Quantifying Message Aggregation Optimisations for Energy Savings in PGAS Models

Aaron Welch
Oscar Hernandez
welchda@ornl.gov
oscar@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Stephen Poole
Wendy Poole
swpoole@lanl.gov
wkpoole@lanl.gov
Los Alamos National Laboratory
Los Alamos, New Mexico, USA

Abstract

Upon breaking past the exascale barrier, HPC systems are facing their greatest challenge yet — a power wall that must be addressed through new methods in both hardware and software. While energy costs are becoming a major issue at all levels, of particular concern is that of the network, as the relative cost of moving data is increasing faster than ever. The partitioned global address space (PGAS) model is critical within certain HPC domains, but is known to suffer from the small message problem, where irregular many-to-many access patterns result in congesting the network with excessive numbers of small messages. To address this, the conveyor aggregation library was developed to defer individual messages and group them for subsequent bulk processing. In this paper, we investigate its impact on energy use related to the network, with a focus on the Slingshot 11 interconnect. We will demonstrate that this strategy is not only highly performant, but also crucial to reducing energy footprints to remain within target power envelopes.

Keywords

OpenSHMEM, conveyors, network aggregation, energy

ACM Reference Format:
Aaron Welch, Oscar Hernandez, Stephen Poole, and Wendy Poole. 2025. Quantifying Message Aggregation Optimisations for Energy Savings in PGAS Models. In *Proceedings of Cray User Group 2025 (CUG '25).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

As HPC grapples with the limits of scaling down and the performance gains per Watt continue to shrink, we must face the new reality of energy efficiency becoming the primary determinant of scalability. One of the major recurring themes toward this end is of reducing the overheads associated with data movement [Shalf 2019]. Transfer of data across the wider network can be notoriously expensive in relation to intranode costs, which can be exacerbated by certain software models or algorithms.

The PGAS memory/communication model is popular within some circles for its ability to naturally express a more familiar and intuitive shared-memory style of programming across the network, as well as exposing locality for effective memory management [Kestor et al. 2013]. OpenSHMEM is a library specification that provides such a model, and is known for being lightweight but powerful, providing a clean and effective mapping to remote direct memory access (RDMA) hardware. However, it exhibits the same issues that the larger PGAS community faces with regard to certain kinds of memory access patterns that can drag network performance down. Specifically, applications which produce large numbers of small, irregular memory accesses can quickly overwhelm networks with too many of these small messages, incurring slowdown and decreased resource efficiency as pipelines stall while waiting for results. Unfortunately, patterns like these can be crucial for many classes of application, such as for graph processing or data analytics.

The typical solution to addressing the performance penalties inflicted by this small message problem is to employ a message aggregation strategy, sacrificing latency when possible to build up larger collections of messages to eventually process in bulk. The bale effort [Maley and DeVinney 2019] is well known for providing this feature to applications through its "conveyor" library, by providing stateful queues and a contract for using them that assures their efficient operation. While its impact on application runtimes has been well documented, its energy efficiency is decidedly less so. Although some preliminary studies have been carried out comparing speedup against energy improvment between a pair of nodes [Hernandez et al. 2024], far more could be done to investigate its effects both deeper and at a much larger scale.

Therein lies the goal of this work — to study its impact on energy efficiency for irregular applications at a larger scale, specifically across a Slingshot 11 interconnect. While previous works have studied other aspects of performance on Slingshot networks [De Sensi et al. 2020], including in relation to OpenSHMEM specifically [Namashivayam et al. [n. d.]], we have not seen any study focused on energy concerns at the level of detail we think is essential in this area. To this end, we will test five applications from the bale suite on ORNL's Frontier supercomputer, comparing implementations using conveyors against those using OpenSHMEM atomic, get, and put (AGP) operations. We will focus primarily on energy improvements possible through the use of conveyors, and break our findings down to their component pieces to not just determine how they affect performance, but also why.

CUG '25, Jersey City, NJ

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Section 2 provides a detailed overview of past work, and addresses its strengths as well as the shortcomings that merited this work. The bulk of this paper's contribution can be found in Secion 3, where we provide a thorough analysis of our experimental results. Our concluding remarks and suggestions for future work can be found in Section 4.

