OAK RIDGE National Laboratory

CUG 2024 | JERSEY CITY, NJ

Quantifying Message Aggregation Optimisations for Energy Savings in PGAS Models

Aaron Welch[1], Oscar Hernandez[1], Stephen Poole[2], Wendy Poole[2]

[1]Oak Ridge National Laboratory
[2]Los Alamos National Laboratory
oscar@ornl.gov, welchda@ornl.gov, swpoole@lanl.gov, wkpoole@lanl.gov

U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATELLE LLC FOR THE DEPARTMENT OF ENERGY

Our Mission, Should We Choose to Accept It

- The world has changed — scaling HPC is starting to grapple with limits and a looming power wall
- The network and data movement across it is particularly troublesome due to its relatively high cost and potential to bottleneck performance
- The partitioned global address space (PGAS) model offers a great way to apply a classic shared memory interface to large distributed systems
- Applications employing it can produce excessive numbers of small atomic, get, and put (AGP) messages, which scales poorly on modern networks
  - This can be a problem for dynamic and irregular applications (e.g. graph problems, data analytics, etc)
- What can we do about it?

Message Aggregation

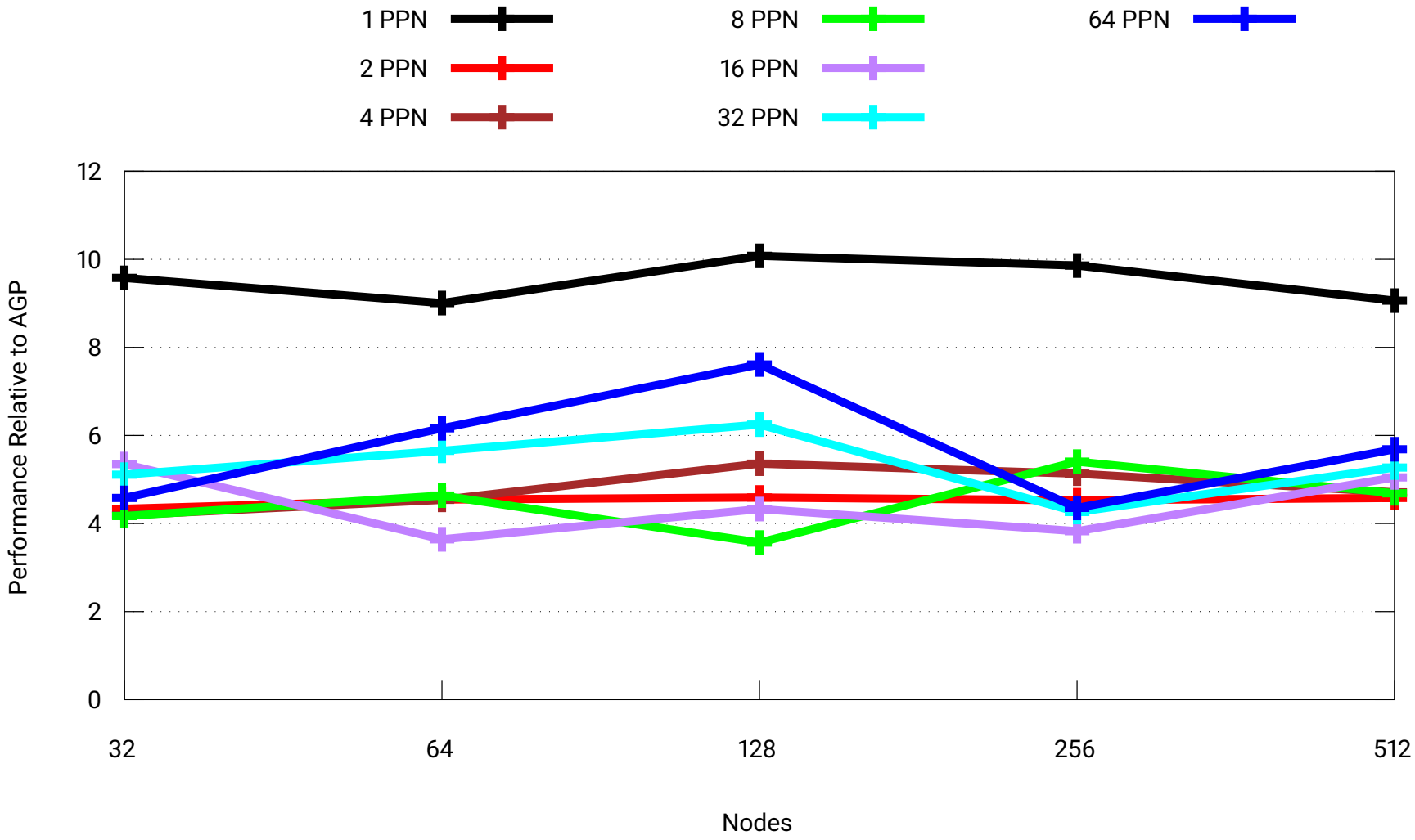
- The bale effort[1] identifies key challenges facing the small message problem and demonstrates methods that can be used to address them
- It provides a series of applications exhibiting key irregular access patterns, aggregation libraries including “conveyors” (our focus), and implementations of these patterns with and without aggregation
- Conveyors are like two-sided stateful message queues with operations to `push` and `pull` messages to/from them
  - Can they help us solve all of life’s problems?

[1] https://github.com/jdevinney/bale

Gotta Go Fast!

- Can it make my jobs run faster?
- Can its performance improvements scale?
- We compared the conveyor version of histogram against a non-aggregated base
- Testing was performed on ORNL's Frontier using 1–64 PPN across up to 8192 nodes
  - 128 nodes per cabinet, each with one AMD Epyc 7713 CPU and 4-port HPE Slingshot NICs (800 Gbps total)
- Testing used weak scaling with OpenSHMEM 1.4 from OpenMPI 5.0.3

Scaling Conveyors: Speedup

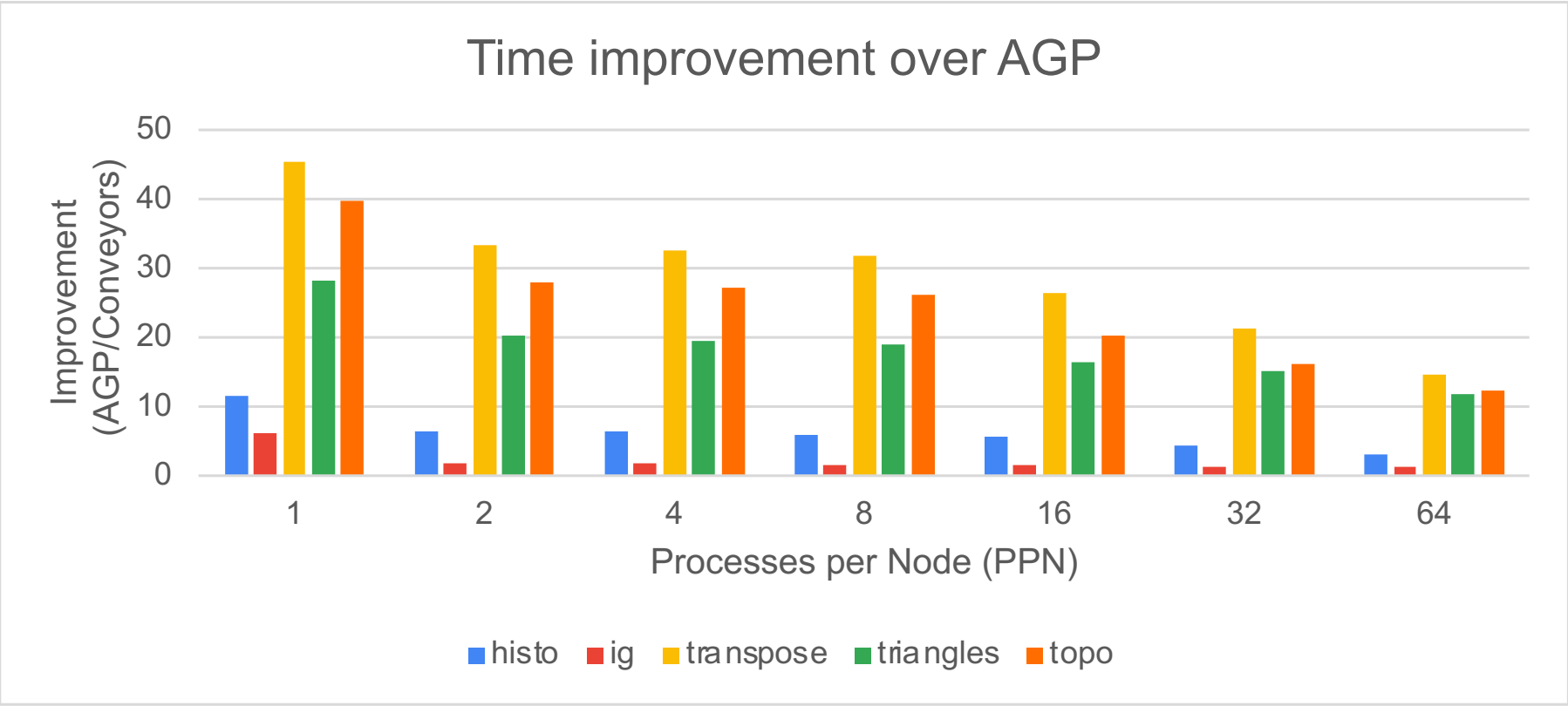


Histogram Speedup

Network Go Vroom!

