Integrating Heterogeneous Computing Techniques into the Daliuge Execution Framework

Feng Wang, Toby Potter, Xavier Simmons, Shifan Zuo, Jiangying Gan

May 15, 2017

Abstract

The ability to run MPI-based pipeline components is considered to be of great importance for DALiuGE, the prototype SKA execution framework. In this report we discuss several possible solutions on how to integrate heterogeneous computing techniques into DALiuGE. We explore various techniques for integrating MPI applications into DALiuGE and conclude that it is better to decompose the MPI application into executable components in order to fully utilize the DALiuGE execution framework. We also investigate using MPI communication within DALiuGE and report that implementing an MPI_Data_Drop can provide MPI communication mechanisms for transferring data between DALiuGE DROPs. We present a basic technical workflow on how to implement a MPI-aware data DROP in DALiuGE. Finally, we recognise the need to continuously research and incorporate heterogeneous techniques into the DALiuGE execution framework.

1 Introduction

The Square Kilometre Array (SKA) [1] will be the largest radio telescope in the world. As the first phase of the project, SKA1 will consist of hundreds of dishes and hundreds of thousands of antennas, and will enable the monitoring and surveying of the sky in unprecedented detail and speed. The second phase of the SKA will expand these capabilities by at least an order of magnitude. According to preliminary analysis, the low frequency component of the SKA1 project (SKA1-low) will produce correlated data at a rate of 466 GiBytes/s, and the mid-frequency component (SKA1-mid) will produce data at 446 GiBytes/s [2]. Correlated interferometry data from the array will be fed into the Science Data Processor (SDP), a Many-Task Computing (MTC) centre that is responsible for continuous data reduction and processing.

The Data Activated Liu Graph Engine [2] - DALiuGE - is designed to be an execution framework (EF) for processing large astronomical datasets at scales required by the Square Kilometre Array. It includes an interface for expressing complex data reduction pipelines consisting of both write-once read-many data sets (Data DROPs) and algorithmic components (Application DROPs). Both Application and Data DROPs form an event-driven framework that implements a processing pipeline as a set of directed acyclic graphs. The

DROPs are distributed over the available compute resources. To further improve and refine this EF, it is necessary to incorporate new features for the DALiuGE. Running heterogeneous applications under DALiuGE is a challenge but very important issue that we must consider in the current stage of development.

2 Project Requirements

With the advent of HPC, many astronomical data software packages, such as AIPS, AIPS++, Miriad and CASA have been implemented to process the massive amounts of astronomical data generated by modern radio telescopes. These frameworks have traditionally used either shared-memory techniques or the Message Passing Interface (MPI) [4] to distribute computation over processors. The distributed framework of DALiuGE has to date been using ZeroMQ (http://www.zeromq.org), raw TCP/IP sockets, and files for communication between nodes on a physical graph. This approach however, does not easily accommodate existing MPI infrastructure, nor does it make use of the decades worth of investment into MPI communication libraries. In this work we explore the following set of goals:

- Investigate supporting MPI pipeline tools with Daliuge

  DALiuGE already has the ability to directly run MPI applications due to the ability of some DROPs, e.g., BashShellAppDrop to invoke an application via a Unix/Linux shell command. However, at present the progress of the MPI application is hidden from DALiuGE - it cannot get any information such as the EVENTS and STATUS from MPI processes. This also means DALiuGE is not able to dynamically allocate and optimise resources for the MPI application at runtime. For example, if we design a logical graph with an eight-way scatter followed by eight iterations, there are total sixty-four drops, each of which is unique. However, for a standard MPI application, there are only eight processes and a loop with eight iterations in each process. In the MPI application instance DALiuGE has no way to track or control the events associated with each DROP. We aim to find ways to make DALiuGE aware of progress within existing MPI applications, control progress of execution and dynamically allocate resources.

- Explore using MPI communication routines to accelerate data transport within Daliuge.

  At present DALiuGE supports a series of communication approaches, such as raw TCP/IP, sockets, ZeroMQ, and files. The addition of MPI communication might accelerate data transport within DALiuGE drops. MPI communication within DALiuGE may be facilitated by including MPI-enabled DATA DROPs.

3 Technical Routine and Discussion

3.1 Monitoring and execution of MPI-based pipeline processing components within DALiuGE

On day one of the workshop we performed a thought experiment to work out how to give DaLiuGE runtime internal monitoring and control of an MPI application DROP. We explored two possible methods to incorporate existing MPI applications into DALiuGE.

1. Intercept MPI and iteration calls and send the events to DALiuGE
2. Decompose the MPI application into stateless components that could be manipulated directly by the DALiuGE graph engine.

