Energy-Aware Test Prioritization for High-Performance Computing: A Multi-Objective Approach

Ninad Anklesaria
Oregon State University
Corvallis, Oregon, USA
anklesan@oregonstate.edu

Manish Motwani
Oregon State University
Corvallis, OR, USA
motwanim@oregonstate.edu

Abstract

High-Performance Computing (HPC) applications require extensive testing with thousands of tests executed on supercomputers and distributed clusters. While traditional software testing is relatively cheap, HPC testing costs are orders of magnitude higher due to greater computational demands and longer execution times. Existing test prioritization techniques target traditional software systems using execution time as a proxy for energy consumption, but remain unexplored for HPC applications where this correlation breaks down due to parallel efficiency, communication overhead, and accelerator usage. We propose a novel multi-objective optimization approach that treats energy consumption as a first-class objective alongside code coverage for HPC test prioritization. Using LAMMPS, a widely-used molecular dynamics simulator, we demonstrate 97.5% average energy savings compared to exhaustive testing while maintaining 95% average statement coverage of modified code, suggesting a promising direction for sustainable HPC testing.

CCS Concepts

• **Software and its engineering** → **Search-based software engineering**; **Software testing and debugging**; • **Computing methodologies** → *Massively parallel and high-performance simulations*; *Genetic algorithms*.

Keywords

Energy-Aware HPC Testing, Test Prioritization, Test Minimization

ACM Reference Format:
Ninad Anklesaria and Manish Motwani. 2026. Energy-Aware Test Prioritization for High-Performance Computing: A Multi-Objective Approach. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 5–9, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3803437.3805582

1 Introduction

High-Performance Computing (HPC) applications run complex scientific simulations such as molecular dynamics simulations, climate modeling, and computational fluid dynamics, which require rigorous testing to ensure correctness. Modern HPC applications maintain extensive test suites including unit tests, integration tests, and regression tests, often comprising thousands of tests executed on modern supercomputers and across distributed nodes of clusters utilizing diverse accelerators such as CPUs and GPUs.

As HPC systems approach exascale [7], energy consumption has emerged as a dominant constraint on how software is developed, tested, and deployed. While traditional software systems such as Elasticsearch consume 117 kWh annually [26], HPC testing costs are likely orders of magnitude higher as large-scale supercomputers can consume megawatts of power, with energy costs often exceeding hardware acquisition costs over the system lifetime. HPC centers are responding by shifting from core-hour allocations to energy-based accounting [4], making energy consumption a first-class resource constraint alongside traditional performance metrics.

Despite the growing importance of energy as a resource constraint, testing practices for HPC applications have largely ignored it, and there is a need for better integration of software testing methods in HPC testing [13]. Test case selection and prioritization research targeting traditional systems optimizes fault detection effectiveness and execution time [28], implicitly assuming that faster tests are necessarily cheaper to run. Recent work [23, 26] shows that execution time is not a reliable proxy for energy consumption. Particularly, in HPC environments, this assumption frequently breaks down, as differences in parallel efficiency, accelerator utilization, and I/O behavior can lead to substantial variation in energy consumption among tests with similar execution times.

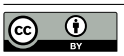
Our Vision: We propose treating energy consumption as an explicit optimization objective in test case selection and prioritization. We explore using a multi-objective search algorithm to find Pareto-optimal test orderings that balance energy efficiency against test suite effectiveness measured using code coverage. This represents a fundamental shift from viewing energy as a derived consequence of execution time to treating it as an independent, measurable, and optimizable resource throughout the software testing life cycle.

Contributions: This paper makes the following contributions:

- Motivation for energy-aware test selection and prioritization for HPC applications, addressing a gap at the intersection of green software engineering and HPC testing.
- **EN**ergy-**A**ware **C**overage-based **T**est prioritization (ENACT) approach for energy-aware HPC testing using multi-objective optimization that explicitly optimizes energy consumption and code coverage.
- Preliminary results from LAMMPS showing that 95% modified statement coverage can be achieved with 97.5% energy savings compared to exhaustive testing.