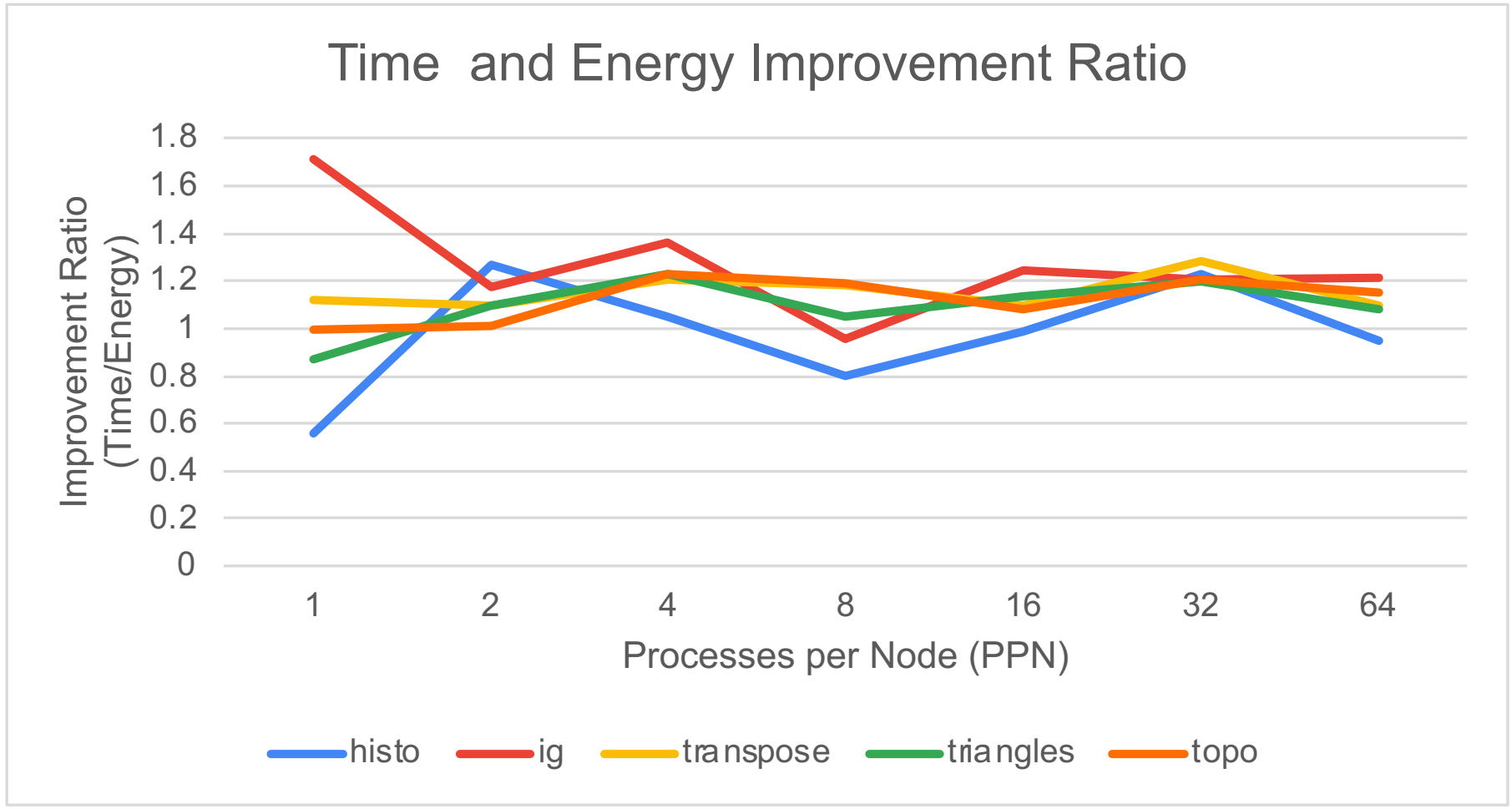
- We can easily see that aggregation with conveyors can yield massive improvements to execution time
- What about its impacts to power/energy?
- What about other applications?
- We added four more applications from the bale suite and compared improvements to time and energy using two Frontier nodes
  - Indexgather
  - Sparse Matrix Transpose
  - Triangle Counting
  - Topological Sort

Scaling Conveyors: Speedup



Relative Speedup of Conveyors over AGP

Scaling Conveyors: Time vs Energy



Ratio of Time/Energy Improvement of Conveyors over AGP

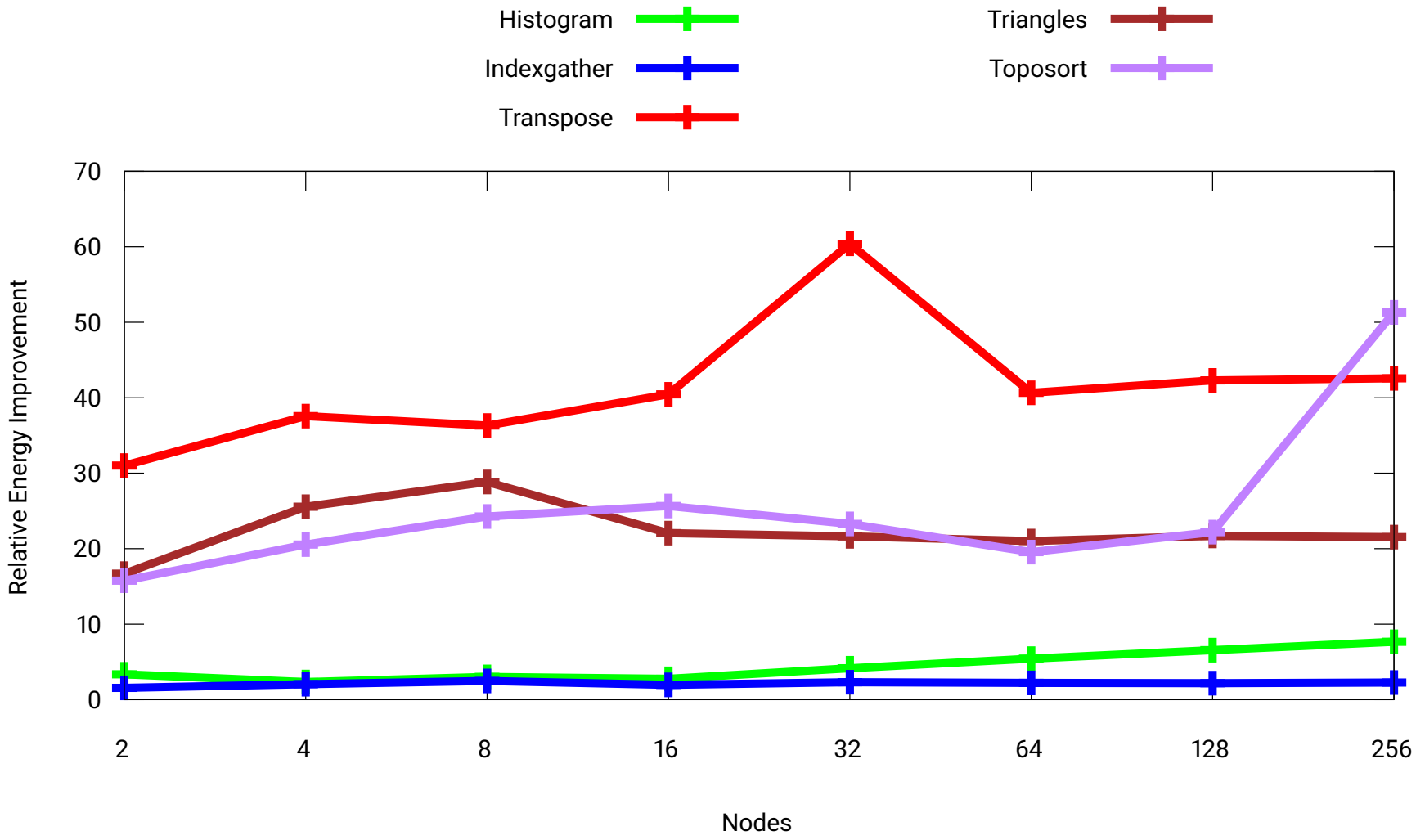
This Is Not What You Advertised!

- Wait! We'll use more nodes and look deeper!
- We'll focus our analysis on our primary concern — energy reduction/efficiency
  - *Where* do changes to performance come from?
  - What is the impact relative to idle energy use?
- We instrumented all five of the aforementioned bale applications to collect Cray Power Management (PM) counters with PAPI
- Each test will be run using the same configuration/input on both AGP/conveyor versions to measure relative improvement gained from using conveyors

Experimental Setup

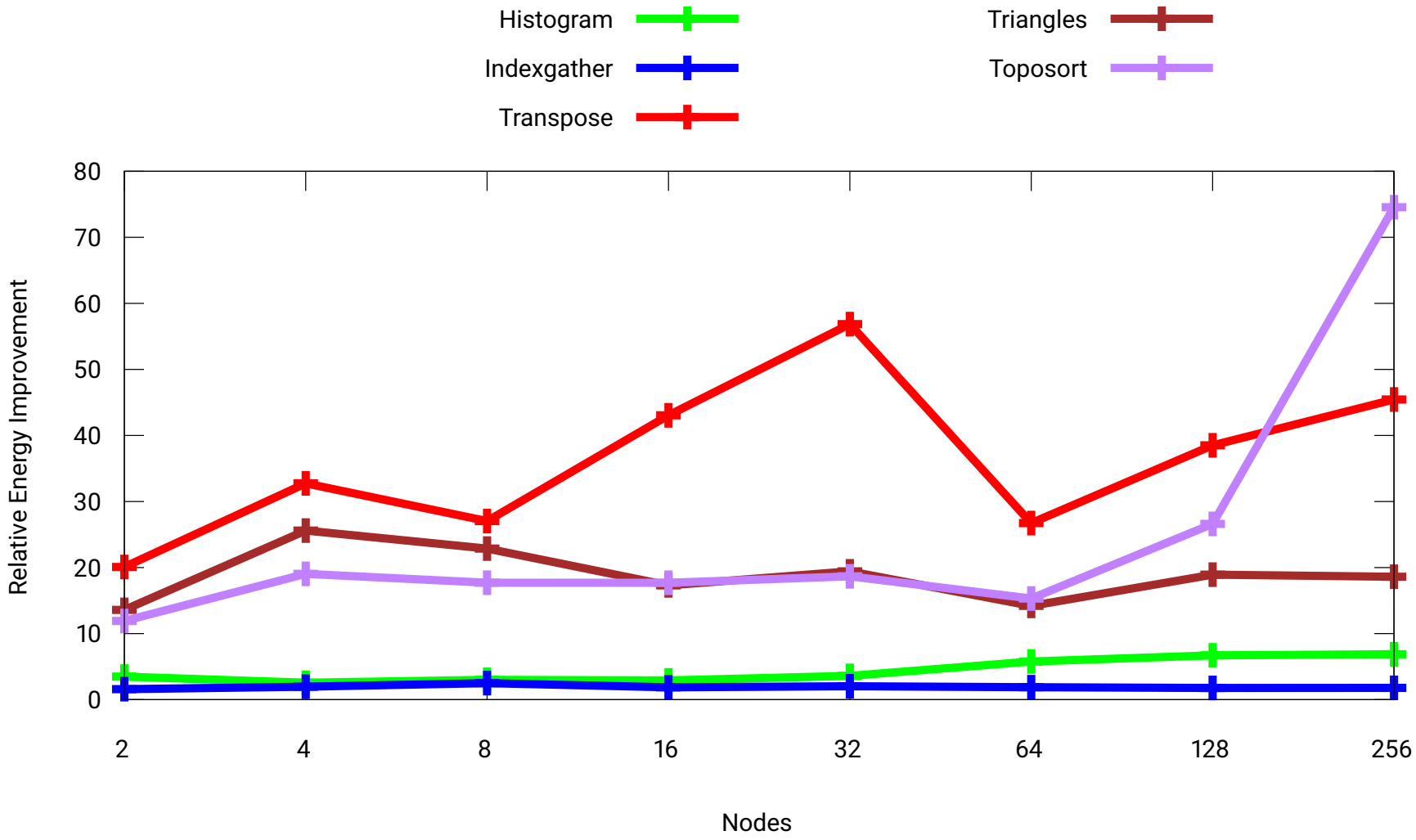
- We used 2–64 processes per node (PPN) on 2–256 nodes of ORNL's Frontier
- Measurements were taken based on application progress checkpoints rather than time for direct comparison purposes
- Prior to each set of measurements, we measured the same counters while the application slept to determine a baseline for the idle profile
  - We will use this profile to eliminate idle energy considerations to compare dynamic energy use
- All results will be shown with respect to conveyors' improvements relative to AGP

