holds or not? Any conditions?

- ➢ 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

Detailed Objectives of this work

- Our Goals
  - *Target*: HPC systems consisting of a number of nodes connected by networks based on msg-passing communication.
  - Investigate the *interplay* between *energy efficiency* and *performance loss* due to error detection and recovery at the increased failure rates from undervolting.
  - *Theoretically* and *empirically* study if undervolting combined with mainstream resilience techniques can save overall energy without significant performance overhead.
  - If undervolting is feasible to save additional energy on top of the state-of-the-art frequency-directed DVFS solutions.

Contributions



- Key Contributions
  - We observe that energy saving could be achieved using undervolting by leveraging appropriate mainstream resilience techniques
  - No requirements of pre-production machines and no modifications to the hardware
  - *Modeling* performance and energy under undervolting anylatically
  - Up to 12.1% energy savings against baseline and 9.1% more energy saved than a state-of-the-art DVFS technique (Adagio).

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

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

  - Here, we manage to present the average failure rate as a function of supply voltage only.

An Example

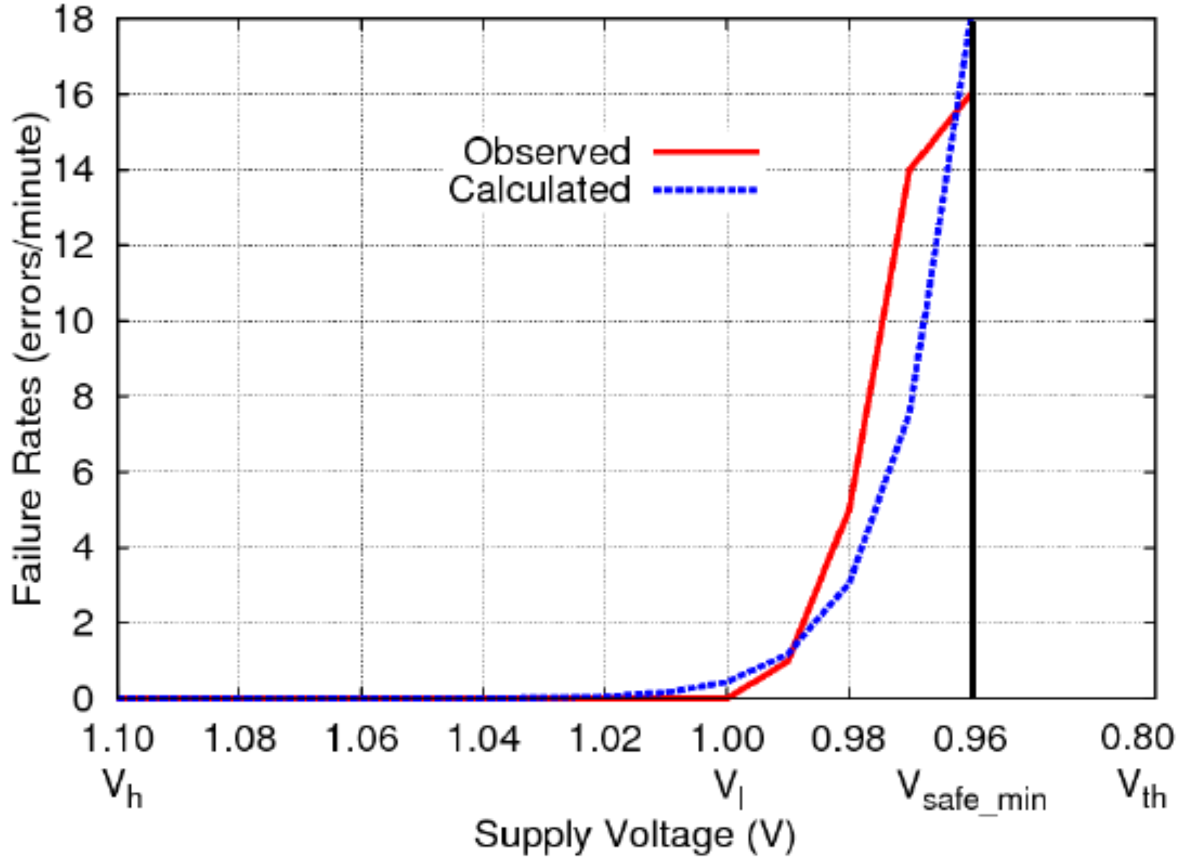


Figure 2. Observed and Calculated Failure Rates as a Function of Supply Voltage for a Pre-Production Intel Itanium II 9560 8-Core Processors (V_h: max volt. paired with max freq., V_l: min voltage paired with min freq., V_{safe_min}: min volt. for pre-production processors, V_{th}: threshold volt.).

Main-stream Software-level 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 (CR and ABFT only)

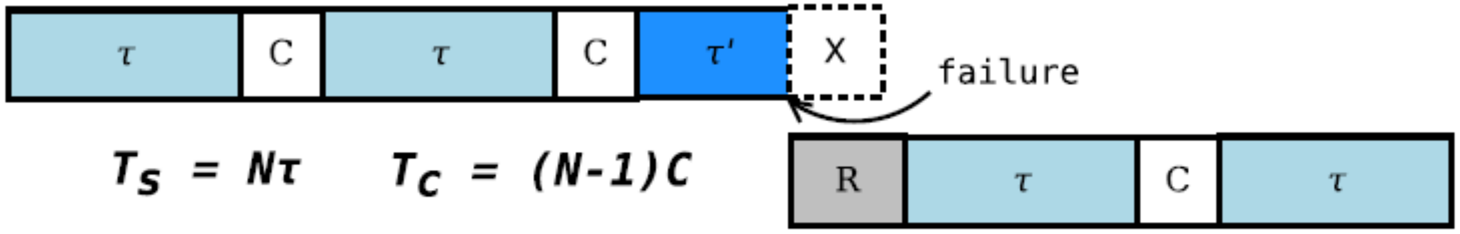


Figure 3. Checkpoint/Restart Execution Model for a Single Process.

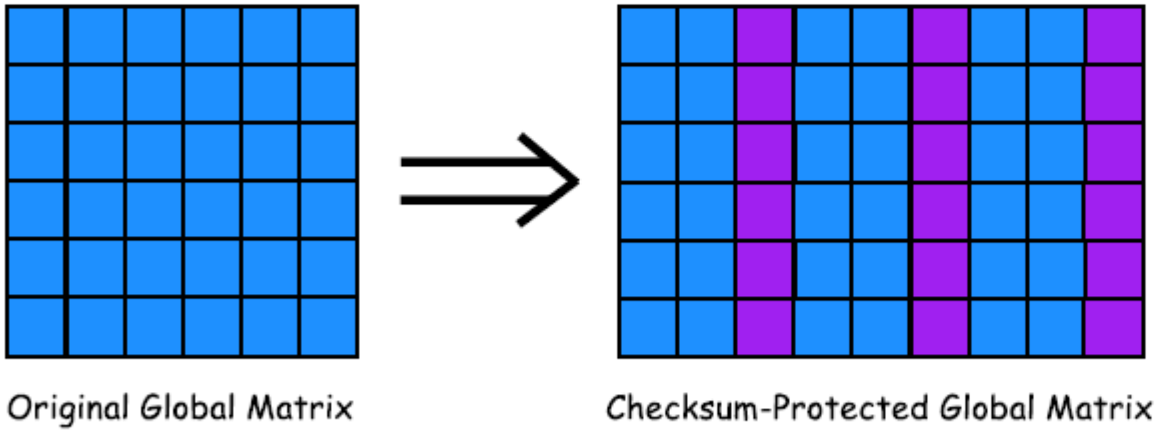


Figure 4. Algorithm-Based Fault Tolerance Model for Matrix Operations.