2 Background and Related Work

This section describes the challenges of testing HPC applications compared to traditional systems (Section 2.1), existing research on



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/26/07
https://doi.org/10.1145/3803437.3805582

energy consumption in software systems (Section 2.2), and existing test case selection and prioritization approaches (Section 2.3).

2.1 HPC Testing Challenges

HPC applications present unique testing challenges because they execute on increasingly heterogeneous architectures. As new features, platforms, and execution configurations are introduced, test suites grow organically, amplifying not only testing time but also the energy cost of maintaining scientific correctness. For example, LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [1] is a widely used molecular dynamics simulator in continuous development since 1995. As a large, long-lived scientific code supporting hundreds of simulation algorithms, force fields, and execution configurations [17], LAMMPS maintains an extensive test suite comprising thousands of tests spanning diverse physical models, problem sizes, and parallel decompositions. These tests target different platforms (e.g., CPUs and GPUs from vendors such as Intel and NVIDIA) and operating systems. Executing the full test suite on production HPC systems can take hours, consume tens of thousands of core-hours, and require massive amounts of energy, making exhaustive testing increasingly prohibitive at scale. Existing research on HPC testing focuses primarily on correctness challenges: numerical precision, non-determinism, race conditions, and portability across architectures [11].

2.2 Energy Considerations in Software Systems

Energy has become a fundamental constraint in high-performance computing. Beyond raw performance, the Top500 ranking now emphasizes performance per watt, and the U.S. Department of Energy's exascale computing initiative enforces strict power budgets of 20–30 megawatts for exascale systems [25]. These constraints make energy a first-class design consideration across hardware, system software, and applications.

In response, HPC centers have deployed increasingly sophisticated energy monitoring and management infrastructure [4]. Modern systems expose fine-grained energy measurements at the node, accelerator, and component levels [5, 15, 27], enabling detailed accounting of energy usage. Some centers have begun transitioning from traditional core-hour allocations to energy-based accounting models that charge users based on energy consumed rather than time on system [9, 16, 24], reflecting the growing dominance of energy costs in large-scale HPC operations.

Recent work has begun addressing energy consumption in testing. Verdecchia et al. [26] proposed energy-aware test prioritization for traditional software systems, using energy as a secondary tie-breaking criterion within a similarity-based approach. However, test prioritization remains unexplored for HPC applications where energy costs are substantially higher and the correlation between execution time and energy consumption breaks down.

2.3 Test Selection and Prioritization

Test suite minimization eliminates redundant tests, test case selection identifies tests relevant to recent code changes, and test case prioritization orders tests to maximize early fault detection [28]. The intuition is that if testing must be terminated early due to time or resource constraints, prioritized orderings will have exposed more faults than arbitrary orderings. Test selection and prioritization become essential when exhaustive testing is impractical.

Prior research has explored numerous prioritization criteria: code coverage (statement, branch, function), fault history, change information, execution cost, and combinations thereof. Multi-objective approaches using evolutionary algorithms (e.g., NSGA-II [6], MOEA/D [29]) have shown promise in balancing multiple criteria such as coverage and execution time [8, 22]. However, existing work suffers two critical limitations for HPC contexts. First, execution time is used as a proxy for cost, ignoring energy as an independent dimension. In HPC environments, parallel efficiency, communication overhead, accelerator utilization, and I/O patterns can cause substantial variation in energy consumption among tests with similar execution times. Second, these techniques are validated primarily on traditional software (web applications, mobile apps, embedded systems) with fundamentally different characteristics than HPC applications. Our prior work has addressed automated repair [19, 20] and LLM-guided differential fuzzing for detecting platform-specific bugs in scientific applications [21], but energy has not been treated as an optimization target in either context.

