Title
Understanding Performance Variability on the Aries Dragonfly Network

Permalink
https://escholarship.org/uc/item/6ns0t6ns

Authors
Groves, T
Gu
Wright

Publication Date
2022-12-17

Peer reviewed
Understanding Performance Variability on the Aries Dragonfly Network

Taylor Groves  
Lawrence Berkeley National Laboratory  
Berkeley, CA  
tgroves@lbl.gov

Yizi Gu  
Computer Science Department  
Rice University  
Houston, USA  
yizi.gu@rice.edu

Nicholas J. Wright  
Lawrence Berkeley National Laboratory  
Berkeley, CA  
njwright@lbl.gov

Abstract—This work evaluates performance variability in the Cray Aries dragonfly network and characterizes its impact on MPI Allreduce. The execution time of Allreduce is limited by the performance of the slowest participating process, which can vary by more than an order of magnitude. We utilize counters from the network routers to provide a better understanding of how competing workloads can influence performance. Specifically, we examine the relationships between message size, process counts, Aries counters and the Allreduce communication-time. Our results suggest that competing traffic from other jobs can significantly impact performance on the Aries Dragonfly Network. Furthermore, we show that Aries network counters are a valuable tool, explaining up to 70% of the performance variability for our experiments on a large-scale production system.

I. INTRODUCTION

This work provides a preliminary characterization of performance variability in the Cray Aries Network [2] and examines how this variability impacts the communication time of collective operations. Aries is a high performance dragonfly network that is found in 17 of the 50 fastest supercomputers [21]. In an Aries network, traffic from competing workloads often share routes over a limited number of intra and inter-group paths, resulting in performance variability. To study the impact of this network-variability, we use the MPI_Allreduce collective. Blocking-collectives (like Allreduce) serve as an implicit synchronization point such that the overall performance is limited by the slowest process in the communicator. Furthermore, Allreduce is one of the commonly utilized collectives and represents a significant portion of execution time for many HPC applications and benchmarks.

Collectives performance may vary widely due to a number of factors including OS noise, network congestion, or computational load imbalance. While collective performance and system noise has been well studied [13], [14], [10], [22], [16], [12], [8], [20], [9], [15], [5], [4], [17], it is clear that the mechanisms and magnitude of interference vary across systems. Different design decisions, such as tapering of

network links, communication offloading can greatly impact how noise propagates through a system. For example, the BlueGene/Q network supports hardware assisted collectives and BlueGene/P provided separate collective and barrier networks, which create a layer of isolation from interference on the general interconnect. These differences across systems and architectures require us to explore performance at a level of detail that general purpose models (such as LogP [7]) cannot readily provide. This work provides a preliminary examination of the correlation between message size, process count, Aries router counters and Allreduce performance.

II. BACKGROUND

a) Allreduce: Allreduce is a widely used collective operation in which a reduction is performed with the result distributed among all processes in the communicator. Allreduce is widely utilized by HPC applications and its performance correlates with significant deviations in application runtime as it is susceptible to noise [4], [17], [9]. In a blocking Allreduce every process in the communicator must synchronize before continuing with local computation. Non-blocking Allreduce has the potential to mask some of the bottlenecks seen in traditional Allreduce, but this is dependent on the amount of communication and computational overlap at the application's disposal. Allreduce Message sizes commonly range between 8 Bytes and 8KiB in the APEX benchmark [18] collection, for this reason we focus on these message sizes in our analysis.

b) Cray Aries Network: The Cray Aries Network [2] is a modern dragonfly network used by many of the world’s fastest supercomputers. Aries is designed by dividing a number of nodes into logical partitions called groups. These groups are fully connected to each other by optical network links with multiple links per arc. Each group is further divided into six chassis with each chassis representing 16 blades. A blade consists of four compute nodes which share a Aries Router (Fig 1). Each Aries Router contains 40 network tiles that provide buffering and routing throughout the network.

When traversing the network, each packet must cross between one and six of these tiles (for minimal routing) to reach its destination. Each tile contains a finite buffer and the potential for the packet to experience congestion (stalls) if a tile downstream does not have the resources to facilitate a
we do not make any attempts to distinguish between related Allreduce latency and the measured counter. The only way to help shape best-case (minimal) latencies, but fail to provide much information about the observed latency of the slowest process. The importance of message size decreases as the number of processes increases in scale.

processes. For each combination of message size and process counts, we perform 30 iterations. We map one process per node and use the same SLURM allocation of hosts for each communicator. For this initial study no effort is made to confine processes to particular routers and frequently multiple nodes reside on the same Aries blade-router. The largest runs of 512 nodes cover two Aries groups (341 processes in one group and 171 in the second).