Performance Modeling

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

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

    - 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:

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

Performance Modeling (Cont.)

- Algorithm-Based Fault Tolerance (ABFT) for Matrix Operations (Cholesky/LU/QR factorization)
  - In CR, 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 (for example, ABFT-enabled dense matrix factorizations--Cholesky factorization):

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

- Performance modeling for other resilience techniques is conceptually similar

Power Consumption Modeling

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

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

  - Consider three power patterns for a node doing CR

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

Energy Consumption Modeling

- With Undervolting and Resilience Techniques

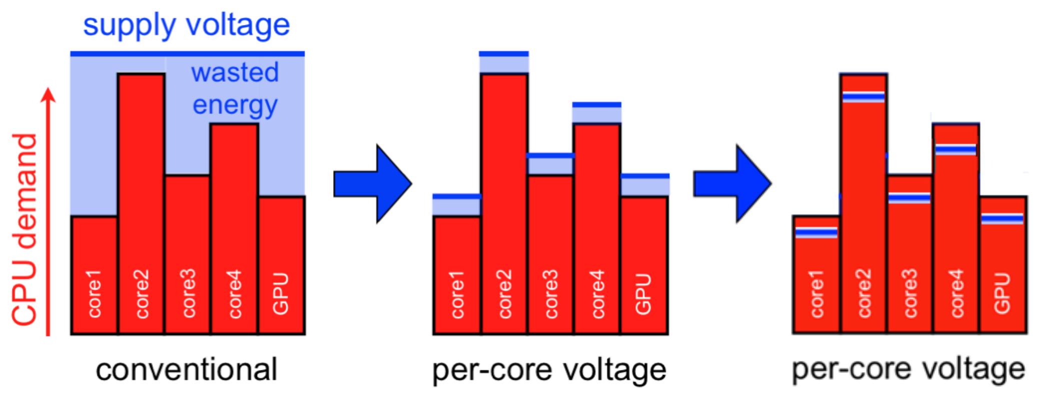
  - Use CR as an example for model building
  - Study homogeneous HPC systems w/o accelerators
  - For a HPC run, we have three variants
    - A *baseline* run with *nominal* frequency and voltage
    - A run with undervolting in the *absence* of failures
    - A run with undervolting in the *presence* of failures

  - Integrating three power patterns, energy cost models

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

Energy Savings over State-of-the-art (Adagio)

- 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.
    - Major Related work include Adagio and CPU-miser
  - Can we further save energy beyond DVFS?
    - Employ a state-of-the-art DVFS technique Adagio
    - Continue undervolting further per selected appropriate F/V
    - Also leverage resilience solutions to guarantee correctness, which costs additional overhead



Energy Savings over Adagio (Cont.)

- Our Strategy
  - 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

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

Energy Saving Conditions over Baseline

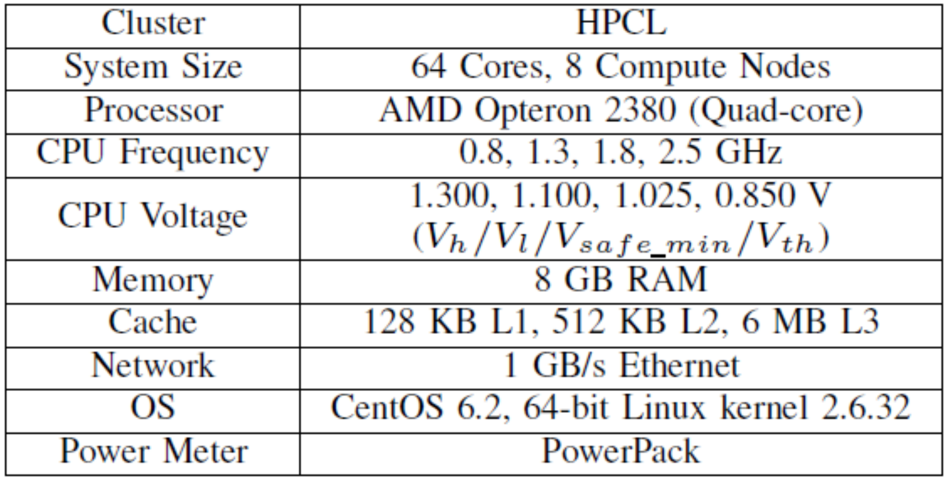
- Given Platform-dependent Parameters (c_1, c_2, c_3, AC', I_{sub}, f, V, P_C)
  - Before Model Relaxation