2 Past Work

The performance benefits and scalability of conveyor aggregation was previously explored in [Welch et al. 2024]. In this work, we looked specifically at bale's histogram application, which was chosen for being the most direct test for conveyors' performance with respect to execution time. In histogram, processing elements (PEs) repeatedly send many small, independent messages to random locations in the network. Similarly to this paper, these tests were also carried out on ORNL's Frontier, and for some test cases ran up as high as 8,192 nodes to determine if its performance benefits could continue to scale that high. Figure 1 shows some of those results, comparing the relative speedup in execution time for a conveyor implementation of histogram oer the AGP from 32–512 nodes using 1–64 PEs to determine if its scalability behaviour changed, which it didn't.

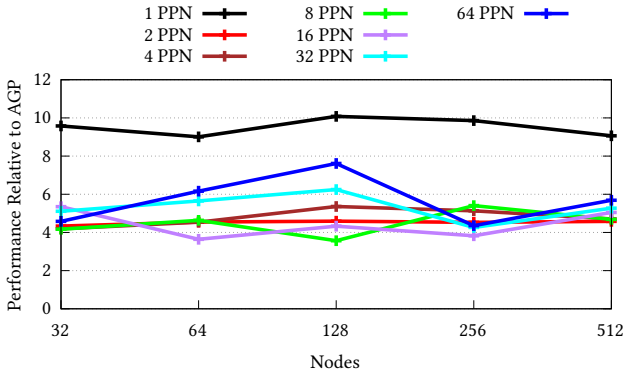

Figure 1: Histogram Speedup

After conveyor's ability to scalably optimise execution time was proven, we looked at the relationship between this speedup and the corresponding energy improvement [Hernandez et al. 2024]. This testing was also done on Frontier, although expanded to use five applications from the bale suite — histogram, indexgather, sparse matrix transpose, triangle counting, and topological sort. Although only using two PEs, the same kind of testing as in the previous work was applied to compare execution speedup of the conveyor implementations compared to the corresponding OpenSHMEM equivalents, as seen in Figure 2. Energy measurements were then taken so that the relative improvements there could also be compared to that of the speedup, as seen in Figure 3, where values above 1 represent execution time improving faster than energy, and values below 1 are the opposite.. Here, we saw that there were same apparent patterns in the shifting of the tradeoff between time and energy as a factor of the processes per node (PPN), though given the small size of the experiment, the results drawn from them can't be conclusively extrapolated to larger scales.

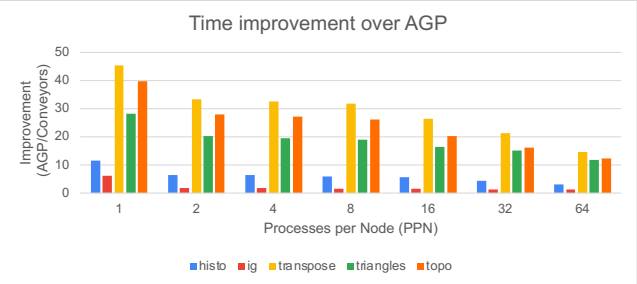

Figure 2: Relative Speedup of Conveyors over AGP

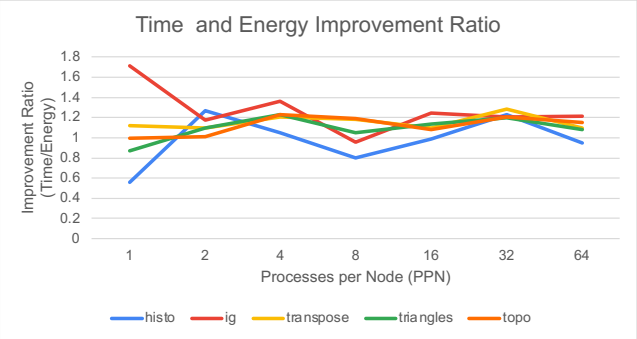

Figure 3: Ratio of Time/Energy Improvement of Conveyors over AGP

3 Evaluation

Our primary concern in this work is that of energy reduction and efficiency, so that is where we will focus our analysis. There are two major factors we want to capture in our experiments:

- Distinctions between *where* changes in the performance profile come from (e.g., consideration of uncore resource utilisation)
- The impact of changes relative to the system's idle state

The second of these features particularly stands out as rather unique — what we are able to measure with this is how much additional energy cost is incurred specifically by the presence or lack of aggregation. Similar to our previous work, we will look at the energy concumption of five of the bale suite's applications — histogram, indexgather, sparse matrix transpose, triangle counting, and topological sort. Each test is ran with both non-aggregated AGP operations as well as conveyors, with performance measured through hardware counters collected with Performance Application Programming Interface (PAPI). We used counters from the Cray Power Management (PM) PAPI component to measure energy consumption at the CPU, memory, and node levels.