We record 25 counters and calculate 1 derived counter for each network tile on the router (40 tiles in total), and for each MPI process. These counters primarily measure flits and stalls across row and column buffers, buses, input queues and virtual channels. These counters do not represent a global view of the system, as we are only able to capture information for routers we have been allocated in SLURM. We utilize airesncl [3] (a third party library which parses counters and bins them into rank and tile combinations) to make analysis more convenient. For this initial study we focus on network tiles exclusively. For later studies we will examine both network and processor tiles.

IV. Analysis and Results

In our analysis we perform simple linear regression of single independent variables, e.g. message size, process count, Aries counter, and a dependent variable (latency of the slowest process in the Allreduce). We report the coefficient of determination ($R^2$) which represents the proportion of the variance explained by the independent variable.

A. Impact of Message Size on Allreduce

Message size is one of the common parameters used to model Allreduce performance. Message size divided by bandwidth determines the gap, or the amount of time needed to place all of the message on the wire. In an ideal environment, message size, together with latency would be the dominant factors in determining performance.

In Figs. 2-3 we show the impact that message size has on for a real Aries system. Fig. 2 shows, given 32-nodes, the latency of the slowest (red circles) and fastest processes (blue triangles) for 30 iterations of each message size. The variation of latency (even among the same processes and physical topology) ranges from 100 to 1400 $\mu$s. The fastest processes (blue triangles) represent close to idealized performance of Allreduce. The simple linear regression of the fastest process is fairly predictable ($R^2 = 0.77$). However, modeling the variation among the slowest process is much more difficult ($R^2 = 0.08$). As we increase the number of processes to 512 in Fig. 3, the accuracy of the simple linear regression falls even lower ($R^2 = 0.001, R^2 = 0.44$ for slowest and fastest processes, respectively). These results suggest that the message size of the Allreduce for workloads common on the Cori system can help shape best-case (minimal) latencies, but fail to provide much information about the observed latency of the slowest process.
In this section we plot the impact of communicator size on Allreduce performance. We keep message size fixed at 512 Bytes and increase the scale of the communicator. Fig. 4 shows that as we scale up the number of processes the variability of Allreduce latency increases. This behavior is true for all message sizes, but for brevity we limit our performance analysis to message sizes of 8B-8KiB. One MPI rank per node. Simple linear regression of the slowest processes has an $R^2 = 0.08$, while the fastest processes has $R^2 = 0.77$.

### Table I: Subset of recorded network counters and coefficient of determination ($R^2$) when used to model Allreduce latency (greater than 0, 90th and 95th percentiles, respectively)

<table>
<thead>
<tr>
<th>Aries Counter</th>
<th>Latency &gt; 0%</th>
<th>$R^2$</th>
<th>Latency &gt; 90%</th>
<th>$R^2$</th>
<th>Latency &gt; 95%</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>INQ_PRF_INCOMING_FLIT_VC0</td>
<td>0.32</td>
<td>0.19</td>
<td>0.65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_FLIT_VC1</td>
<td>0.44</td>
<td>0.63</td>
<td>0.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_FLIT_VC2</td>
<td>0.23</td>
<td>0.25</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_FLIT_VC3</td>
<td>0.02</td>
<td>0.01</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_FLIT_VC4</td>
<td>0.21</td>
<td>0.09</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_PKT_VC2</td>
<td>0.26</td>
<td>0.10</td>
<td>0.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_PKT_VC4</td>
<td>0.21</td>
<td>0.70</td>
<td>0.68</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_ROWBUS_2X_USAG_CNT</td>
<td>0.06</td>
<td>0.00</td>
<td>0.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_INCOMING_PKT_TOT</td>
<td>0.19</td>
<td>0.19</td>
<td>0.71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INQ_PRF_ROWBUSSTALL_CNT</td>
<td>0.22</td>
<td>0.48</td>
<td>0.71</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the previous two subsections we demonstrated that the traditional measure of message size and process count are fairly limited in their importance in determining Allreduce performance on the Aries network. There are many potential reasons this could be, including OS-noise or network congestion. In order to try to determine the root cause of the performance variability we have recorded 25 Aries counters for each router assigned to an allocated node.