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

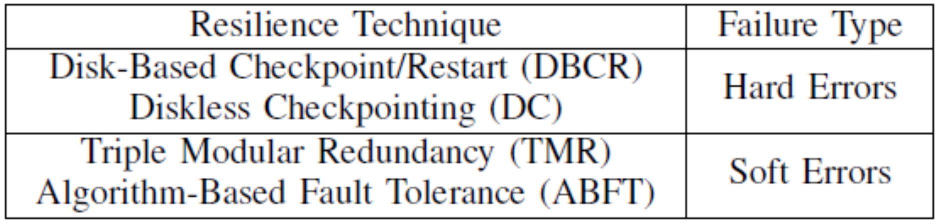
  - After Model Relaxation (N-$1 \approx N$)

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

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	Failure Type
Disk-Based Checkpoint/Restart (DBCR) Diskless Checkpointing (DC)	Hard Errors
Triple Modular Redundancy (TMR) Algorithm-Based Fault Tolerance (ABFT)	Soft Errors

Implementation

- Failure Rate Calculation
  - *Limitation*: HPC production machines do not allow further voltage reduction beyond V_l
    - No noticeable errors was observed for the voltage range V_l to *Vh* our platform (computation and memory intensive workloads running for weeks)
  - Estimate failure rates between V_l and V_{safe_min} since the testbed does not allow further voltage reduction beyond
  - Use the equation below to calculate the failure rates between V_l and V_{safe_min}
    - High accuracy shown in the previously illustrated example

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

Implementation (Cont.)

➢ Undervolting Production Processors
  - Modify the northbridge/CPU FID and VID control reg.
    - Register values are altered using *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 (commonly not accessible)
    - Advanced ECC memory support is required

Implementation (Cont.)

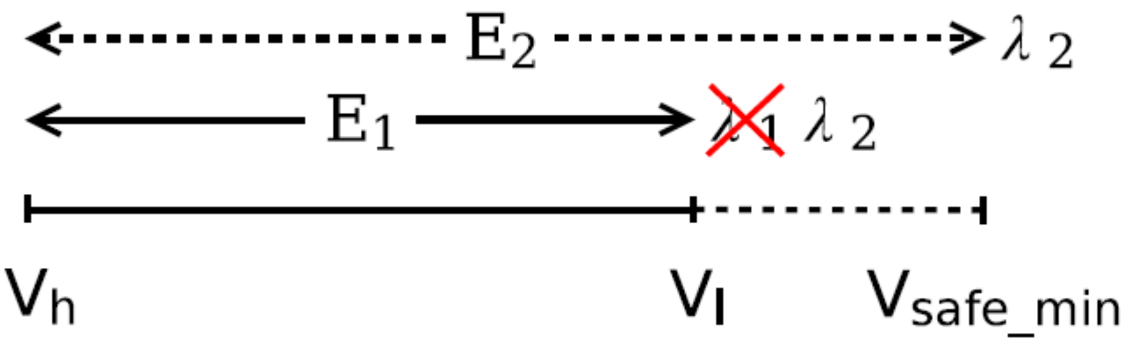
- NB/CPU FID/VID control register format and formula

***Bits* (64 bits in total)**	***Description***
63:32, 24:23, 21:19	Reserved
32:25	Northbridge Voltage ID
22	Northbridge Divisor ID
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 * 0010100h = 1.300 V