Energy Improvement



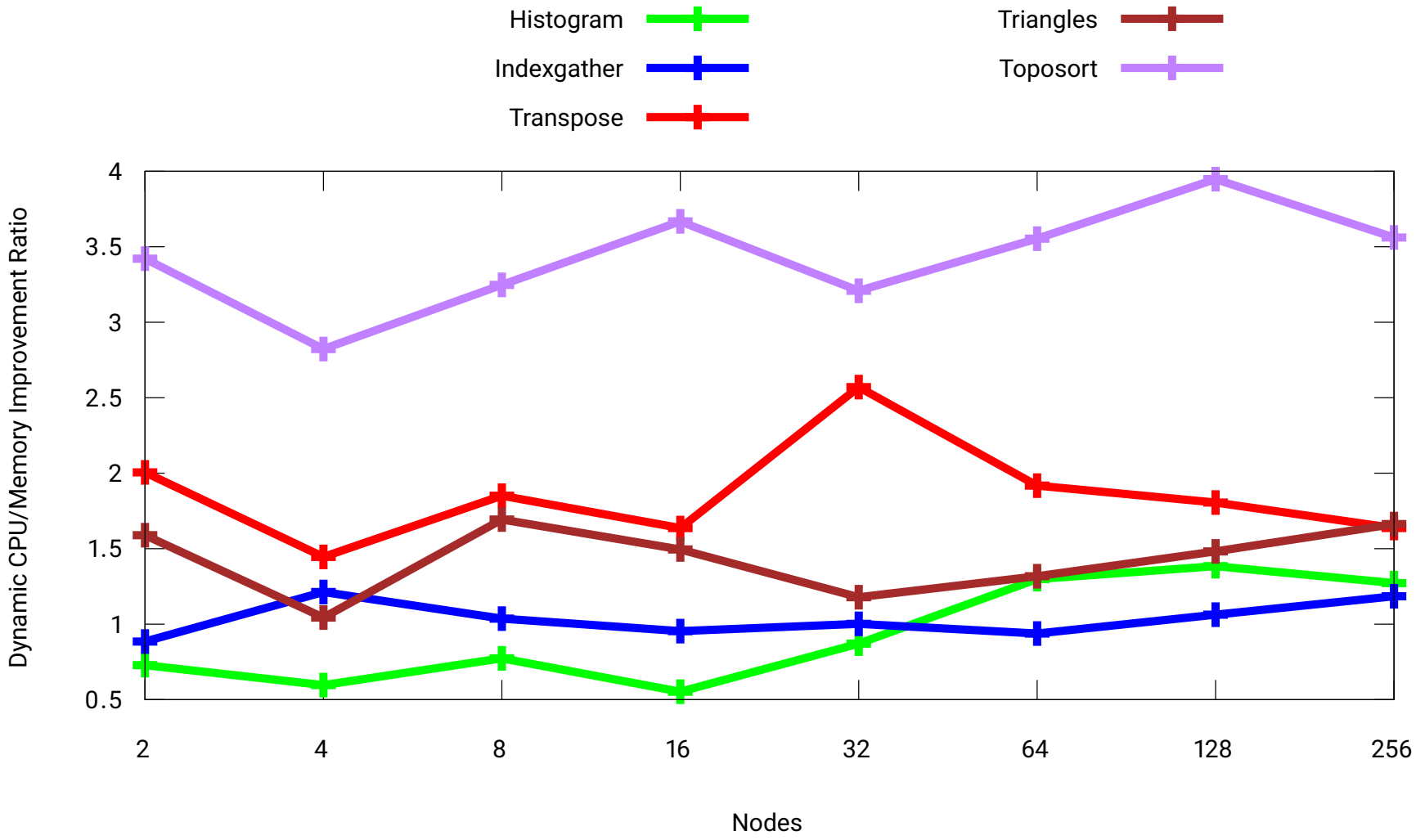
Energy Improvement

Dynamic Energy Improvement



Dynamic Energy Improvement

Energy Improvement Ratios

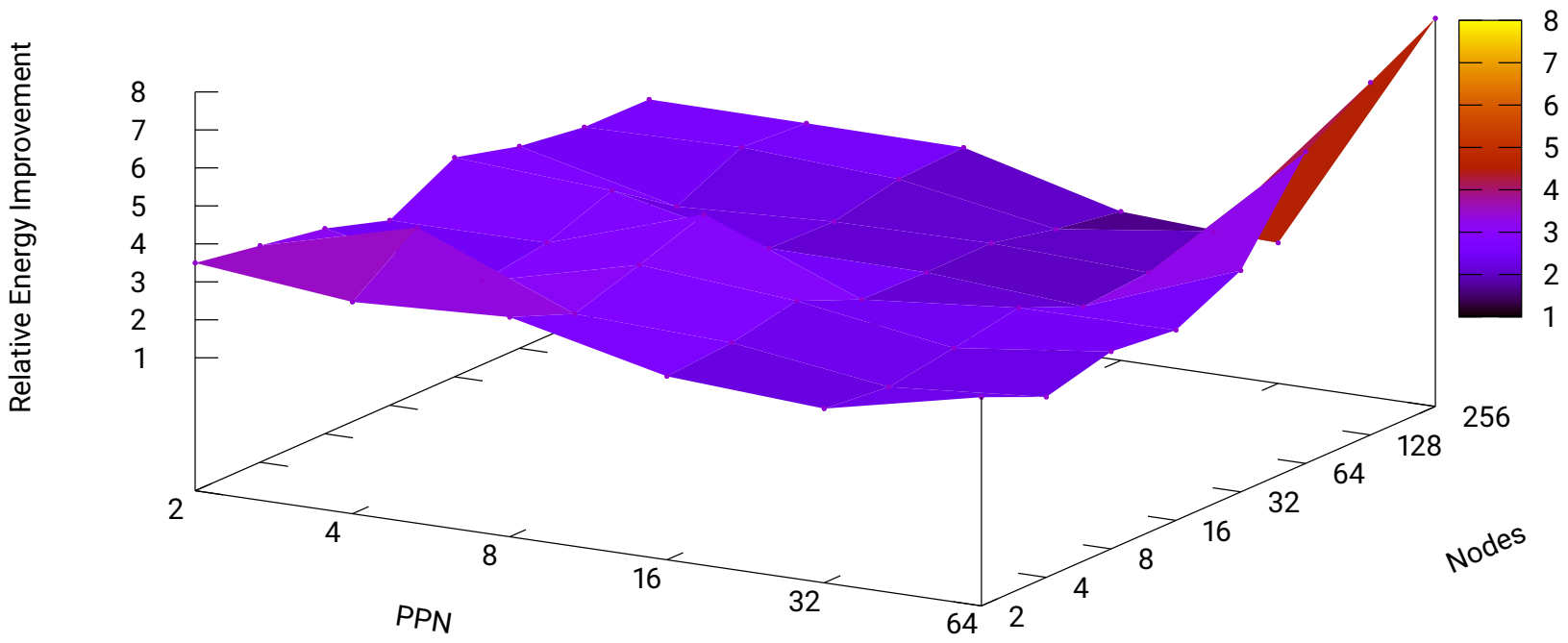


Dynamic CPU/Memory Energy Improvement Ratio

Histogram

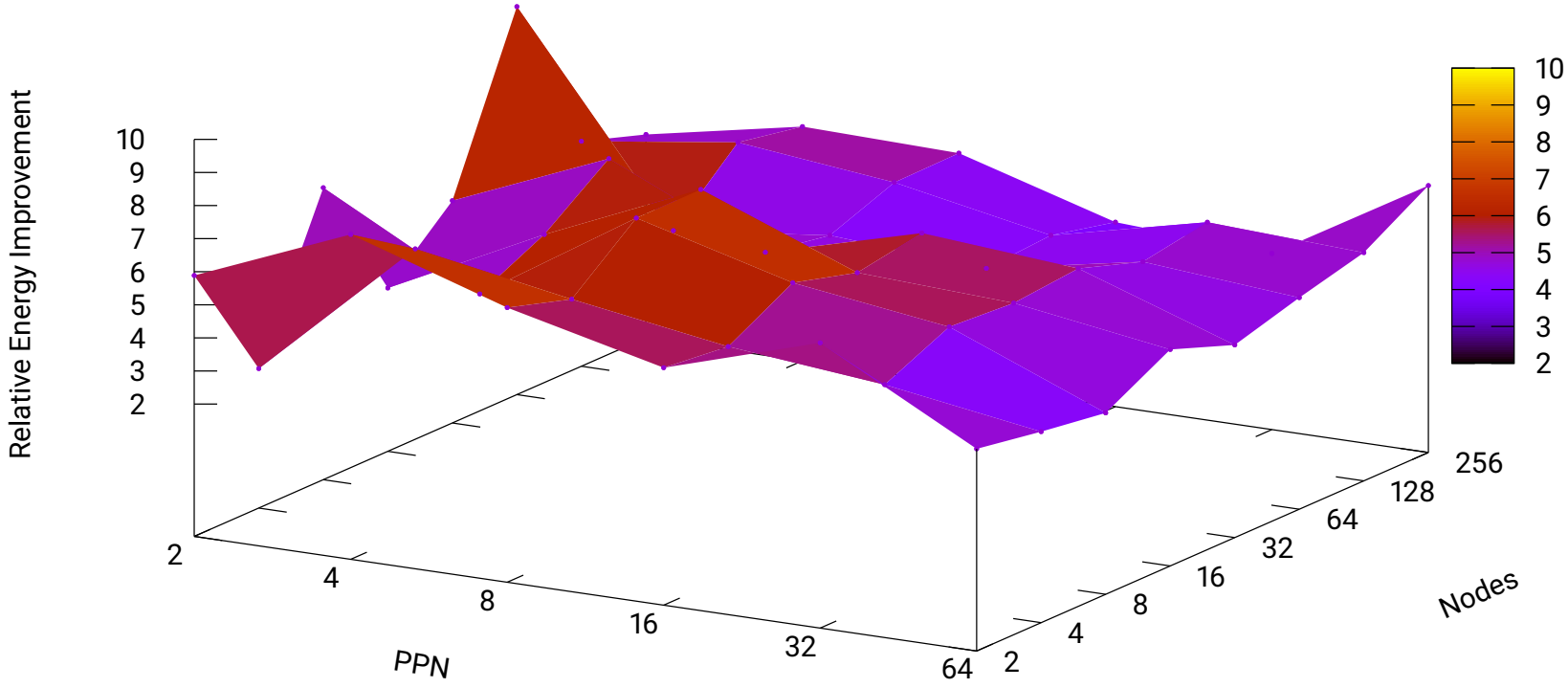
- Represents a communication pattern where processing elements (PEs) are asynchronously sending lots of small updates to random locations on other PEs
- Each PE generates a uniform list of random indices in a distributed table and updates each index by incrementing its value
- Provides the simplest and most direct benchmark of the aggregation strategy's performance potential

