Heterogeneity-Aware Task Allocation for Modern HPC Systems

Sowmya Yellapragada[1], Jessica Imlau Dagostini[2], Kevin Gott[3], Rebecca Hartman-Baker[3]
[1]University of Utah, [2]University of California, Santa Cruz, [3]Lawrence Berkeley National Laboratory

U.S. DEPARTMENT OF ENERGY
Office of Science

ABSTRACT

Modern supercomputing systems exhibit performance heterogeneity even across seemingly identical hardware due to memory capacity differences and deployment variations. This heterogeneity causes substantial computational waste in scientific applications. We present Performance-Aware and Relation-Aware load balancing algorithms that use empirically measured node characteristics and a Relative Performance Matrix (R_{ij}) to optimize task distribution. Evaluation on NERSC Perlmutter with 14 AMReX computational kernels with broad applicability to memory-bound and compute-intensive applications in climate modeling, astrophysics, and computational fluid dynamics, demonstrates 99.9% scheduling efficiency, achieving 4.4-11.5% improvements over traditional methods in moderate heterogeneity scenarios and up to 300x improvements in extreme CPU-GPU configurations.

PROBLEM DEFINITION

- Given: Task vector T = {t_1, t_2, …, t_n} with execution times and performance factors of available nodes P = {p_1, p_2, …, p_m}
- Objective: Minimize makespan = max(L_j/p_j) where L_j is the total work assigned to node j
- System Characteristics:
  - Performance heterogeneity: moderate (1.09x - 1.26x for A100 variants) to extreme (50x - 430x for CPU/GPU)
  - Relation matrix $R_{ij} = p_j / p_i$ captures relative node capabilities.

LIMITATIONS & SCOPE

- Traditional homogeneous schedulers fail to reach the theoretical optimal makespan of *total_work/total_capacity*, creating significant computational waste at scale.
- Task transferability: Assumes tasks can migrate between nodes
- Two-phase operation: Performance profiling followed by task assignment
- Application scope: Optimized for compute-intensive scientific kernels

RELATED WORK

- Traditional approaches: Work-Stealing, HEFT, CPA achieve 60-80% efficiency
- Our contribution:
  - Performance-aware + Relation-aware algorithms
  - Empirical performance measurements
  - R_{ij} matrix optimization

METHODOLOGY

- **Algorithms Design**:
  1. **Performance-Aware:** Greedy assignment using scaled completion times from measured node performance.
  2. **Relation-Aware:** Uses R_{ij} matrix for smart redistribution with imbalance penalties and bonus alignment.
- **Performance Characterization:**
  a. Performance Modeling: $p_i = t_{baseline} / t_i$
  b. Heterogeneity, $H = \max(p_i) / \min(p_i)$
- **Benchmarking:** Run representative AMReX kernels on NERSC Perlmutter system with A100 40GB and 80GB nodes.

FRAMEWORK

Homogeneous Load Balancer

```
INPUT: tasks[], nodes[]
1. INITIALIZE node_loads[] = 0
2. FOR each task:
3.     best_node = argmin(node_loads[i])
4.     ASSIGN task to best_node
5.     UPDATE node_loads[best_node] += task.size
```

Performance-Aware Load Balancer

```
INPUT: tasks[], nodes[], performance_factors[]
1. INITIALIZE node_loads[] = 0
2. SORT tasks by size (largest first)
3. FOR each task:
4.     best_node = argmin(node_loads[i] + task.size/perf[i])
5.     ASSIGN task to best_node
6.     UPDATE node_loads[best_node] += task.size/perf[best_node]
```

Relation-Aware Load Balancer

```
INPUT: tasks[], nodes[], performance_factors[], rij_matrix[][]
1. INITIALIZE node_loads[] = 0
2. SORT tasks by size (largest first)
3. FOR each task:
4.     FOR each node i:
5.         completion_time = node_loads[i] + task.size/perf[i]
6.         balance_penalty = compute_imbalance_penalty(i, rij_matrix)
7.         redistribution_bonus = compute_redistribution_bonus(i, rij_matrix)
8.         score[i] = completion_time + balance_penalty - redistribution_bonus
9.     best_node = argmin(score[i])
10.    ASSIGN task to best_node
11.    UPDATE node_loads[best_node] += task.size/perf[best_node]
```

Evaluation Methodology

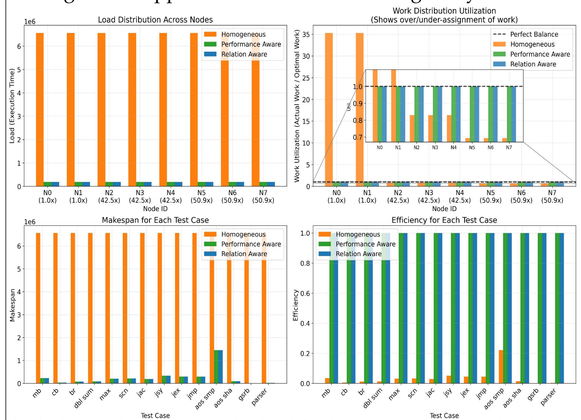
- **Test Configuration:**
  - 1M tasks across 6 heterogeneous nodes
  - NERSC Perlmutter A100 3x(40GB+80GB)
  - Grid Sizes: 256^3, 128^3, 64^3, 32^3
- **Benchmark Suite:** 14 AMReX kernels
  - **Memory-bound:** *mb* (daxpy), *dbl sum* (reduce), *scn* (scan), *max* (reduction)
  - **Compute-intensive:** Jacobi variants (*jac, jsy, jex, jmp*), *cb* (compute-bound), *gsrb* (Gauss-Seidel Red-Black)
  - **Memory access patterns:** *aos smp/sha* (Array of Structures operations)
  - **Specialized:** *br* (branching), *parser* (parsing workloads)
- **Performance Metrics:**
  - Efficiency:*ideal_makespan/actual_makespan*
  - Load balance variance across nodes
  - Heterogeneity: 1.09x – 1.26x
  - Scalability: tested across grid resolutions
  - Complexity: From *O(nlogn + nlogm) to O(nlogn + nm)*, for n tasks, m nodes
- Beyond AMReX: Climate modeling, astrophysics, CFD, ML distributed training
- Applicable to: Independent tasks, measurable heterogeneity, load imbalances