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



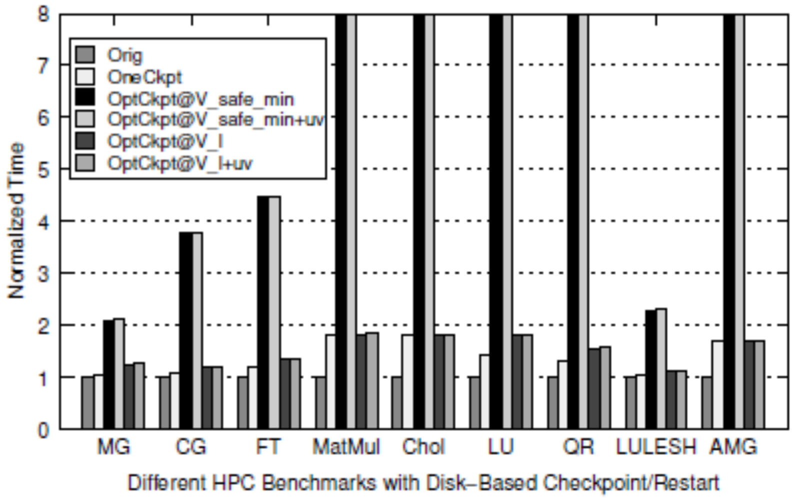
Benchmarks

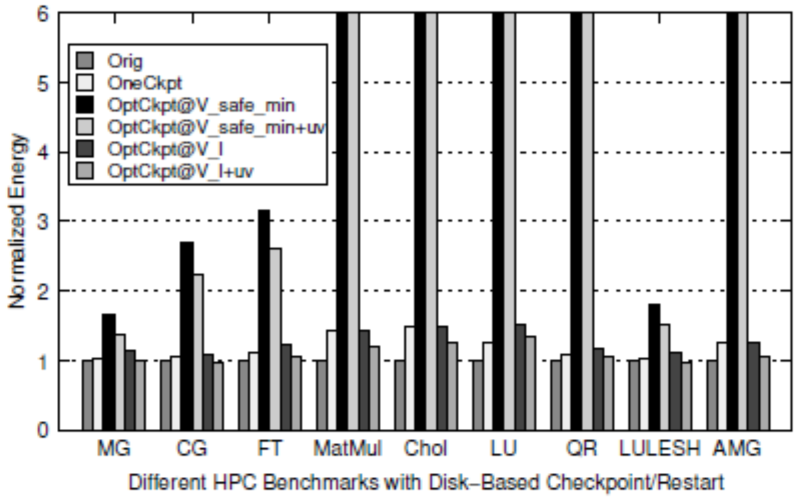
- NASA-concerned HPC Benchmarks
  - MG, CG, and FT from the NPB benchmark suite
- DOE-concerned HPC Benchmarks
  - LULESH
  - AMG
- Widely-used Numerical Linear Algebra Libraries
  - Matrix multiplication
  - Cholesky factorization
  - LU factorization
  - QR factorization