Results: Histogram



Histogram Dynamic CPU Energy Improvement

Results: Histogram

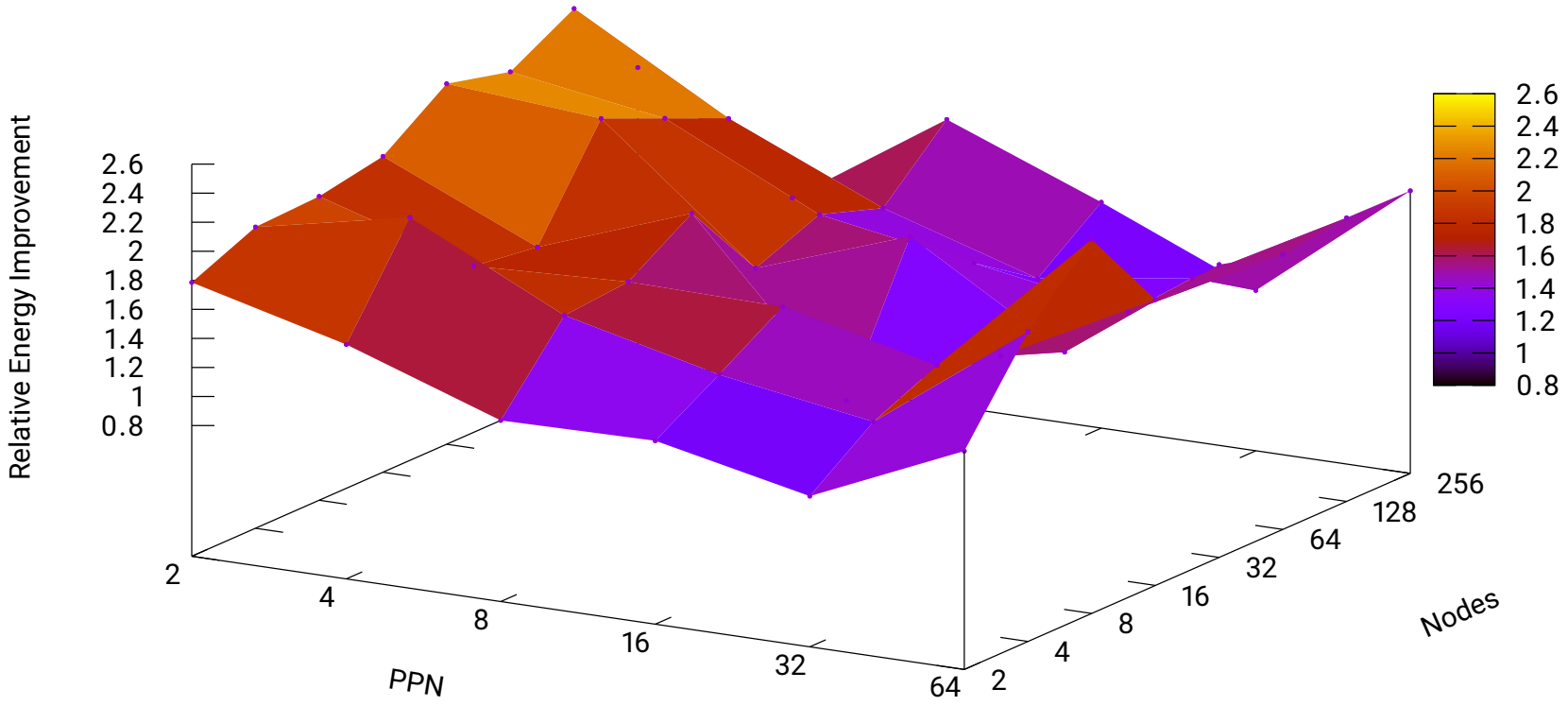


Histogram Dynamic Memory Energy Improvement

Indexgather

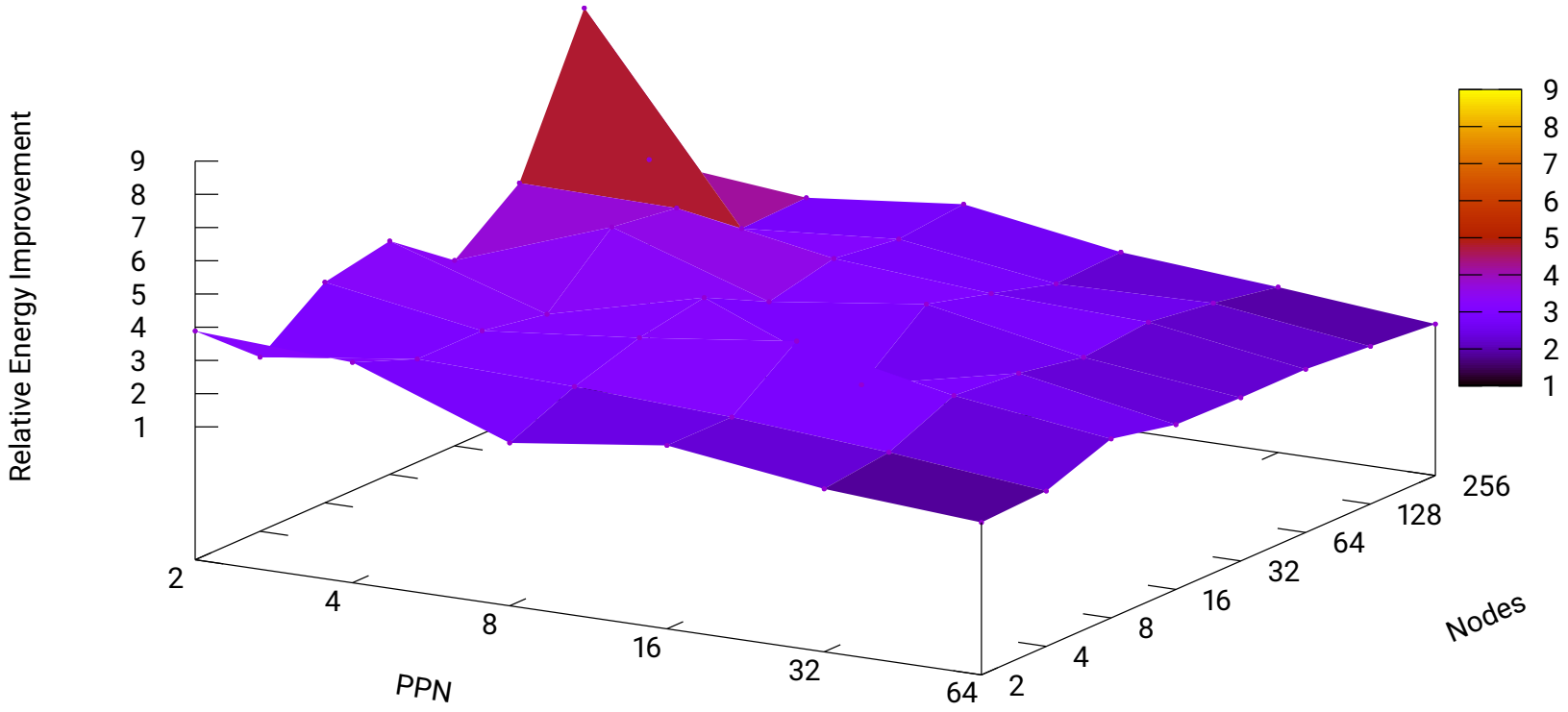
- Similar to histogram, indexgather represents a pattern involving asynchronous reading or random elements from a distributed array
- Each PE generates a list of random elements and then performs get operations on them back to back
- Bale version for AGP was modified to use a single-element version of the newer non-blocking gets for better comparison