3 ENACT Approach

Given a test suite $T = \{t_1, \ldots, t_n\}$ and a code change Δ with modified statements S_Δ, ENACT seeks a subset $T' \subseteq T$ and an execution ordering π that jointly optimize the following four objectives:

1. Maximize Modified Statement Coverage:
$C(T') = \frac{|\bigcup_{t \in T'} cov(t) \cap S_\Delta|}{|S_\Delta^{cov}|}$

2. Minimize Total Energy Consumption: $E(T') = \sum_{t \in T'} e(t)$

3. Minimize Total Execution Time: $\tau(T') = \sum_{t \in T'} \tau(t)$

4. Minimize Test Suite Size: $|T'|$

Here, $cov(t)$ denotes the set of statements covered by test t, $e(t)$ is the energy consumption in joules, and $\tau(t)$ is the execution time in seconds. $S_\Delta^{cov} \subseteq S_\Delta$ denotes the subset of modified statements that are *coverable*—i.e., exercised by at least one test in T. Normalizing by $|S_\Delta^{cov}|$ avoids penalizing solutions for untestable changes.

Algorithm 1 presents the ENACT optimization procedure, which unifies test *selection* and *prioritization* into a single multi-objective framework.

Initialization and Evaluation (Lines 1–3). Line 1 generates an initial population P of N random configurations, each encoding a test subset T' and execution ordering π. Lines 2–3 decode and evaluate each configuration on the four objectives.

Evolution (Lines 5–9). Over G generations, the population evolves. Line 6 generates offspring by applying Simulated Binary Crossover [6] to pairs of parent configurations, allowing offspring to inherit complementary test subsets and orderings from both parents, with polynomial mutation introducing further diversity. Line 7 decodes and evaluates each offspring. Line 8 merges parents and offspring into a combined pool R. Line 9 ranks R by pairwise comparison: (T_1', π_1) is superior to (T_2', π_2) if it is at least as good on all objectives and strictly better on at least one. Line 10 retains the N best configurations, preferring diverse trade-offs.

Energy-Aware Reordering (Lines 12–15). Line 12 extracts the optimal configurations $\mathcal{P}$ from the final population. Lines 13–15 reorder tests so that those covering more modified statements per joule run earlier, enabling faster fault detection.

Algorithm 1 ENACT: Multi-Objective Test Optimization

Require: Test suite T, modified statements S_Δ, parameters N, G
Ensure: Pareto front $\mathcal{P}$ of optimal (T', π) configurations
1: $P \leftarrow$ Initialize N random configurations encoding (T', π)
2: **for** each $s \in P$ **do**
3: Decode s into (T', π); evaluate $C(T'), E(T'), \tau(T'), |T'|$
4: **end for**
5: **for** $g = 1$ to G **do**
6: $Q \leftarrow$ Generate offspring from P
7: Decode and evaluate each offspring in Q
8: $R \leftarrow P \cup Q$
9: Rank R by pairwise objective comparison
10: $P \leftarrow$ Select best N configurations from R
11: **end for**
12: $\mathcal{P} \leftarrow$ Extract optimal configurations from P
13: **for** each $(T', \pi) \in \mathcal{P}$ **do**
14: Reorder π by marginal coverage per joule
15: **end for**
16: **return** $\mathcal{P}$

Output (Line 16). The algorithm returns $\mathcal{P}$, where each configuration represents a distinct trade-off among coverage, energy, time, and suite size. To quantify the benefit of reordering, we measure the cumulative energy to reach coverage thresholds of 80%, 90%, 95%, and 100%.

Implementation and Parameter Settings. We implement ENACT using pymoo [3], setting population size (N) and number of generations (G) to 80 and 120 respectively based on a small parameter sweep, and averaging results over three random seeds.

4 Evaluation

This section describes our experiment procedure (Section 4.1) to evaluate ENACT on LAMMPS and the preliminary results (Section 4.2) analyzing the trade-off between modified code coverage, test suite execution time, and energy consumption.