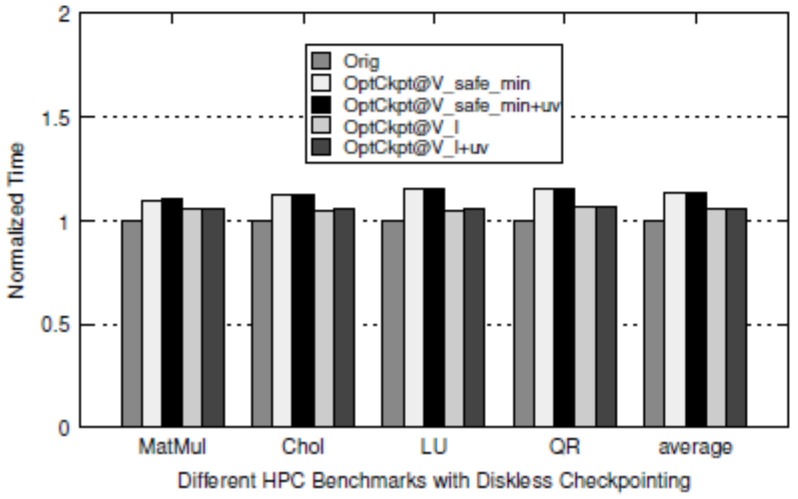
Evaluation

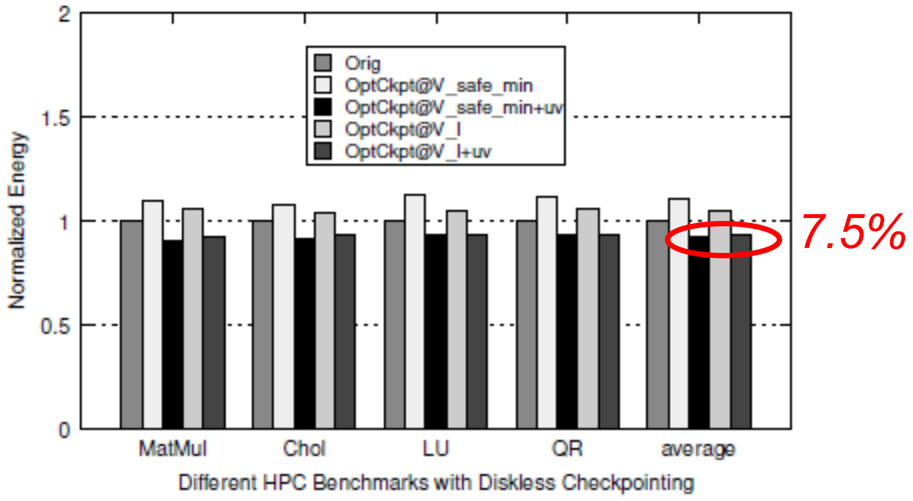
- Test Scenarios
  - Checkpoint-kind resilience techniques (DBCR/DC)
    - OneCkpt: Checkpoint/restart is only performed *once*
    - OptCkpt@Vx: Checkpoint/restart is performed with the *optimal* checkpoint interval at Vx
    - OptCkpt@Vx + uv: OptCkpt@Vx + undervolting
  - Non-checkpoint resilience 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 + nominal voltage
    - Adagio + uv: predicted frequency + undervolting

Experimental Results (DBCR vs. DC)