Results: Indexgather



Indexgather Dynamic CPU Energy Improvement

Results: Indexgather

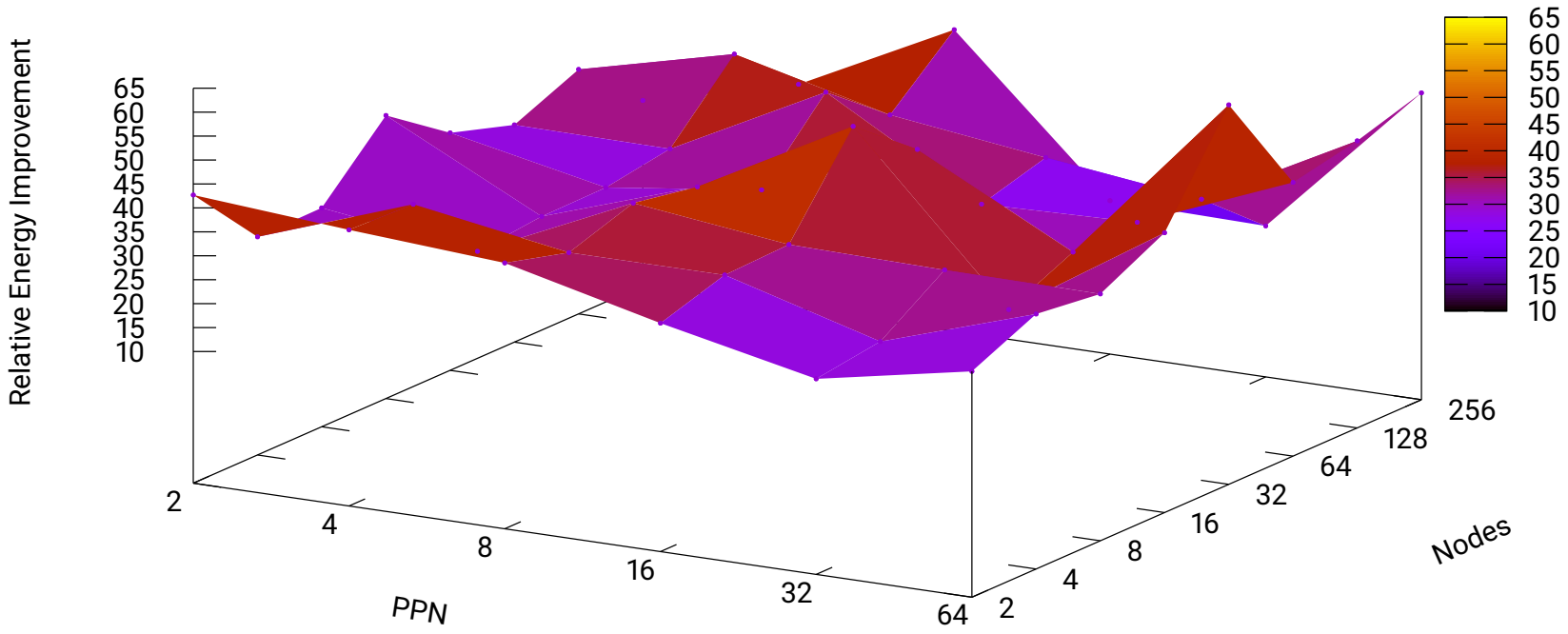


Indexgather Dynamic Memory Energy Improvement

Sparse Matrix Transpose

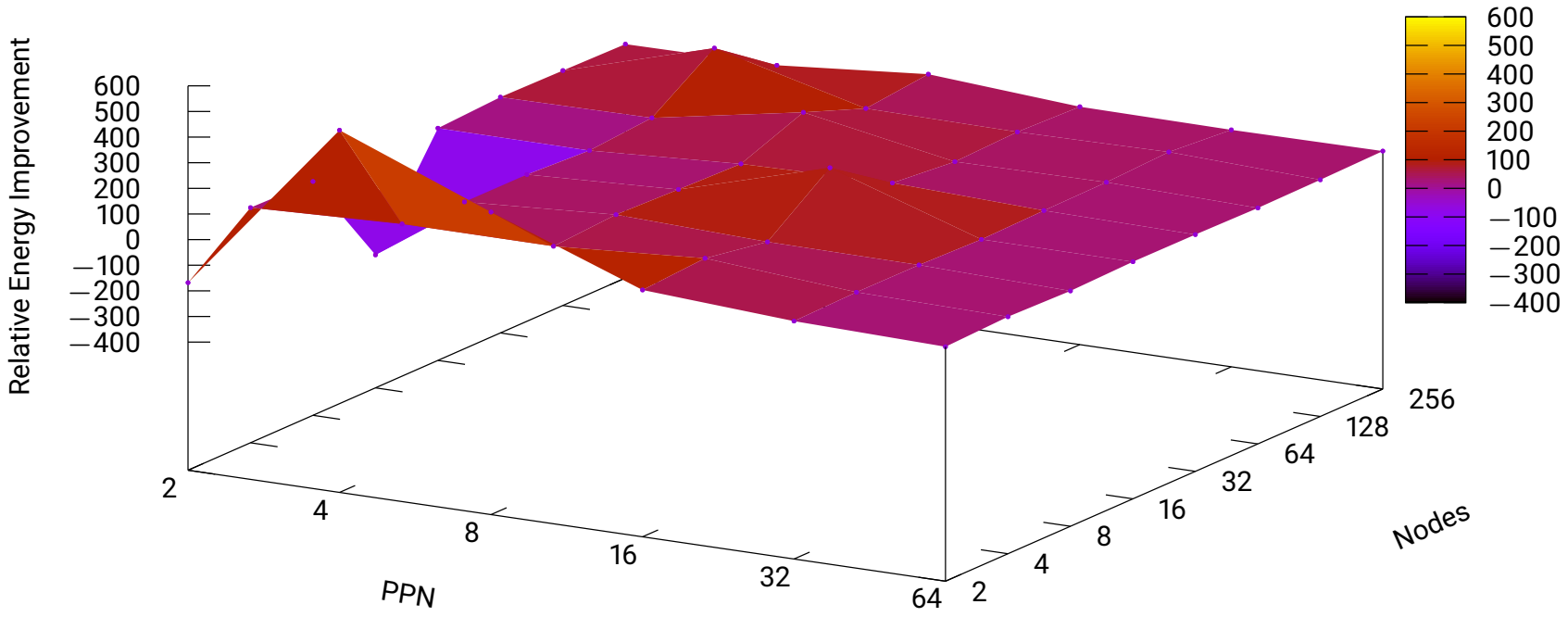
- Computes the transpose of a sparse matrix in compressed sparse row (CSR) format
- Difficult to aggregate due to synchronisation as PEs need when determing locations for writing data
- First phase uses a histogram pattern to determine the number of nonzeroes in each column
- Second phase acquires the new location for each value, then writes it to the destination PE

Results: Sparse Matrix Transpose



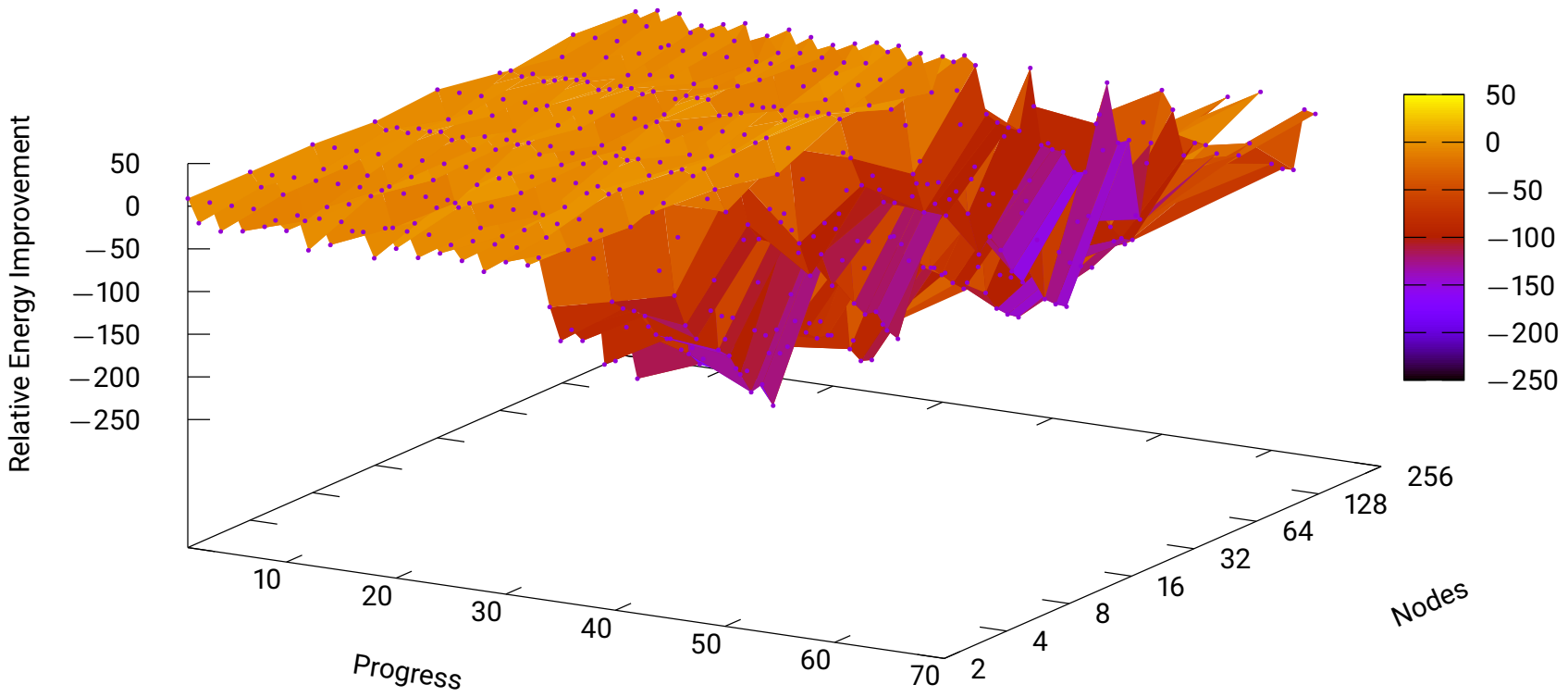
Transpose Dynamic CPU Energy Improvement