4.1 Experiment Procedure

Data Collection: We collected per-test energy consumed, execution time, and modified code coverage for LAMMPS as described below.

Energy data: We measured CPU and DRAM energy consumption using Intel RAPL [12] . Tests were executed either serially or in sharded workflows across multiple nodes, with each shard running tests serially, enabling reliable per-test energy attribution. The full test-suite consisting of 613 tests consumes 188.6 kJ of CPU and DRAM energy and 5812 seconds of execution time per run.

Modified statement coverage and *test execution time:* Statement-level coverage was collected using `gcov`. We aggregated these data across the LAMMPS source code to map specific test cases to the program statements they exercise. We measured individual test execution time using Python's `time.perf_counter()`.

Code change data: We analyzed commit-level code changes from the LAMMPS GitHub repository to identify modified source-code statements and evaluate test coverage with respect to these changes. Data were collected using the LAMMPS Linux CI workflow on GitHub Actions [18]. We considered commits from a one-year period and associated each commit with its corresponding unit test executions. In total, we analyzed 2,746 commit-associated test-suite executions. After removing duplicate executions, failed or incomplete runs, and commits involving only non-source-code changes, we retained 1,257 unique test runs corresponding to distinct source-code changes.

Table 1: Energy Savings and Test Selection Statistics. All values are percentages relative to full suite execution ($n = 613$) across 1,257 runs.

Coverage	Energy Savings (%)				Avg. Tests
Target	Mean	Median	p10	p90	Selected
80%	99.27	99.82	99.04	99.94	8.04
90%	98.51	99.71	96.88	99.94	15.82
95%	97.53	99.65	95.39	99.93	23.83
100%	94.34	99.36	88.41	99.93	48.29

Infrastructure. Experiments were run on homogeneous production HPC compute nodes consisting of three Dell PowerEdge R740 servers, each equipped with dual 12-core Intel Xeon Silver 4116 processors (2.10 GHz) and 256 GB of RAM. Nodes were interconnected via InfiniBand, and experiments were executed under controlled conditions to ensure measurement consistency.

For each of the 1,257 modified code changes, we ran ENACT to explore trade-offs between modified statement coverage, test energy consumption, and test execution time. Results were aggregated across multiple executions.

4.2 Preliminary Results

We analyze the relationship between energy consumption and modified statement coverage using the Pareto-optimal test subsets derived using ENACT.

Non-trivial trade-offs exist: As illustrated in Figure 1, the resulting Pareto fronts exhibit substantial variation in energy–coverage efficiency across different commit sizes. Certain tests deliver high coverage-per-joule, while others incur disproportionately high energy costs for marginal coverage gains. This variability confirms that coverage effectiveness and energy consumption are weakly correlated, motivating energy-aware test selection rather than coverage or time-based heuristics alone.

Substantial energy savings are achievable through early stopping: As seen in Table 1, when test execution is stopped after reaching 80% of change-aware coverage, the median energy saving relative to full-suite execution is 99.8%, with the 10th–90th percentile range spanning 99.0% to 99.9%. Even at higher coverage targets, savings remain pronounced: at 90% coverage, the median saving is 99.7%, and at 95% coverage, the median saving remains 99.6%. These results demonstrate that a small, carefully selected subset of tests is often sufficient to exercise the majority of modified code, while avoiding the overwhelming energy cost of executing the full test suite. Our experiments show that the energy consumed for running ENACT is only 94.25 J (0.05%) of the average energy saved by running the minimized test suites, confirming that ENACT's overhead is negligible relative to energy savings.