If we were trying to examine every counter for every network tile over 30 iterations we would have to evaluate approximately 150 million counter measurements for the 512 node runs.\(^1\) Because we are trying to correlate performance of the slowest process with Aries counters, we extracted the maximum value of each counter for all tiles in each iteration. By extracting the maximum value, we reduce the number of measured counters (to 7500) and make our first evaluations significantly easier. We then iterated through each counter’s maximum value and performed simple linear regression between the counter and several percentiles of Allreduce latency.

In this section, we examined the 512 node runs across all message sizes. Typically we would differentiate by message size, but given the results in Fig. 3, we have shown message sizes limit.
size is not correlated to performance at high process count on our Aries network. Furthermore, we selected 512 process runs, since they experience the highest variability. We performed three simple linear regressions for each performance counter and full set of latency results, the 90th and the 95th percentile. In Table I, we show the results. For brevity we limit the table to counters who had an $R^2$ of 0.2 or higher.

The counters `INQ_PRF_INCOMING_PKT_VC*` record the flits that traversed a tile over a virtual channel (VC). If the system did not have competing workloads, we might expect that this value would remain constant across identical Allreduce operations. However, because we are on a shared system, competing workloads may result in additional traffic on shared routes. Request (put) flits travel through VC0-VC3, while VC4-VC7 transmit response (get) flits. Table I shows stronger correlation between Allreduce latency and request traffic. Additionally, at the 90th and 95th percentile the significance of request traffic increases while response traffic decreases. `INQ_PRF_INCOMING_PKT_TOT` is the sum of flits across all VCs, and has a $R^2$ of 0.71 for the 95th percentile.

`INQ_PRF_INCOMING_PKT_VC*` reports the rowbus stalls on a particular VC. Stalls on the request channel VC2 had an $R^2$ of 0.68, but stalls in the response channel VC4 were also related to a lesser degree ($R^2 = 0.25$). The counter `INQ_PRF_ROWBUSSTALL_CNT` is the sum of rowbus stalls across all VCs and had a $R^2$ of 0.71. The flit counters and rowbus stall counters both suggest that competing workloads do contribute to performance degradation on Aries networks.

In Fig. 5 we can see that there is a reasonable correlation but a counter does not explain all of the variability. In particular, there is one data point that has a latency of 35ms, but does not report a high number of stalls. Our hypothesis is that a high latency Allreduce could be the result of multiple stalls of lesser degree across several tiles in the route, rather than the tile with the maximum row-bus stalls. This is something we would like to explore further in future work.

**V. RELATED WORK**

Much of the work studying the impact of system noise on collective operations takes place in simulation, which provides a controlled environment for evaluation [12], [14], [8], [22], [16], [5], [15]. While simulation and does provide valuable insights, large-scale simulators are most frequently built on top of general purpose models (such as LogP). HPC simulators are frequently generalized for several reasons: 1) to explore design decisions in future systems for which architectural details are undecided, 2) simplifications allow for faster simulations at larger scales and 3) vendor implementation details are often trade-secret and not shared with the wider academic community. The downside of simulation is that these generalizations are unable to perfectly represent to all scenarios of real-world performance. For example no existing simulator perfectly matches the real adaptive routing strategies implemented by Cray, which are kept secret.

Other related work involves benchmarking and understanding performance on real systems [6], [20], [4], [10], [13], [11]. Our own work falls into this category, measuring Allreduce performance on a production system (NERSC’s Cori). Our work is most similar to work by Grant, Pedretti and Gentile [11] the major difference being that they examined a Cray Gemini system while we focus on the Aries network. Similarly, in work by Brandt et al. [6] they demonstrate the collection and analysis of Aries performance counters. Our work differs from this by presenting additional analysis focused on the performance of MPI_Allreduce.

**VI. CONCLUSIONS AND FUTURE WORK**

This work takes a simplified approach to exploring the relationship between Aries counters and Allreduce performance. Our results show that:

1) Traditional models of Allreduce performance (based on message size and process count) fail to capture performance variability of the Aries network.

2) Collective traffic on the Aries network does have substantial variability that increases with process count and Aries group count (Fig 4).

3) While there is no “silver bullet” among the counters investigated, changes in network traffic and stalls account for a 70% of the variability in the slowest 10% of Allreduce communication.

In future work we would like to see how our results using simple benchmarks maps to more complex applications. In order to provide greater accuracy, we will explore the use of more sophisticated techniques combining a wider range of network and communication features. Additionally it would be interesting to examine how we might reduce the noise in the collected data. For example by 1) removing unrelated network tiles and 2) examining all network tiles utilized by a workload (rather than the tile with maximal recorded counter). Given the complexity of routing and cross-job traffic, this presents an interesting challenge for future research.