3.1 Experimental Setup

We performed our experiments using 2–64 PPN on 2–256 nodes of ORNL's Frontier supercomputer. Each node has a single AMD Epyc 7A53 2 GHz CPU and four HPE Slingshot 11 200 Gbit/s NICs. The same workload was used for each combination of application, PPN, and node count to ensure comparable results. Instrumentation was performed by hand by direct use of PAPI to both minimise overhead as well as allow for us to measure based on progress checkpoints (i.e., measure the energy required to perform the first n updates of histogram).

Prior to each set of measurements, we made the applications sleep for 30 seconds, obtaining measurements before and after to determine our idle consumption values. We then break each application into equal-sized chunks of work and proceed to measure the impact that the AGP/conveyor implementations have to the measured performance counters to complete each chunk. This allows to compare the two implementations' dynamic energy use (i.e., above idle) against each other to determine the relative improvement (e.g., a result of 1 would indicate equivalent results, whereas 2 would indicate that that the AGP version used twice as much energy compared to conveyors).

3.2 Results Overview

We will first look at the total raw energy use for each of the five applications on 2–256 nodes using 64 PPN, as seen in Figure 4. Consistent with our preliminary results from [Hernandez et al. 2024], despite the direct simplicity of histogram and indexgather's access patterns and now accounting for idle consumption, the remaining three applications are where the impact is felt the most. The overall pattern we see in raw energy improvement is also consistent (albeit slightly lessened) with that of the dynamic energy improvement, as seen in Figure 5.

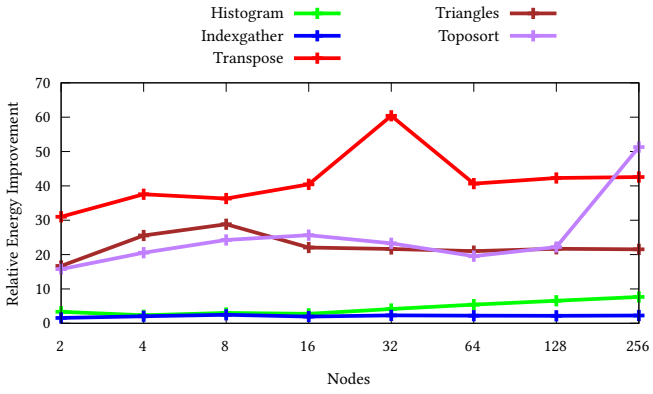

Figure 4: Energy Improvement

In fact, while not shown, the same general pattern appeared across all metrics, but where things get interesting is in comparing them against each other. Figure 6 shows they dynamic CPU energy improvement compared to that of the memory, with values above 1 indicating a greater increase in CPU energy reduction, and values below 1 indicating a greater reduction of energy spent on

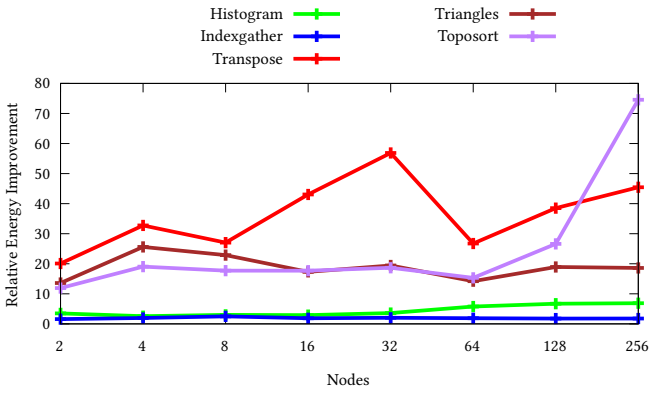

Figure 5: Dynamic Energy Improvement

memory accesses. With this, it becomes clear that while it's been established that there's significant improvements across the board to time and all tested levels of energy use, there's a disproportionately higher emphasis on the CPU. The one notable exception to this is histogram, for which the energy improvements are consistently greater for memory below 64 nodes, providing as much as twice the improvement to energy use as that of the CPU.