Diminishing returns dominate near-complete coverage: As seen in the "p10" column of Table 1, marginal coverage gains

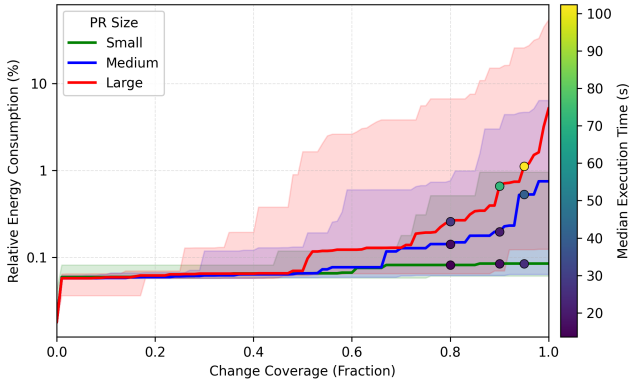


Figure 1: Consolidated Pareto Frontiers. Lines show the median relative energy cost (log-scale) to achieve coverage targets for Small ($\leq$ 14 lines), Medium (15 – 53 lines), and Large ($>$ 53) code changes. Shaded regions denote $p10$–$p90$ percentile bands. Markers are color-coded by median execution time at target coverage, reflecting the non-linear relation between testing duration and energy expenditure.

near 100% are dominated by energy-intensive tests with low incremental value. This supports adaptive stopping criteria tuned to coverage–energy budgets.

Execution time is not a reliable proxy for energy: Despite similar cumulative execution times, our analysis shows that individual tests often differ in energy consumption by factors of 2–3× due to variations in computational intensity, memory access patterns, and parallel efficiency. This observation validates treating energy as a first-class optimization objective, rather than inferring energy costs solely from execution time.

Comparison with baselines: At 95% coverage target, ENACT outperforms: (1) Random test selection that achieves a mean energy saving of 44.76% (vs. 97.53%), and (2) Greedy coverage-per-joule approach that achieves a mean energy saving of 87.9% (vs. 97.53%).

5 Vision and Future Directions

Our results hint at something larger than test prioritization: energy consumption may be a previously invisible dimension of software quality assurance in scientific computing. Just as fault detection and execution time have dedicated tooling and community benchmarks, energy must become a measurable, optimizable, and reportable property of HPC testing and debugging workflows. The software engineering community has developed sophisticated techniques for time-efficient testing; we believe an equally rich discipline awaits for making testing energy-efficient, with HPC as the right domain to seed it.

Short–term vision: Establishing Generalizability. The most critical open challenge is whether the energy–coverage trade-off structure observed in LAMMPS generalizes across HPC test suites or is an artifact of molecular dynamics workloads. This is the linchpin: techniques that are LAMMPS-specific offer limited scientific impact. To resolve it, we will apply ENACT without modification to three applications with fundamentally different computational profiles: QMCPACK (stochastic, GPU-intensive) [14], Quantum ESPRESSO (memory-bound, iterative) [10], and MFEM (communication-intensive, distributed) [2]. For each, we will measure whether a small Pareto-optimal subset still captures 95% coverage at under 5% of full-suite energy cost. A positive result across all three would establish generalizability and justify the long-term agenda below; a negative result would instead motivate application-specific energy modeling as the next priority. Concurrently, we will replace statement coverage with mutation scores, since control-flow exercise alone poorly captures numerical correctness, and mutation-based effectiveness is the stronger validity criterion the community will require in practice.

A prerequisite to this challenge is solving per-test energy attribution in realistic HPC settings. Our current setup measures energy under controlled serial execution. In production, energy is reported at the component level, tests may contribute to overlapping measurement intervals, and concurrent execution across nodes makes attribution ambiguous. Resolving this is a necessary step toward making energy-aware test prioritization deployable in real HPC workflows.