Results: Sparse Matrix Transpose



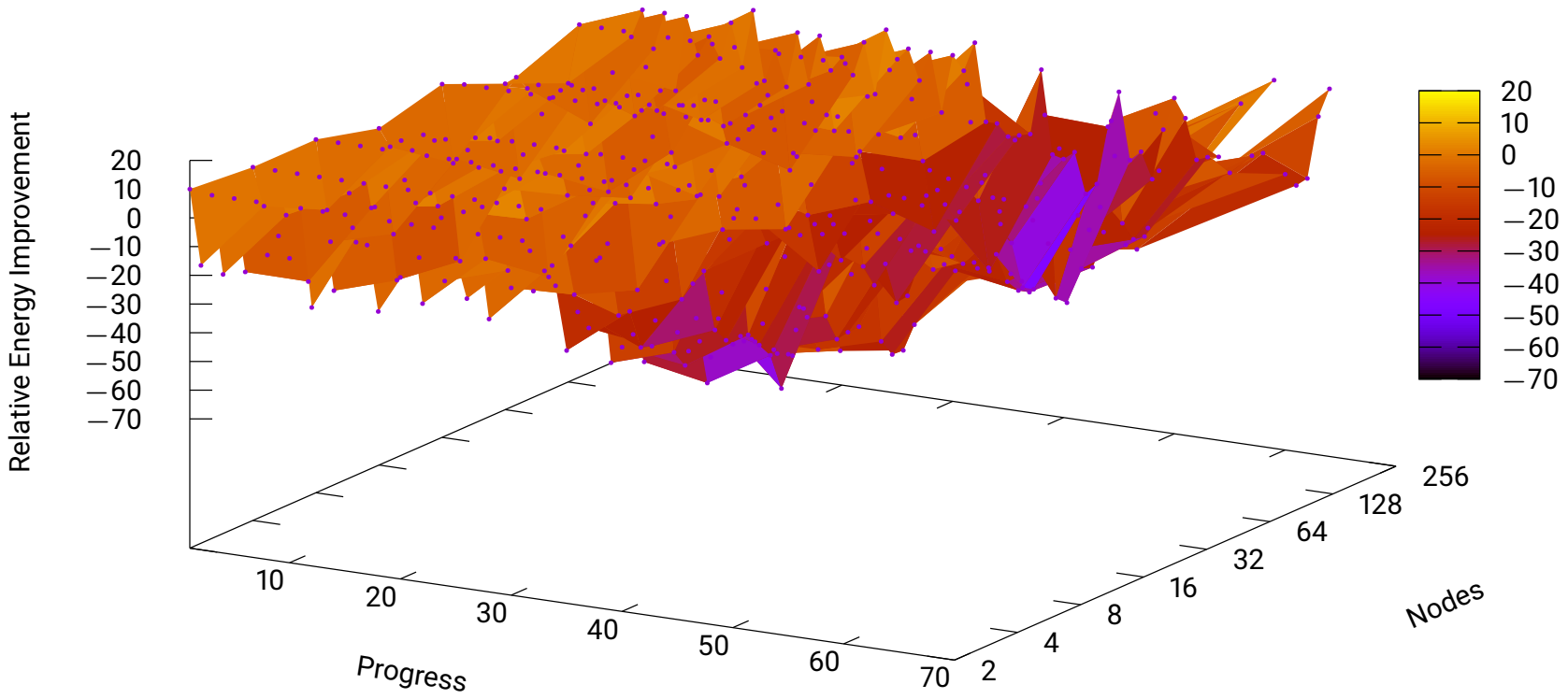
Transpose Dynamic Memory Energy Improvement

Results: Sparse Matrix Transpose



Transpose Dynamic CPU Energy Improvement Over Time

Results: Sparse Matrix Transpose



Transpose Dynamic Memory Energy Improvement Over Time

Triangle Counting

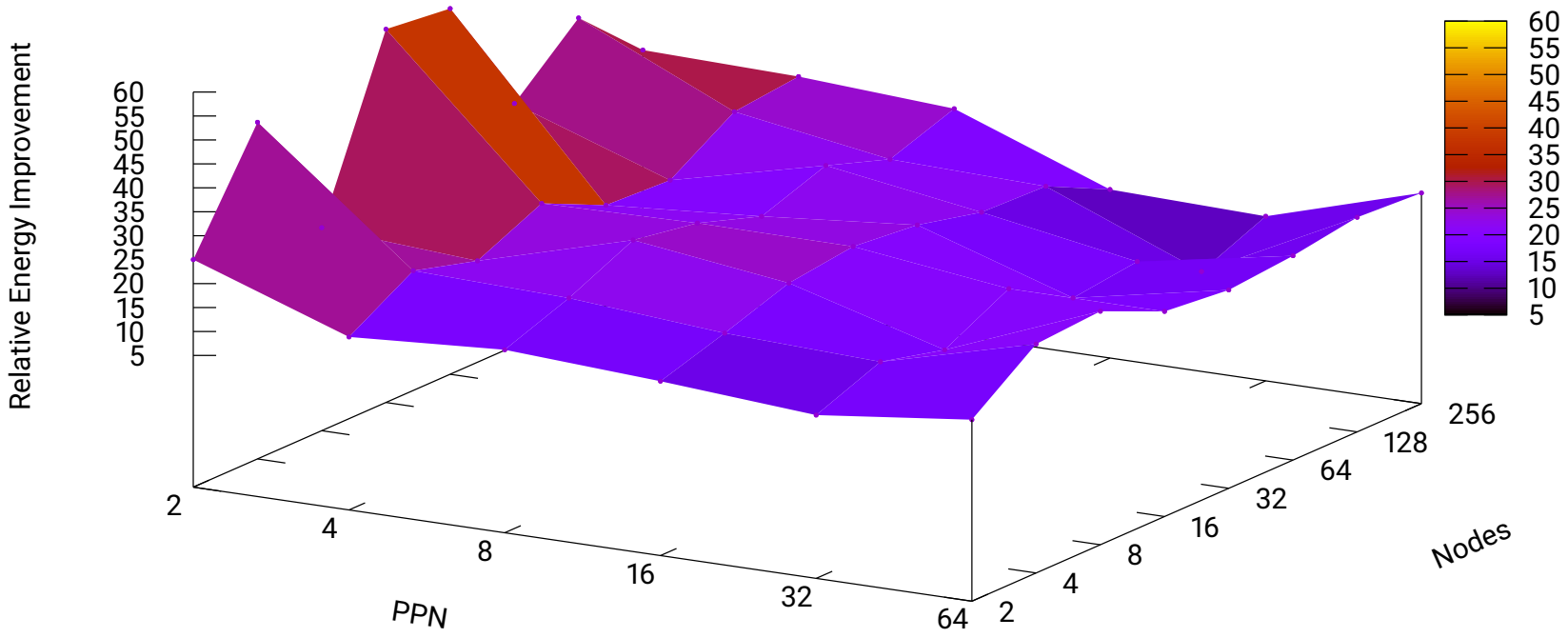
- Counts triangles in undirected graphs by computing either $L \odot L \times U$ or $L \odot U \times L$, where L and U are the lower/upper triangular matrices of sparse input

- Two ways to implement its algorithm:

  - PEs put their nonzeroes to remote rows (maps well to aggregation)

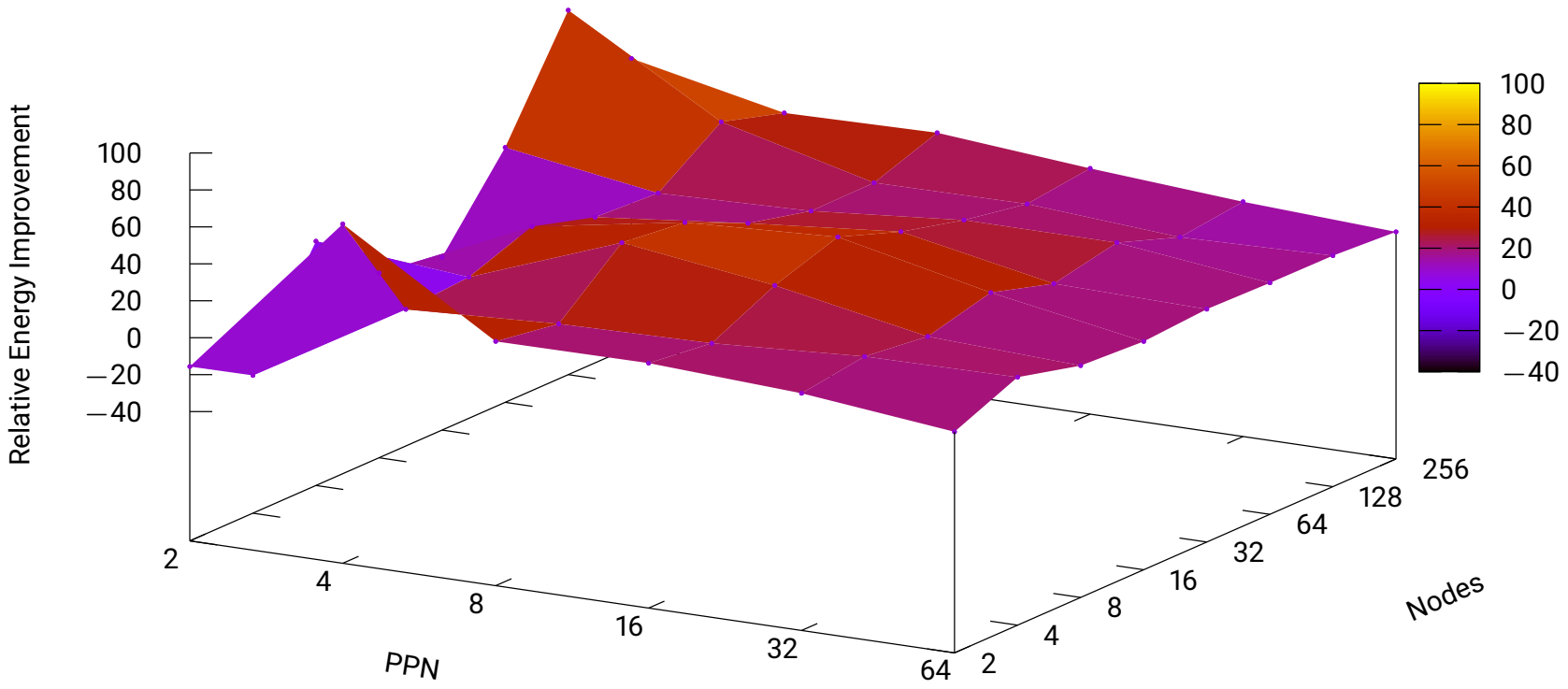
  - PEs get their data from other PEs themselves (maps well to AGP)