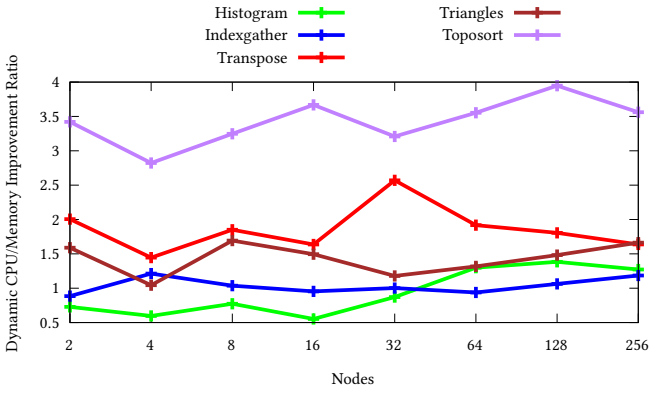

Figure 6: Dynamic CPU/Memory Energy Improvement Ratio

We can also note that the energy improvements for sparse matrix transpose, triangle counting, and topological sort easily dwarf that of histogram and indexgather. While we will address the results of all applications, we will apply additional focus toward these standouts so as to better determine the nature of these improvements. The next few sections will now individually examine each of these five applications in greater detail. For the figures showing results across progress checkpoints, results are shown using 32 PPN so as to mitigate some noise observable at 64 PPN, likely due to conflict with **OS!** threads, for which up to 8 cores are normally reserved.

3.3 Histogram

Histogram is effectively the starting point for the bale applications. Not only does it represent the simplest and most direct use for

conveyor aggregation, but it also serves as a good baseline for comparing aggregation strategies without introducing the complexity of additional variables. Histogram reproduces the patten of gathering and binning data from a large data set, with each PE continuously writing independent updates to random locations in a distributed table in a tight loop. This is also noteworthy for being a common component pattern used by a number of other more complex applications, including within bale itself.

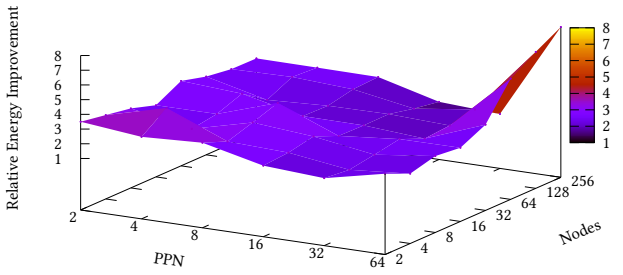

Figure 7: Histogram Dynamic CPU Energy Improvement

Figure 7 shows the CPU energy improvements of the conveyor implementation of histogram relative to that of the AGP version. As was noted in Section 3.2, this application was noteworthy for faring better in energy improvement over the CPU, but now we can see this in greater detail as the PPN varies from 2–64. We observe that this effect mostly applies to 64 PPN on higher node counts, remaining comparatively lower otherwise. Comparing this to the memory results in Figure **??**, we can clearly see the shift in memory dominating the relative energy concumption, though we also can observe a curve along the PPN counts, peaking around 8 PPN. This is consistent with our results from [Hernandez et al. 2024], albeit with the pattern becoming much more clear.

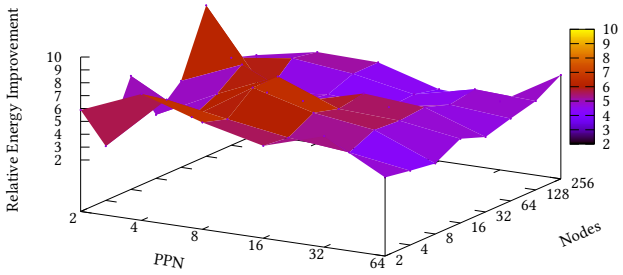

Figure 8: Histogram Dynamic Memory Energy Improvement

3.4 Indexgather

The next application is indexgather, which is another building block pattern very similar to histogram, but in reverse — instead of repeatedly writing into random remote locations, it issues reads accessing data from them. The comparison here is much more favourable for the AGP version than it otherwise would be since we modified the bale implementation to use non-blocking fetches instead of the original blocking routines. This provides a more fair evaluation for applications using the more recent OpeNSHMEM implementations supporting such non-blocking operations.

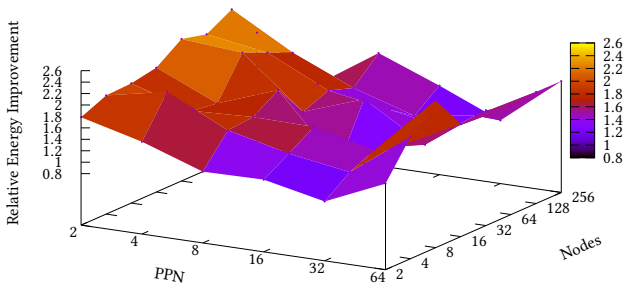

Figure 9: Indexgather Dynamic CPU Energy Improvement

Figure 9 shows the CPU energy improvements relative to conveyors for indexgather. It is interesting to note that its behaviour with respect to the PPN count stands in contrast to that of histogram — relative energy gains are much higher for lower values before it stabilises to roughly that of the memory imprvement for higher counts. This can be seen as compared to the energy improvement results in Figure 10. In this case, the relative energy savings over AGP is fairly stable across most all PPN and node counts. This begs the question of whether it would continue to remain constant for even higher PPN counts, whereas the CPU savings might continue to increase and causing the CPU/memory ratio to climb, but since there were no more cores this theory could not be tested.