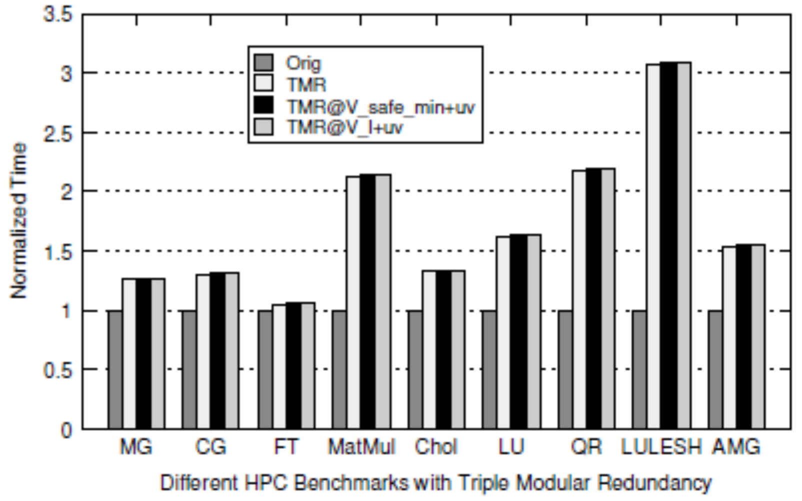


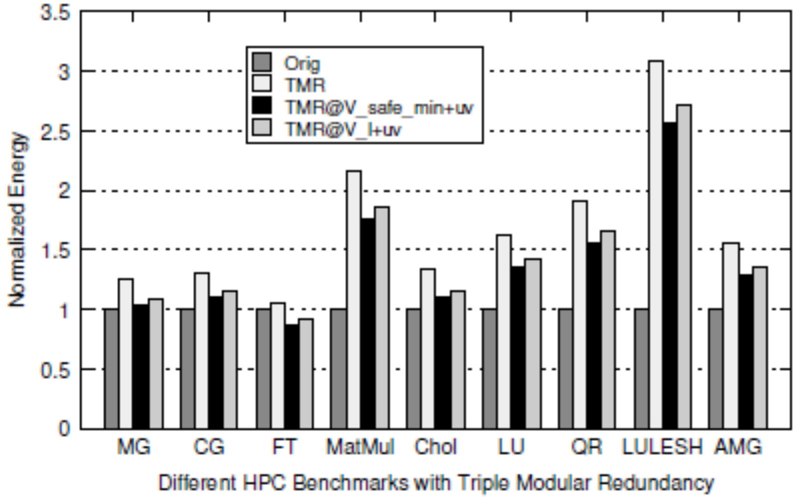


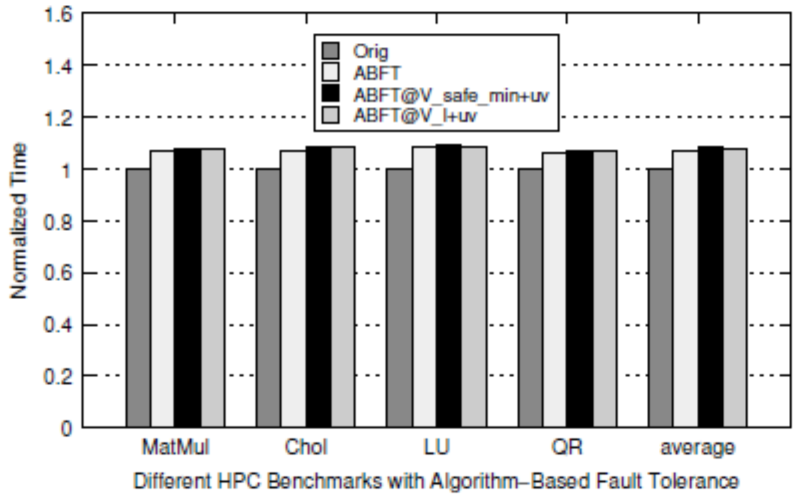


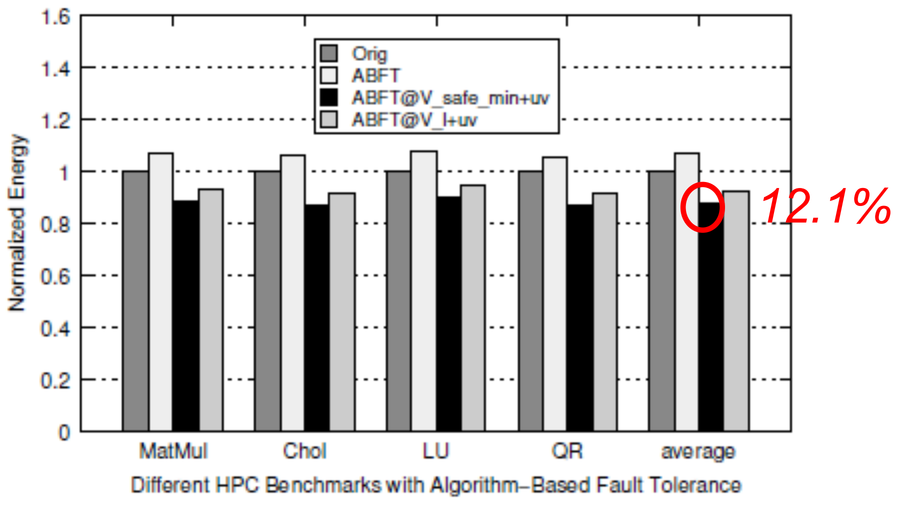


Experimental Results (TMR vs. ABFT)