For either method DALiuGE needs to be made aware of the logical structure of the MPI application. We suggest this can occur by uploading a logical graph of the MPI application prior to execution, or having DALiuGE "learn" the logical graph at runtime. The process of constructing a logical graph may be accomplished either manually or with a parser that generates a DALiuGE JSON file from MPI application source code. During execution, components of the MPI application being run correspond directly to points in the logical graph. Monitoring conceivably allows a restart of the MPI component from the point of failure and optimal placement on resources.

The first method investigated involved the intercepting MPI calls from the MPI application drop using a custom library that is preloaded before the main MPI library. The custom library re-implements standard MPI calls in such a way that it notifies DALiuGE of the calls and passes the data through to the host MPI library. We regard this approach as merely a monitoring solution, however, as it does not afford DALiuGE the flexibility to apply graph-based optimisation.

During a deeper discussion of implementing the intercept method, we encountered problems with assigning unique UUID's to MPI events as they may not have the same ordering from one execution to the next. Since MPI calls are rank-oriented we suggest that it may be possible to generate a UUID for each MPI call that mapped directly to a vertex on the physical graph using self modifying preprocessor directives (this would prevent us from implementing our own code modifier).

The second method we considered was to decompose an MPI application into several stateless components that could be separately invoked by the DaLiuGE engine. By enabling this capability, DaLiuGE could then further optimise the graph topology of an existing static MPI application. It would also provide a one-to-one mapping of the MPI application to the optimised graph topology and allow for runtime monitoring. Decomposing an MPI application into stateless components requires significant effort, however, and it is not clear if the expended developer time warrants the gains made by giving DALiuGE greater control over the running MPI application.

3.2 Enhancing DALiuGE with MPI communicators

On day two of the workshop we looked at how communication within DALiuGE could be enhanced using the MPI communication framework. Several decades of research has been

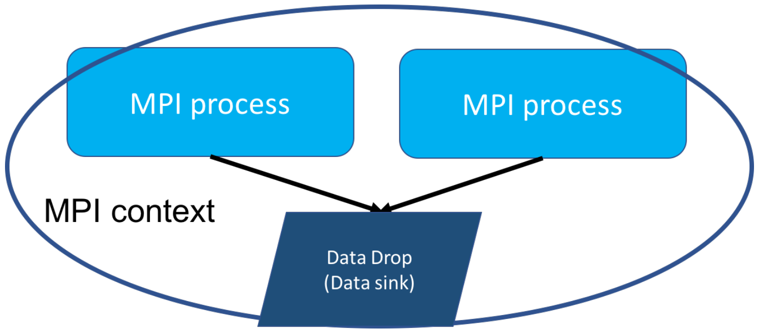

Figure 1: Sophisticated communication patterns like MPI gather can be obtained when a MPI context (shown here as an ellipse) is created between MPI processes (Applications DROPs) and Data DROPs.

spent on making optimised MPI communication patterns and libraries. It therefore makes sense to have the option to use these libraries within DALiuGE in addition to the ZeroMQ, file, and direct socket methods that it already supports.

In order for MPI communication to occur, processes need to be part of an MPI communication context, in which each process has a unique identifier, or rank. The communication context may be created on initialisation of DALiuGE, or during runtime with MPI dynamic process control. The most basic communication pattern within DALiuGE is simply point-to-point - application components get information from data drops and pass data drops to other components. It may also be desirable to have more sophisticated communication patterns like scatter/gather and reduce. Figure 1 shows the MPI gather communication pattern that is enabled when MPI processes are linked by an MPI context.

We investigated two forms of dynamic process control to create the context. The first method involves spawning child MPI processes from a master process and connecting the child process to the parent. The second method involves a more peer-to-peer approach in which the master advertises its context and its peers connect to the context in a way that forms an MPI intercommunicator between MPI contexts of peer processes. We find the peer-to-peer approach is favourable as it gives more control to DALiuGE to place individual processes and has more flexibility with regard to process placement.

We extend the idea of dynamic MPI context creation to the basic point-to-point data transport model in DALiuGE. Figure 2 shows data transport from a source MPI application drop through to an MPI data drop. In this instance the source MPI drop forms a master MPI process. The MPI drop forms an MPI context with the master MPI and data is transported using MPI send and receive calls. We achieve a preliminary system design of MPI Data Drop that is implemented as a mixin to both Application and Data DROPs. An application drop on a read/write request would instantiate a communication context and advertise its address to either its upstream or downstream drops (depending on the direction of the communication). This write/read function would then also invoke the connected data drops to create its own communication world and connect to the master's advertised address. Once a connection was successfully established, data could be sent or received across this communication fabric to take full potential of MPI hardware acceleration. That connection