RESULTS

CPU + GPU Mixed System

Kernel	Heterogeneity	Homogeneous Efficiency	Perf-Aware Efficiency	Rel-Aware Efficiency
gsrb	430.5x	0.3%	99.9%	99.9%
parser	255.9x	0.5%	99.9%	99.9%
cb	127.2x	1.2%	99.9%	99.9%
jac	50.9x	2.8%	99.9%	99.9%

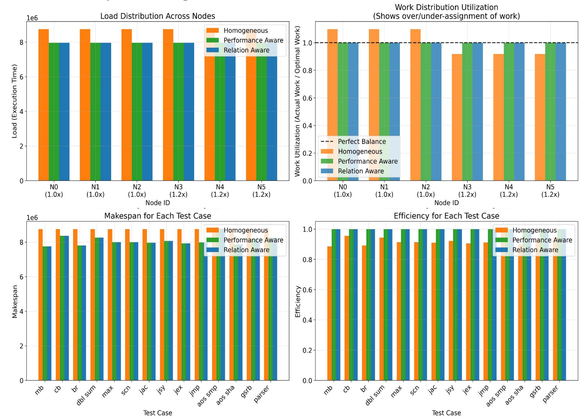
Finding: Up to **300x efficiency improvement** over homogeneous approaches in extreme heterogeneity scenarios.



A100 40GB vs 80GB

Kernel	Heterogeneity	Homogeneous Efficiency	Perf-Aware Efficiency	Rel-Aware Efficiency
mb	1.26x	88.5%	99.9%	99.9%
br	1.24x	89.2%	99.9%	99.9%
jac	1.20x	91.0%	99.9%	99.9%
cb	1.09x	95.6%	99.9%	99.9%

Finding: 4.4% – 11.5% efficiency improvements even in moderately heterogeneous environments.

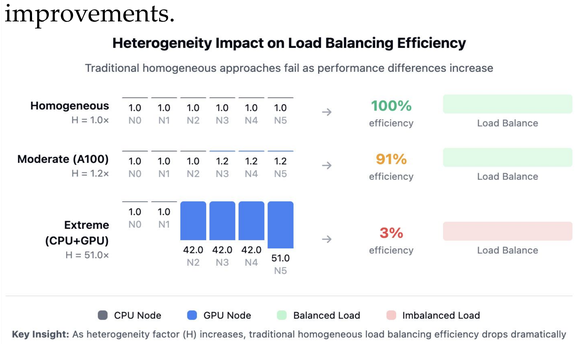


Results shown are for grid size 128; load and work distribution correspond to the **`jac`** kernel.

A100 GPU 40GB vs 80GB

Kernel	40GB Time	80GB Time	Perf Ratio	Speedup
mb (daxpy)	3.10e-04	2.46e-04	0.794	1.26x
br (branch)	2.15e-04	1.73e-04	0.805	1.24x
aos sha	2.37e-03	1.93e-03	0.814	1.23x
jex (Jacobi)	6.68e-04	5.53e-04	0.828	1.21x
jac (Jacobi)	6.44e-04	5.38e-04	0.835	1.20x
jmp (Jacobi)	6.48e-04	5.42e-04	0.836	1.20x
max(reduction)	1.70e-04	1.43e-04	0.841	1.19x
parser	1.69e-03	1.51e-03	0.894	1.12x
cb (compute)	8.74e-04	8.01e-04	0.916	1.09x

Memory-intensive kernels show 12-26% advantage for 80GB, while *compute-bound* kernels show 9-14% improvements.



CONCLUSIONS

Performance heterogeneity exists even among seemingly identical hardware and can significantly impact efficiency. The proposed heterogeneity-aware algorithms achieve up to 99.9% efficiency, delivering 4.4–11.5% improvements over traditional methods in intra-generation GPU scenarios and up to 300x gains in CPU-GPU heterogeneous systems. This work underscores the critical need to move beyond homogeneous scheduling assumptions in modern HPC environments.

FUTURE WORK

- Explore extensions to accommodate fixed node allocations.
- Analyze the impact of spatial locality on communication overhead using Space Filling Curves.
- Multi-Objective Optimization: Balance performance, energy, and communication.
- Multi-site validation (1000+ nodes)

ACKNOWLEDGMENTS

This work was supported by the Computing Sciences Summer Program at NERSC, LBNL.
Special thanks to Kevin, Rebecca, and Jessica.

Scan to access the heterogenous load balancer code and see more resources