Results: Triangle Counting



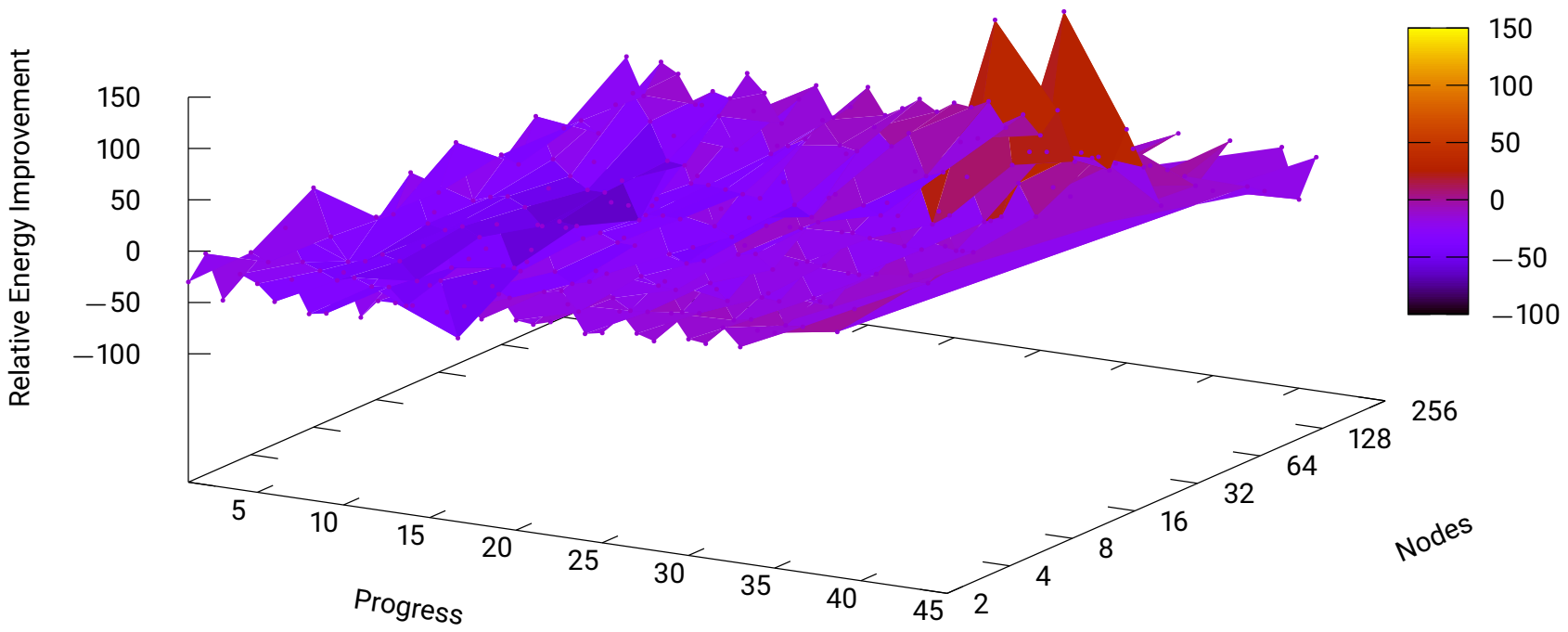
Triangle Counting Dynamic CPU Energy Improvement

Results: Triangle Counting



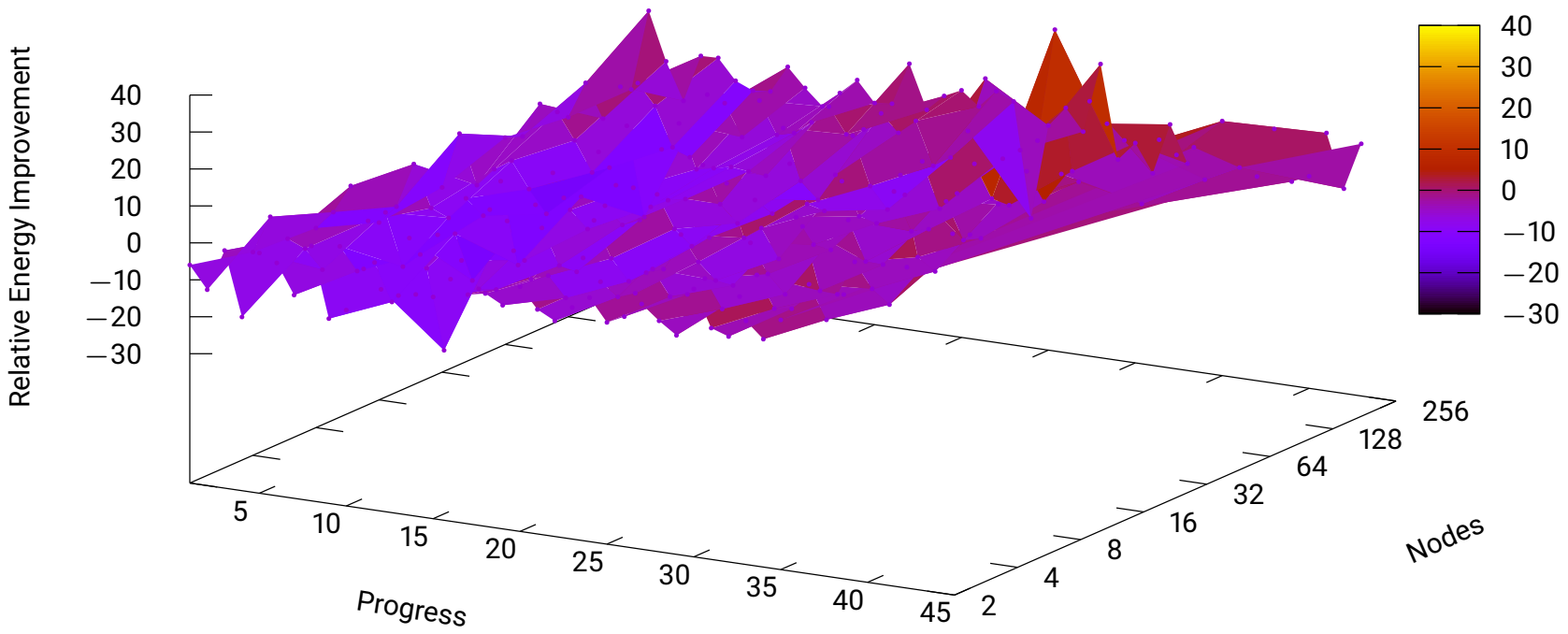
Triangle Counting Dynamic Memory Energy Improvement