Long-term vision: Energy-Aware Testing and Debugging. Assuming generalizability is confirmed, we envision a broader research program treating energy as a first-class concern across the full HPC quality assurance lifecycle. On the testing side, adaptive systems would continuously update energy-coverage models from historical CI executions, dynamically reprioritize tests as code evolves, and report energy alongside fault detection effectiveness as a standard metric in test suite evaluation. On the debugging and repair side, the opportunity is less explored and potentially higher impact. Fault localization in HPC typically requires executing many diagnostic test configurations at scale, and automated program repair compounds this further by repeatedly executing candidate patches to validate correctness. Yet the relationship between energy expenditure and progress through the localize-then-repair pipeline is entirely unstudied. We envision techniques that minimize the energy cost of the full diagnostic and repair sequence, treating each execution as an information-theoretic investment and prioritizing patch candidates by their likelihood of success per joule. Together, energy-aware testing, debugging, and repair represent a novel synergy between green software engineering, automated program repair, and HPC systems research—communities that have not previously intersected around energy as a shared concern.

6 Conclusion

As HPC systems scale toward exascale and beyond, energy efficiency becomes paramount. Software testing practices must evolve to treat energy as a first-class resource constraint. We present a vision for energy-aware multi-objective test selection and prioritization, formulating test ordering as an explicit optimization problem balancing energy consumption against test suite effectiveness.

Our preliminary results from LAMMPS demonstrate that this approach is both feasible and effective, revealing non-trivial energy-coverage trade-offs. Substantial energy savings (97.5%, on average) are achievable while maintaining high modified code coverage (95%, on average), offering practitioners flexible strategies that match their resource constraints and quality requirements.

Acknowledgments

This research was supported by startup funds provided by Oregon State University.

References

[1] 2026. LAMMPS Molecular Dynamics Simulator. https://www.lammps.org. Accessed: January 2026.

[2] Robert Anderson et al. 2021. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications* 81 (2021), 42–74. https://doi.org/10.1016/j.camwa.2020.06.009

[3] Julian Blank and Kalyanmoy Deb. 2020. pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509. https://doi.org/10.1109/ACCESS.2020.2990567

[4] Julita Corbalán, Luigi Brochard, and Karsten Kutzer. 2022. *Optimizing Power and Energy in HPC Data Centers with Energy Aware Runtime.* Technical White Paper. Lenovo Press. https://lenovopress.lenovo.com/lp1646.pdf

[5] S Corda, M Bacis, C Cavazzoni, and P Gschwandtner. 2023. Accurate Measurement of Application-level Energy Consumption for Energy-Aware Large-Scale Simulations. *Journal of Computational Science* 66 (2023), 101933.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[7] Exascale Computing Project. 2024. About the Exascale Computing Project. U.S. Department of Energy. https://www.exascaleproject.org/about/ Accessed: 2024.

[8] Kamal Garg and Shashi Shekhar. 2024. Optimizing Test Case Prioritization Through Ranked NSGA-2 for Enhanced Fault Sensitivity Analysis. *Innovations in Systems and Software Engineering* 20 (2024), 307–328. https://doi.org/10.1007/s11334-024-00561-6

[9] Yiannis Georgiou, Thomas Cadeau, David Glesser, Danny Auble, Morris Jette, and Matthieu Hautreux. 2014. Energy Accounting and Control with SLURM Resource and Job Management System. In *International Conference on Distributed Computing and Networking*. Springer, 96–118.

[10] Paolo Giannozzi et al. 2009. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter* 21, 39 (2009), 395502. https://doi.org/10.1088/0953-8984/21/39/395502

[11] Maya Gokhale, Ganesh Gopalakrishnan, Jackson Mayo, Santosh Nagarakatte, Cindy Rubio-Gonz'alez, and Stephen F. Siegel. 2023. Report of the DOE/NSF Workshop on Correctness in Scientific Computing, June 2023, Orlando, FL. *ArXiv* abs/2312.15640 (2023). https://api.semanticscholar.org/CorpusID:266551439

[12] Intel Corporation. 2022. Running Average Power Limit (RAPL) Energy Reporting. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html.