3.5 Sparse Matrix Transpose

The sparse matrix transpose application does exactly what it advertises to — it takes a distributed sparse matrix encoded in compressed sparse row (CSR) format whose non-zero values are spread out across many PEs and produces the transposed result (also distributed across the same PEs). The value of this algorithm needs little introduction due to its prevalence in linear algebra and data analytics. Transpose consists of roughly two phases:

- Determine the number of non-zeroes in each column using a histogram pattern
- Perform the actual transpose by fetching each $A_{i,j}$ and writing them to the their appropriate $A_{j,i}$ destinations

Figure 11 and Figure 12 show the relative dynamic energy improvements achieved for the CPU and memory, respectively. The results seem pretty stable across all PPN and node counts, albeit

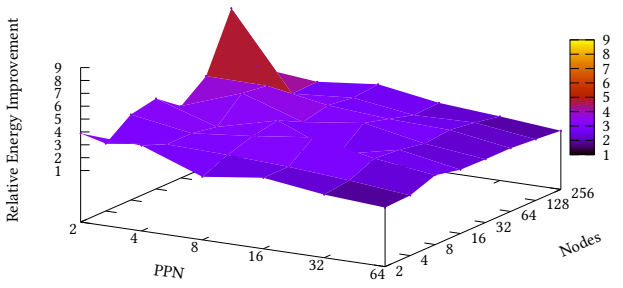

Figure 10: Indexgather Dynamic Memory Energy Improvement

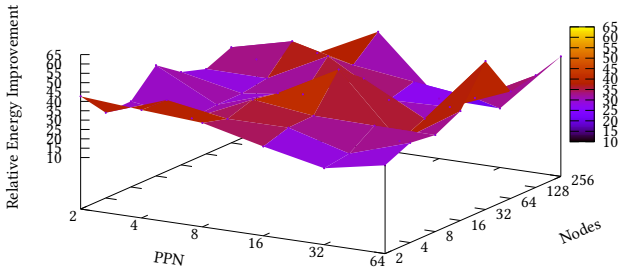

Figure 11: Transpose Dynamic CPU Energy Improvement

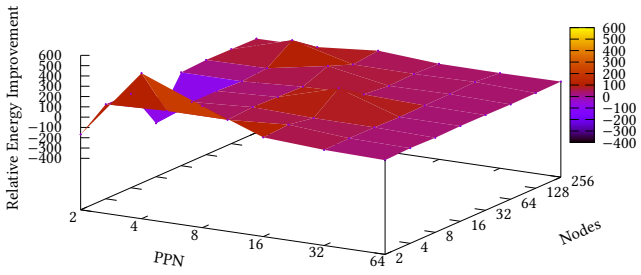

Figure 12: Transpose Dynamic Memory Energy Improvement

with a much higher improvement rate. Probing deeper, we look at the more detailed results over the checkpointed length of each execution, as described in Section 3.1, the results of which are seen in Figure 13 and Figure 14. Here, we look at the results for 32 PPN to avoid contending with some of the outliers seen for 64 PPN.