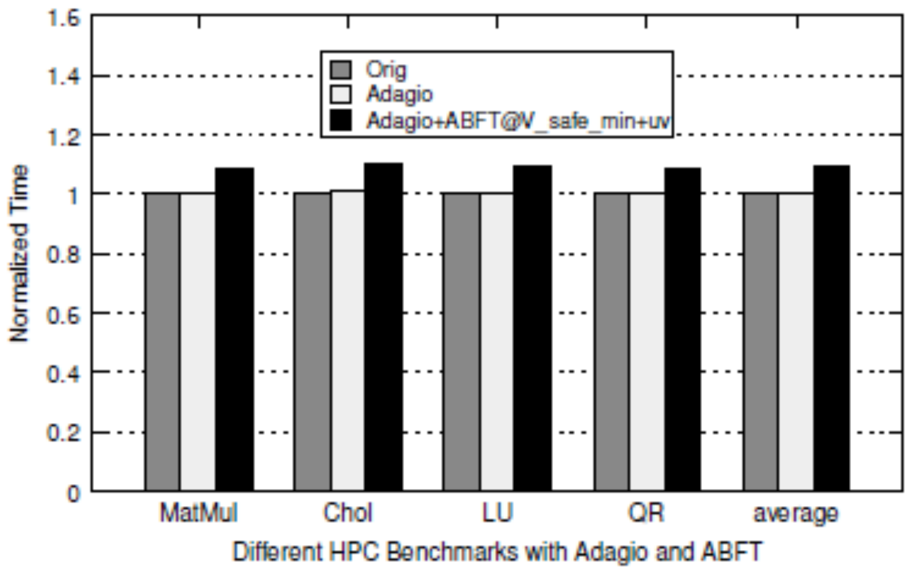


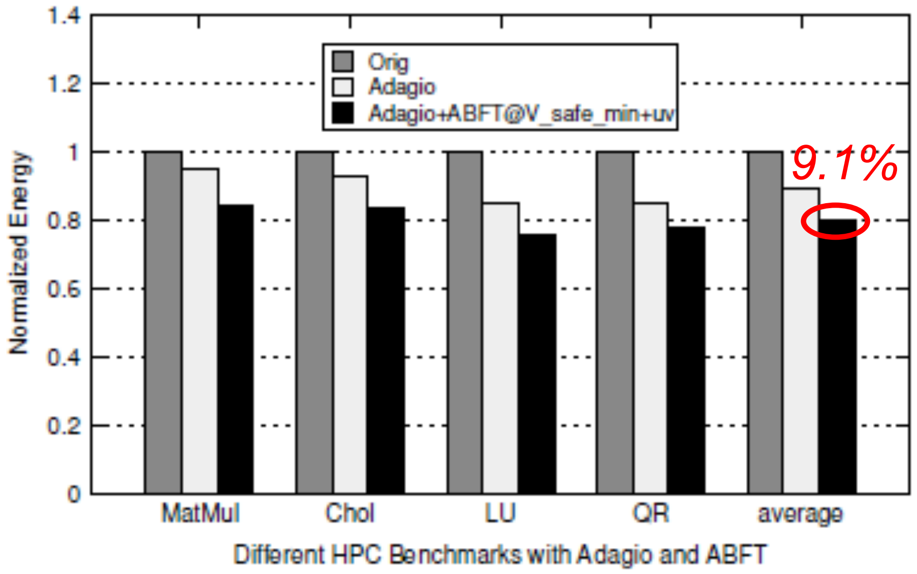






Experimental Results (Adagio + Undervolting)





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
  - – Enabling appropriate undevolting interfaces for common users might be an option for future HPC systems to save energy, without redesigning the hardware.
  - – Feasible to save energy beyond classic DVFS solutions
- ➢ Ongoing Directions
  - – Migrate undervolting to more types of hardware (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
  - – Exploring the realm of NTV and STV for future HPC scenarios