[13] Upulee Kanewala and James Bieman. 2014. Testing Scientific Software: A Systematic Literature Review. *Information and Software Technology* 56 (10 2014). https://doi.org/10.1016/j.infsof.2014.05.006

[14] Jeongnim Kim et al. 2018. QMCPACK: an open source *ab initio* quantum Monte Carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter* 30, 19 (2018), 195901. https://doi.org/10.1088/1361-648X/aab9c3

[15] Michael Knobloch, M Foszczynski, W Schwarz, T Zeiser, G Hager, and G Wellein. 2014. Mapping fine-grained power measurements to HPC application runtime characteristics on IBM POWER7. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 850–857.

[16] Adam Krzywaniak and Pawel Czarnul. 2021. Energy-based Accounting Model for Heterogeneous Supercomputers. *arXiv preprint arXiv:2110.09987* (2021).

[17] Lammps. [n. d.]. Lammps/lammps: Public development project of the LAMMPS MD software package. https://github.com/lammps/lammps

[18] lammps. 2026. Unittest for Linux /w LAMMPS_BIGBIG · Workflow runs · lammps/lammps. https://github.com/lammps/lammps/actions/workflows/unittest-linux.yml

[19] Manish Motwani. 2021. High-Quality Automated Program Repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 309–314. https://doi.org/10.1109/ICSE-Companion52605.2021.00134

[20] Manish Motwani and Yuriy Brun. 2023. Better Automatic Program Repair by Using Bug Reports and Tests Together. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, Melbourne, Australia, 1225–1237. https://doi.org/10.1109/ICSE48619.2023.00109

[21] Manish Motwani, Aakash Kulkarni, Yunhan Qiao, Matthew Davis, and Ziyan Chen. 2025. LLM-Guided Differential Fuzzing for Detecting Platform-Specific Bugs in Scientific Applications. In *Proceedings of the IEEE International Conference on AI x Software Engineering (AIxSE)*. IEEE, Laguna Hills, CA, USA, 59–66. https://doi.org/10.1109/AIxSE64906.2025.00015

[22] Lian Qu, Zihe Song, Yingxian Zhou, Yiheng Huang, and Junjie Chen. 2024. Segment-Based Test Case Prioritization: A Multi-objective Approach. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024)*. ACM, New York, NY, USA. https://doi.org/10.1145/3650212.3680349

[23] Enrique Barba Roque, Luis Cruz, and Thomas Durieux. 2025. Unveiling the Energy Vampires: A Methodology for Debugging Software Energy Consumption. In *Proceedings of the 47th International Conference on Software Engineering (ICSE)*. IEEE, 2406–2418.

[24] Saransh Sukprasert, Kent Milfeld, Noel Gaffney, and Dan Stanzione. 2025. Core Hours and Carbon Credits: Incentivizing Sustainability in HPC. *arXiv preprint arXiv:2501.09557* (2025).

[25] U.S. Department of Energy. 2024. FY 2024 Advanced Scientific Computing Research Budget Request. https://www.energy.gov/documents/fy-2024-advanced-scientific-computing-research-budget-request.

[26] Roberto Verdecchia, Emilio Cruciani, Antonia Bertolino, and Breno Miranda. 2025. Energy-Aware Software Testing. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering: New Ideas and Emerging Results* (Ottawa, Ontario, Canada) *(ICSE-NIER '25)*. IEEE Press, 101–105. https://doi.org/10.1109/ICSE-NIER66352.2025.00026

[27] Joseph P White, Ryan T Plessinger, Robert L DeLeon, Thomas R Furlani, and Steven M Gallo. 2018. Monitoring and Analysis of Power Consumption on HPC Clusters using XDMoD. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 853–862.

[28] Shin Yoo and Mark Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing, Verification and Reliability* 22, 2 (2012), 67–120. https://doi.org/10.1002/stvr.430

[29] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007), 712–731. https://doi.org/10.1109/TEVC.2007.892759