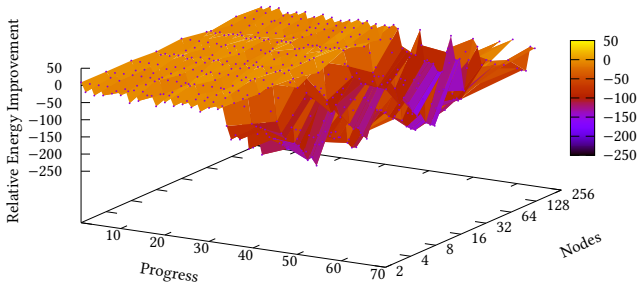

Figure 13: Transpose Dynamic CPU Energy Improvement Over Time

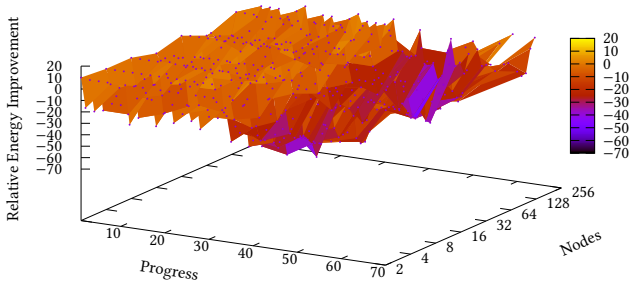

Figure 14: Transpose Dynamic Memory Energy Improvement Over Time

These plots clearly show the split between the transpose algorithm's first and second phases, with the first being pretty stable as a nominal improvement over AGP, but with the second phase being characterised by chaotic fluctuations and a massive *drop* in dynamic energy improvement. That is to say, the AGP version uses substantially *less* dynamic energy than conveyors. If it seems counter-intuitive that this could be true while conveyors still clearly outperforms in dramatically lower overall energy consumption, the key here lies in how much energy is consumed while the system is idle and how that factors into the results.

Since these plots effectively filter out not just execution time but also the energy cost of idling, they represent a fair comparison independent of those concerns. However, this also implies that

the massive difference in energy consumption comes largely as the result of paying for idle time while waiting on the results of network operations. In particular, there are two categories for idle time to account for here — the idle time incurred during the execution of a given implementation, and the idle time that comes after the completion of the faster implementation when compared to the slower one (which is really a better insight into a portion of the first category for the slower implementation). Due to the effects of the latter category of idle time not being visible in Figures 13 and 14, these suggest that either the second phase of transpose is either more resource-intensive for conveyors, suffers greater idle time, or a combination of both. In any case, the prior figures demonstrate a clear payoff for aggregation in terms of the resulting drop in overall energy consumption.

3.6 Triangle Counting

The next application searches for and counts triangles in undirected graphs using the lower and upper triangular matrices of its sparse adjacency matrix.

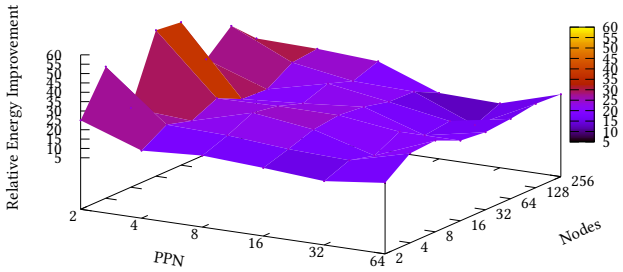

Figure 15: Triangle Counting Dynamic CPU Energy Improvement

Figure 15 shows the relative CPU energy improvement achieved for conveyors, with dramatically higher gains for 2 PPN before dropping down to a lower but stable improvement at and beyond 4 PPN. It is not immediately clear what might cause this bump, though we note that it may not be the most realistic choice for production runs. Figure 16 shows the relative memory energy savings, which is remarkably stable across different PPN/node configurations. Of the three especially high-performing conveyor implementations, this one was the most balanced with respect to CPU vs memory improvement, which these figures corroborate.

Figures 17 and 18 again show the breakdown of CPU and memory energy improvement throughout the course of execution. While these figures show a general increase in benefit at high node counts, there is also a clear trend of increasing improvement as the application gets closer to completion. Since this is a more dynamic graph algorithm, this suggests that it becomes more computationally inefficient to process as we get deeper into the graph (or more specifically, its adjacency matrix) when employing the classic AGP