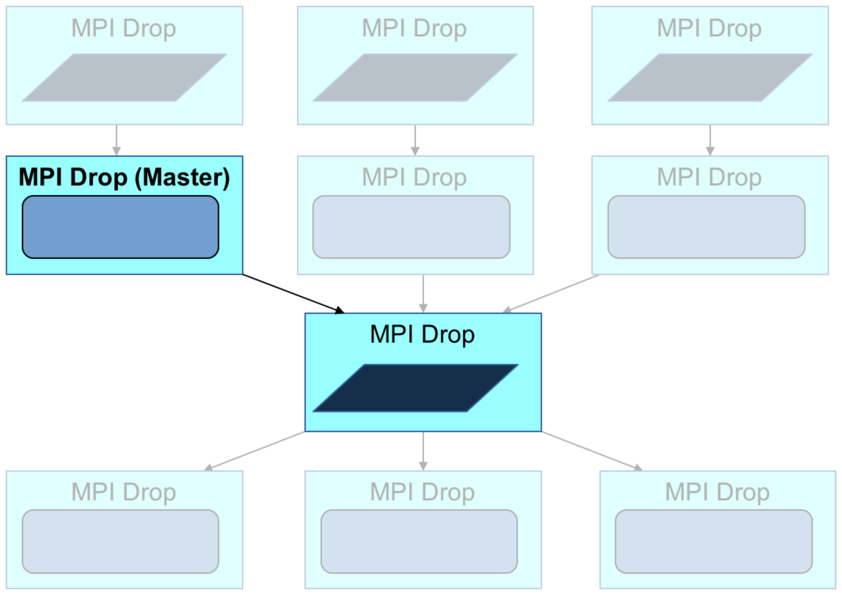

Figure 2: Dynamic MPI process management applied to simple message passing within DALiuGE. MPI contexts and hence MPI communication pathways are dynamically created between DROPs. Shown here is data being moved from an MPI application drop to an MPI-enabled data drop using simple MPI sends and receives.

would remain established for the remainder of the lifetime of the application. This would allow a drop to connect and disconnect dynamically without the performance penalty of reinstating MPI communication worlds and destroying them after each transaction.

4 Conclusions

We investigated and further understood the low-level implementation of DALiuGE. Two approaches to run MPI application under DALiuGE have been discussed and explored. We realise that ultimately MPI applications will need to be decomposed if they want to take full advantage of the DALiuGE engine. Meanwhile, we agree that using MPI to enhance the communication ability and performance between DROPs is valuable. Such communication is probably going to be achieved using dynamic MPI process management whereby MPI contexts are established at runtime, thus allowing data to pass over MPI links between application and data DROPs. In a summary, we find that continuous improvement of DALiuGE is a worthwhile task for for all SKA stakeholders.

5 Six month work plan

Five tasks listed as follows would be further studied and implemented in the next 6 months. A system prototype would be implemented

- Decomposing SageCal - 4 weeks

  SAGECal [3] is a self-calibration technique that uses the Expectation-Maximization (EM) algorithm to obtain the maximum likelihood estimation of the instrument and sky parameters. While the plain EM algorithm has been applied in estimating parameters for non-linear, superimposed signals, SAGECal uses an improved EM algorithm known as the SAGE algorithm to speed up the convergence and reduce the computational complexity.

  To further test and qualify the approaches, we intend to decompose the source code of SegaCal and construct the corresponding DROPs of the SAGECal. We hope to find a proper and versatile way to immigrate the MPI applications to the DALiuGE execution framework.

- Testing ZeroMQ vs MPI for data communication - 4-8 weeks

  Indeed, ZeroMQ is a simple but high performance queue software. It would provide low-latency data communication in Point-Point mode. However, if MPI communication mechanism can be integrated into the DALiuGE, the data communication model would has even more options. We will incorporate a new type of DATA DROP, i.e., MPI_DATA_DROP, to provide the high performance data communication between DROPs. Additionally, we may look at more sophisticated MPI communication patterns. For example, MPI_Scatter would distribute concurrent data to each processor, and MPI_Gather would collect data in parallel. It would significantly improve the low-level communication performance.

- Speeding up Algorithms by GPUs with DALiuGE - 8-12 weeks

  We will continue to study the algorithm speeding up approaches based on heterogeneous computing techniques such as GPUs, MIC, and MPI.

References

[1] Carilli, C. and Rawlings, S., *Science with the Square Kilometer Array: motivation, key science projects, standards and assumptions*, arXiv preprint astro-ph/0409274, 2004.

[2] Wu, Chen, et al., *DALiuGE: A Graph Execution Framework for Harnessing the Astronomical Data Deluge*,arXiv preprint arXiv:1702.07617 (2017).

[3] S. Yatawatta, S. Zaroubi, G. De Bruyn, L. Koopmans, J. Noordam, Radio interferomietric, *calibration using the sage algorithm*, Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop, 2009. DSP/SPE 2009. IEEE 13th, IEEE, 150155, 2009.

[4] Gropp, W., Lusk, E., Doss, N. and Skjellum, A., *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel computing, 22(6), pp.789-828, 1996