Results: Triangle Counting



Triangle Counting Dynamic CPU Energy Improvement Over Time

Results: Triangle Counting

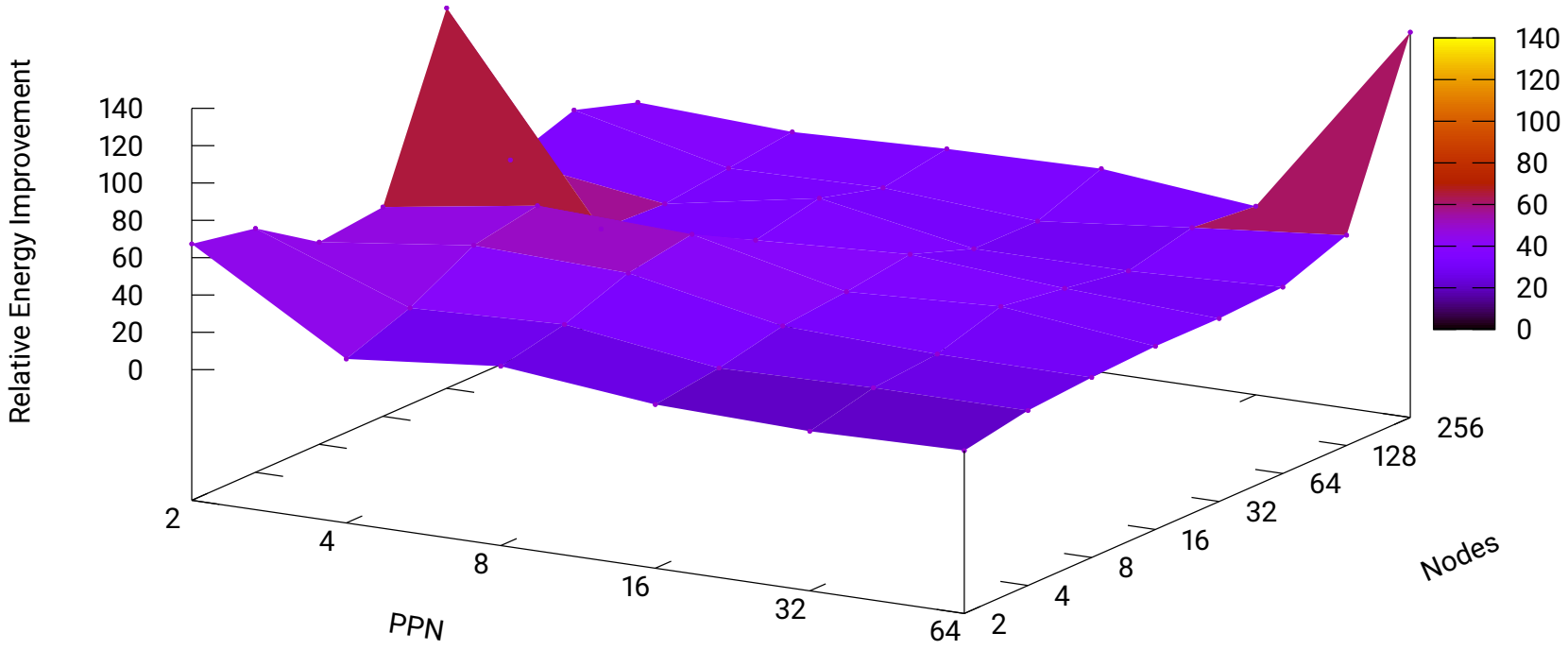


Triangle Counting Dynamic Memory Energy Improvement Over Time

Topological Sort

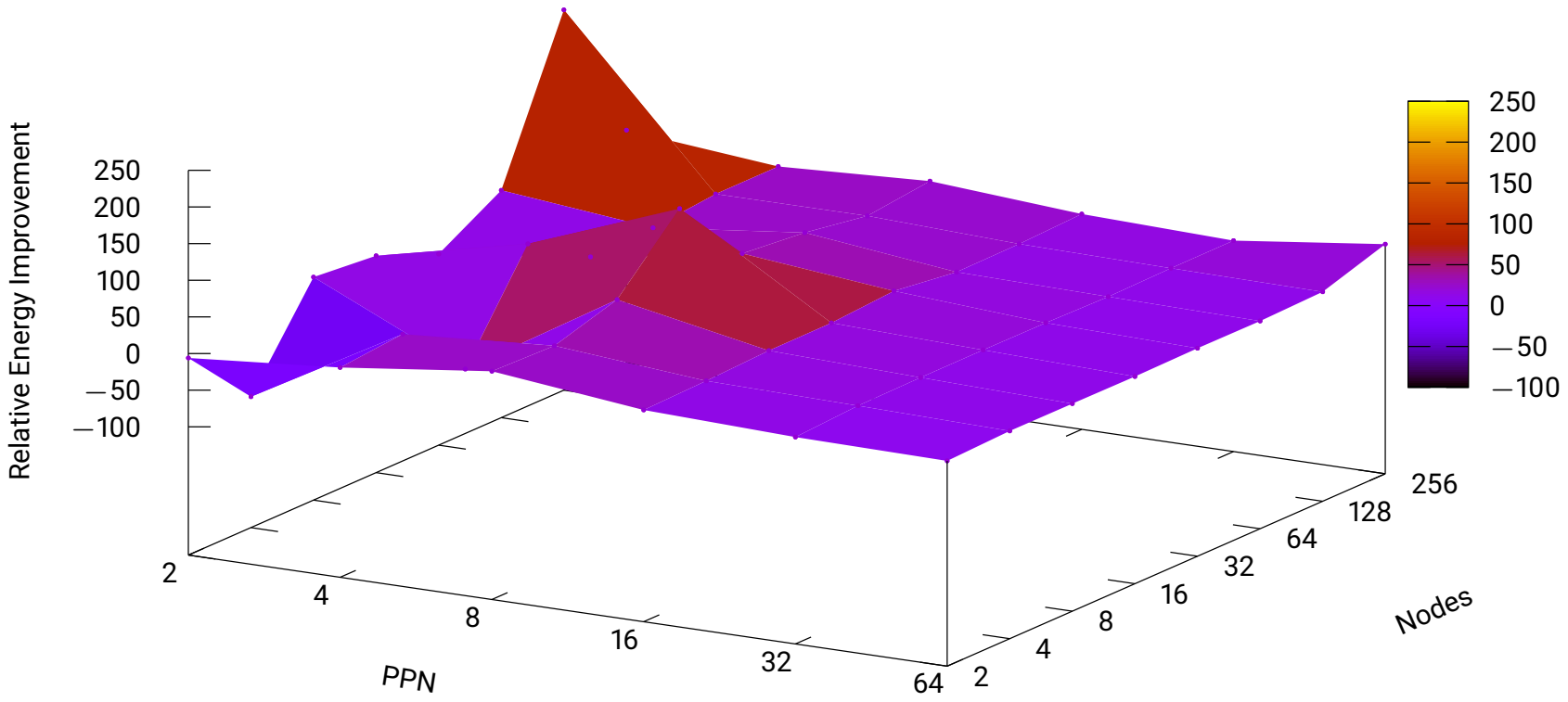
- Searches for a linear ordering of the vertices in a directed graph such that every vertex comes before any other that has an edge leading to it
- Culmination of several of the pattens in the bale suite
- Not completely latency-tolerant
  - Works through many iterations of a work queue that grows as the graph is processed
  - Searches for all vertices of degree 1, executes work loop on them, then removes them and their edges

Results: Topological Sort



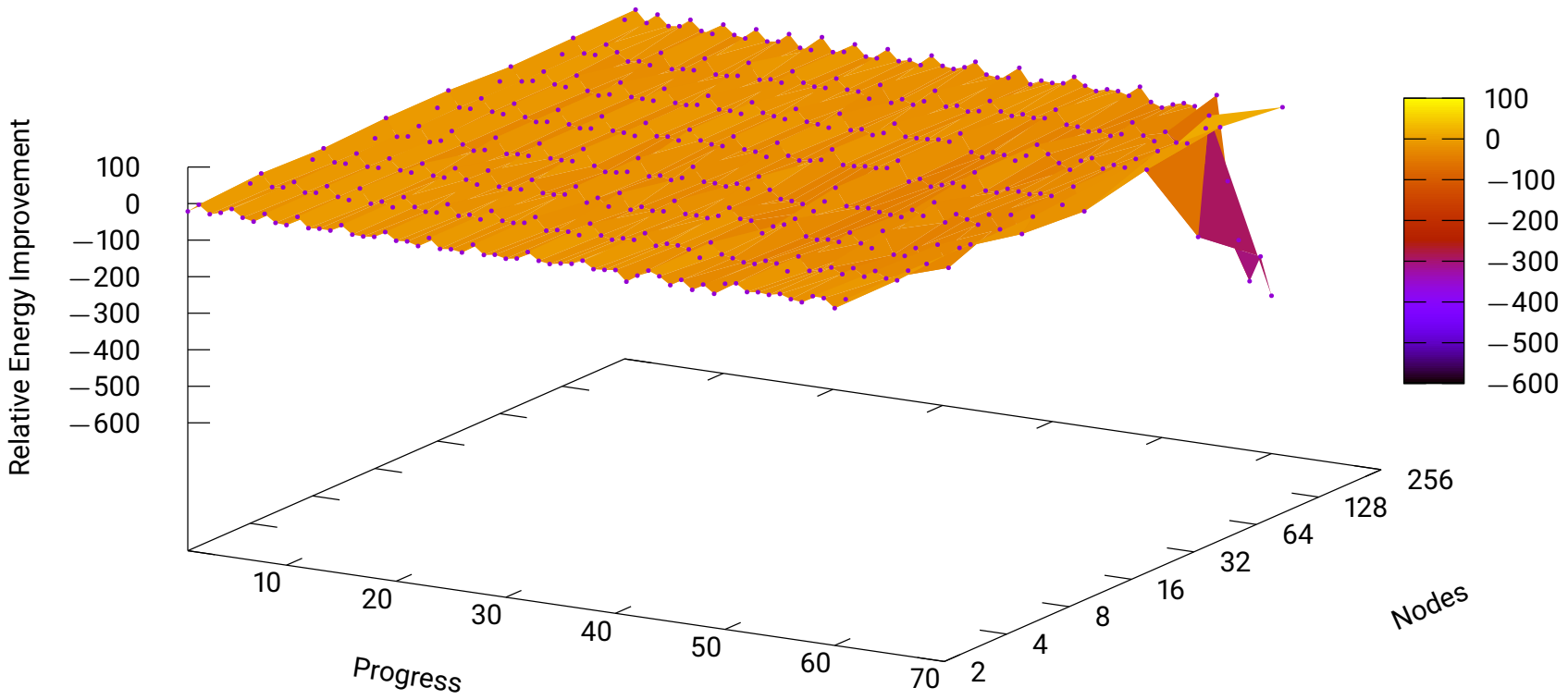
Topological Sort Dynamic CPU Energy Improvement

Results: Topological Sort



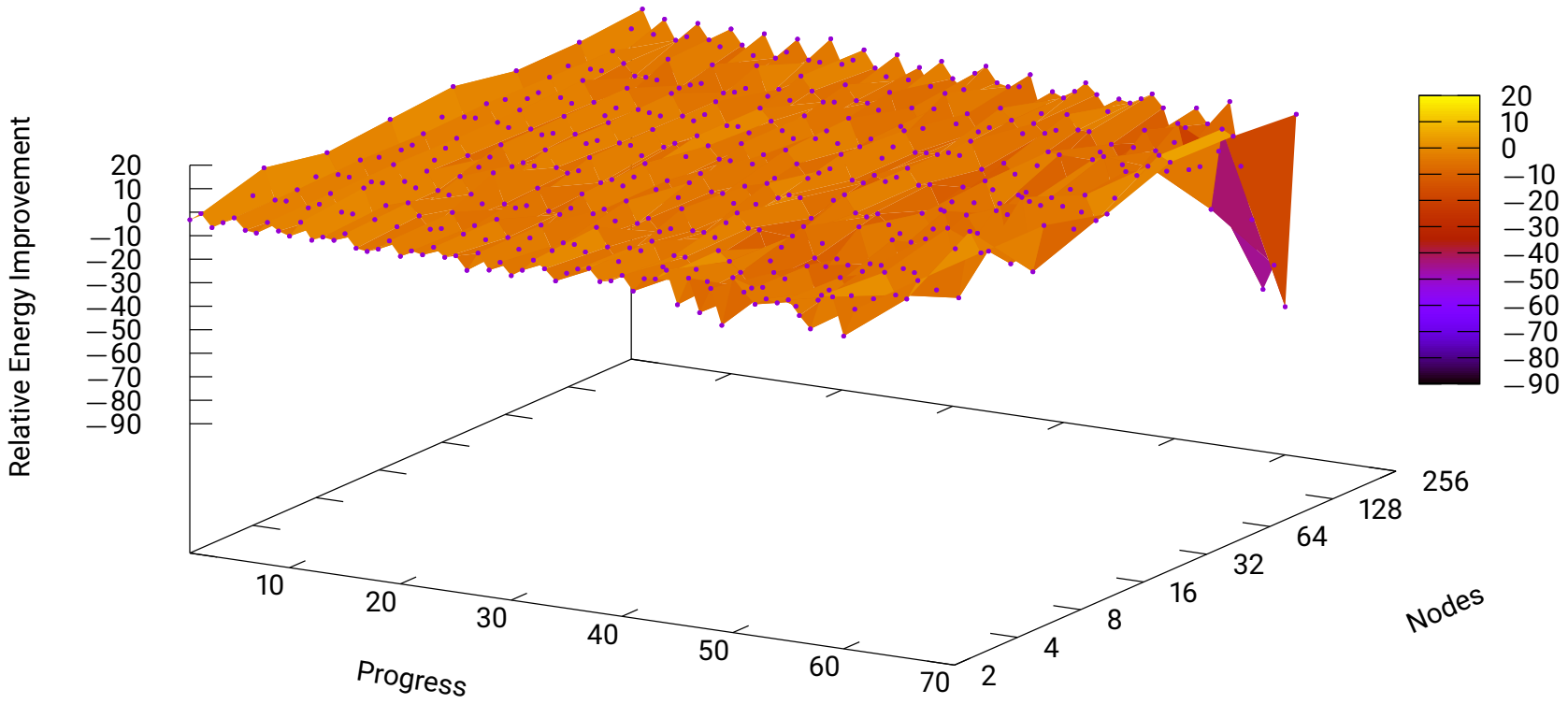
Topological Sort Dynamic Memory Energy Improvement

Results: Topological Sort



Topological Sort Dynamic CPU Energy Improvement Over Time

Results: Topological Sort



Topological Sort Dynamic Memory Energy Improvement Over Time

Conclusions

- We can clearly see that conveyor aggregation provides dramatic improvements to both execution time and energy consumption
- A big question that remains is whether additional improvements could be made through good use of offloading requests to dedicated/cheaper devices
- Future work could investigate portability of results to other platforms/networks

Questions?