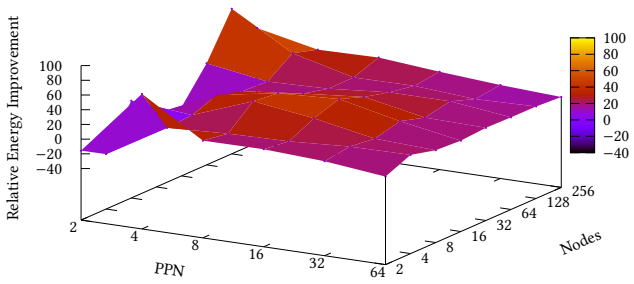

Figure 16: Triangle Counting Dynamic Memory Energy Improvement

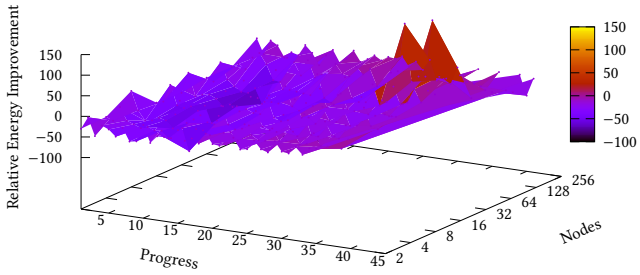

Figure 17: Triangle Counting Dynamic CPU Energy Improvement Over Time

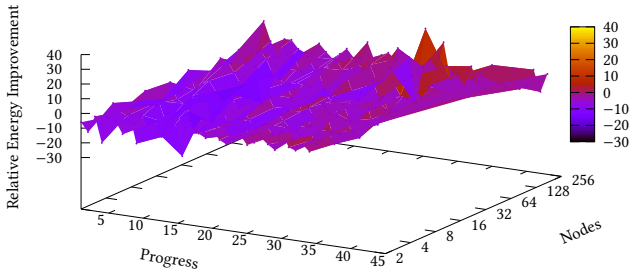

Figure 18: Triangle Counting Dynamic Memory Energy Improvement Over Time

approach. It is also worth noting that this application does not exhibit the same substantial drops in relative performance that the second phase of sparse matrix transpose did, implying that there is comparatively less of its improvement that comes from a reduction in idle time after the conclusion of the conveyor implementation (while not explicitly shown here, this was indeed substantiated by the runtimes of our results).

3.7 Topological Sort

The final application we tested from bale searches for a linear ordering of the vertices of a directed graph such that every vertex comes before any other that has an edge leading to it. This is the culmination of several smaller bale applications and as such is a prime target for demonstrating the viability and flexibility of aggregation. This is also one of the more interesting applications in that it is partially but not completely latency tolerant, due to working through many iterations of a work queue that grows as progress is made. In short, it searches for all vertices with degree 1 (i.e., only one outbound edge), then executes its primary work loop on these vertices, removing them and their edges from the graph as it proceeds. This invariably results in new degree 1 vertices that can then be worked on, with the application continuing to execute in this fashion until no more vertices remain.

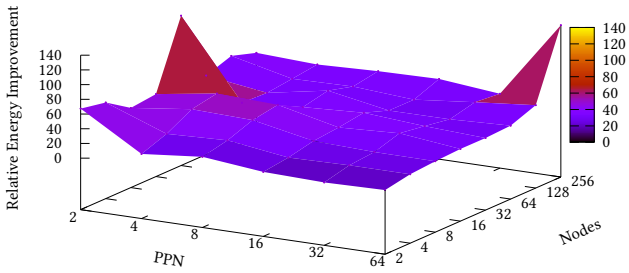

Figure 19: Topological Sort Dynamic CPU Energy Improvement

Figures 19 and 20 show the relative dynamic CPU and memory energy improvements achieved with conveyor aggregation, respectively. Other than a couple outliers, these show fairly stable improvements to both CPU and memory use, with a slight increase in relative energy improvement at 8 PPN. It is unknown what causes 8 PPN to be such an exceptional number, but its overall impact is fairly minor.

Figures 21 and 22 show the detailed breakdowns over the course of execution. Excepting a substantial drop at the very tail end of execution for high node counts, these show remarkably stable improvement patterns, with greater emphasis on CPU over memory gains. These consistently show improvement compared to the degradation we saw in the second phase of sparse matrix transpose,

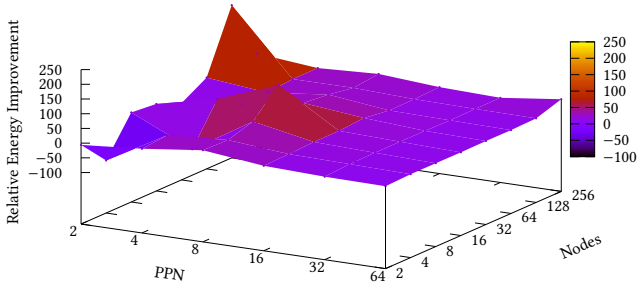

Figure 20: Topological Sort Dynamic Memory Energy Improvement

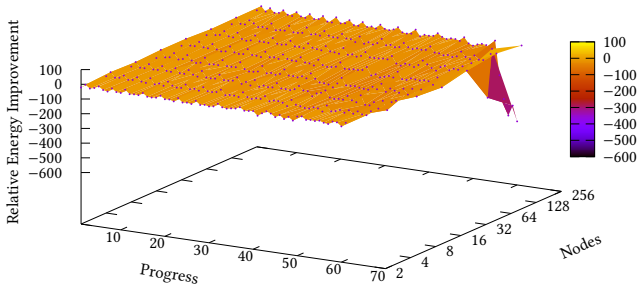

Figure 21: Topological Sort Dynamic CPU Energy Improvement Over Time

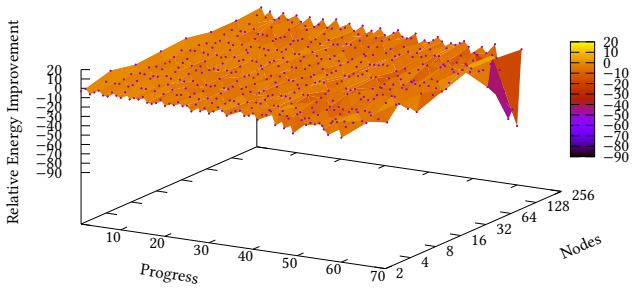

Figure 22: Topological Sort Dynamic Memory Energy Improvement Over Time

again suggesting that greater improvement comes from during execution compared to a reduction in idle time after the conveyor implementation's end.

The performance drop at the end comes as a result of the algorithm we previously described. As the application neards the end of its execution, each iteration on the work queue can be expected to become smaller as the number of degree 1 vertices shrinks, making the algorithm increasingly less latency tolerant as a result. The effects of this can then become noticeably more pronounced at higher node counts as a greater proportion of the data accesses will access network resources further away.

4 Conclusion

It's already been well established that network aggregation and conveyors provide substantial improvements to performance with respect to execution time when applied to applications exhibiting the small message problem. With the results of this work, we can also now see the clear benefits for application energy use. This benefit is easily able to achieve an order of magnitude reduction in energy footprint for some classes of applications, which is a level of improvement not usually attainable. As a result, aggregation has confidently proven itself to be vital to scaling these applications, especially as the burden of energy inefficiency continues to rise.

For future work, it would be interesting to repeat the study on other HPC systems and interconnects. Past results have strongly indicated that the Slingshot 11 network used by Frontier, while far from immune to the effects of congestion applied by the small message problem, is substantially more resilient to them. This could suggest that the already dramatic improvements observed in this study could potentially be magnified even further in other networks.

Additionally, our results identify the CPU as a prime source of these energy savings through less incurred idle costs, though we note that aggregation itself is not particularly CPU-intensive, instead being likely more dependent on instructions per cycle (IPC), memory access metrics, and general concurrency. Thus, it would be worth investigating whether far weaker CPUs with higher core counts might be capable of increasing this efficiency as well. Even more so, it could be valuable to determine whether some or all of the onus of aggregation may be simply offloaded to another device such as a SmartNIC, and whether this could generate even more energy savings by reducing the cost of aggregation itself.

Acknowledgments

This research used the Frontier resource of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This work was funded through Strategic Partnership Projects Funding Office via Los Alamos National Laboratory with IAN 619215901.

References

Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. 2020. An In-Depth Analysis of the Slingshot Interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. https://doi.org/10.1109/SC41405.2020.00039

Oscar Hernandez, Aaron Welch, Wendy Poole, and Stephen Poole. 2024. Preliminary Study on Message Aggregation Optimizations for Energy Savings in PGAS Models. In *VIVEKfest 2024: In Honor of Vivek Sarkar's Contributions to Parallelism and Programming Languages, part of SPLASH 2024*.

Gokcen Kestor, Roberto Gioiosa, Darren J. Kerbyson, and Adolfy Hoisie. 2013. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*. 56–65. https://doi.org/10.1109/IISWC.2013.6704670

F Miller Maley and Jason G DeVinney. 2019. Conveyors for streaming many-to-many communication. In *2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 1–8.

Naveen Namashivayam, Bob Cernohous, Mark Pagel, and Nathan Wichmann. [n. d.]. Early Experience in Supporting OpenSHMEM on HPE Slingshot NIC (Slingshot 11). ([n. d.]).

John Shalf. 2019. HPC Interconnects at the End of Moore's Law, In Optical Fiber Communication Conference (OFC) 2019. *Optical Fiber Communication Conference (OFC) 2019*, Th3A.1. https://doi.org/10.1364/OFC.2019.Th3A.1

Aaron Welch, Oscar Hernandez, Wendy Poole, and Stephen Poole. 2024. Scalable Small Message Aggregation on Modern Interconnects. In *VIVEKfest 2024: In Honor of Vivek Sarkar's Contributions to Parallelism and Programming Languages, part of SPLASH 2024*.