An Applications Approach to Benchmarking and Performance Modelling Low Latency Interconnection Networks

by

Dean Gordon Chester

A thesis submitted to The University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

The University of Warwick

October 2021

Copyright

Contents

List of Figures

List of Tables

Listings

Acknowledgements

I would like to thank my supervisors, Prof. Stephen Jarvis, Dr. Suhaib Fahmy and Dr. Gihan Mudalige for their support and in depth discussions that have shaped this work over the past four years. Their encouragement to develop as a scientist and continue to explore and push ideas and concepts has been an invaluable experience.

I would also like to thank Dr. Simon Hammond at Sandia National Laboratories for his continued support and mentorship throughout the project; without his guidance this project would not be where it is today. Additional thanks go to Dr. Taylor Groves at the National Energy Research Scientific Computing Center for discussions around network congestion and other types of distributed applications away from multi-physics.

I would also like to thank my past and present colleagues and friends in the High Performance Scientific Computing Group: Dr. Steven Wright, Dr. James Davies, Dr. James Dickson, Dr. Dominic Brown, Dr. Richard Kirk, Dr. Andrew Owenson, Andrew Lamzed-Short, Alex Cooper, Archie Powell and Kabir Choudry for making the office environment an enjoyable place to work and study. Special thanks to Dom and Richard for having been there since the start and through the entire journey from the frustration to the laughter which encouraged me to improve and continue.

From the Department of Computer Science I wish to thank Maria Ferriero, Emma Woollacott, Sharon Howard, Dr. Roger Packwood, Paul Williamson, Richard Cunningham and the rest of the secretarial and technical staff for their assistance around administration, technical support throughout my Ph.D allowing me to focus on research.

To the staff from AWE who have gone above and beyond to aid my research, thank you Dr. Timothy Law, Dr. Seimon Powell and Prof. Richard Smedley-Stevenson.

Dr. Matt Ismail and Dr. Dugan Witherick based at the Scientific Computing Research Technology Platform, thank you for your patience and assistance with providing access to computing resources to facilitate my research.

To my family and friends, Mum, Dad, Dr. Adam Chester, Tara, Hope, Louise, Ben, Dr. John Galvin, William, Linda, Auntie Penny, Uncle Dave, Paul Fellows, Simon Cooper, Dr. Simon Fowler, Dan Prince, Dr. Peter Butcher, Joe Frangoudes, Noah Hall, Helen McKay and James Van Hinsbergh thank you for the help and support over the last four years, I would have not been able to have done this without your support. I would lastly like to thank my partner Fiona for her love and support.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work presented (including data generated and data analysis) was carried out by the author except in the cases outlined below:

- The simulator presented in Chapter 5 was developed at Sandia National Laboratories as a collaborative effort between the Department of Energy (DOE), Industry and Academia. Some of the work outlined in this thesis has contributed to the development, testing and validation of this simulator.
- Measured data from Astra was collected by Dr. Simon D Hammond.
- Measured data from Cori was collected by Dr. Taylor L Groves.
- Figure 4.1 was created by Dr. Steven A Wright.
- Inode Sizes from Cori's Storage Systems were collected by Dr. Lisa Gerhardt, Kirill Lozinskiy and Ravi Cheema (data for Figure 4.14).

Parts of this thesis have been published by the author:

[35] D. G. Chester, S. A. Wright, and S. A. Jarvis. Understanding communication patterns in HPCG. *Electronic Notes in Theoretical Computer Science*, 340:55–65, 2018

[34] D. G. Chester, S. A. Wright, S. D. Hammond, T. R. Law, R. P. Smedley-Stevenson, S. Maheswaran, and S. A. Jarvis. Full-system modeling and simulation : contributions towards coupling, contention, and I/O. In *MODSIM 2019*, 2019

[32] D. G. Chester, T. L. Groves, S. D. Hammond, T. R. Law, S. A. Wright, R. P. Smedley-Stevenson, S. A. Fahmy, G. R. Mudalige, and S. A. Jarvis. StressBench: A Configurable Full System Network and I/O Benchmark Framework. In *Proceedings of ISC HIGH PERFORMANCE 2021*, 2021

[33] D. G. Chester, T. L. Groves, S. D. Hammond, T. R. Law, S. A. Wright, R. P. Smedley-Stevenson, S. A. Fahmy, G. R. Mudalige, and S. A. Jarvis. StressBench: A Configurable Full System Network and I/O Benchmark Framework. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021 (Awarded Best Paper)

Abstract

As the field of High Performance Computing (HPC) approaches the Exascale era we see larger systems coming online with a rich set of applications and programming paradigms given the diverse system architecture employed to deliver petascale levels of performance. Underpinning these distributed applications is the use of interconnected nodes; something which can contribute to significant performance degradation when a machine is highly utilised.

This thesis examines the interactions between communication patterns commonly seen in distributed applications written on top of Message Passing Interface (MPI), with a benchmark framework (StressBench) designed to orchestrate concurrent communication patterns. Application replay through StressBench yields reproducible applications with in 15% difference in runtime, showing that it provides a useful; abstraction from commercially sensitive production applications. A congested workload is distributed across two supercomputers demonstrating a slow down for application and Input Output (I/O) traffic, and the effects of job placement on I/O and application traffic is investigated with the benchmark framework.

This thesis documents a validation methodology for a layered simulator built on top of Structural Simulation Toolkit (SST). Using the validated hardware platforms accurate performance models are developed for four systems, and two applications on top of the SST which are then used to evaluate future network designs, both to support the development of next generation interconnection networks and design responses for an Request for Proposal (RFP).

This thesis is dedicated to my Grandfather.
William ‘Bill’ Chester
(1934-2018)

Sponsorship and Grants

The research presented in this thesis was made possible by the support of the following benefactors and sources:

- Joint AWE-Warwick University: Centre for Computational Physics (CCPP) (Grant Number: 30411884)

Abbreviations

ACK Acknowledgement

API Application Programming Interface

ASIC Application Specific Integrated Circuit

AWE Atomic Weapons Establishment

BDW Broadwell

BM Benchmark

BSP Bulk Synchronous Parallel

CFD Computational Fluid Dynamics

CG Conjugate Gradient

CPU Central Processing Unit

DOE Department of Energy

GPFS General Parallel File System

GPU Graphics Processing Unit

FFT Fast Fourier Transform

FLOP/s Floating-Point Operations Per Second

HPC High Performance Computing

HPCG High Performance Conjugate Gradients

I/O Input Output

IPL Instruction-Level Parallelism

IQR Inter-Quartile Range

ITAC Intel Trace Analyzer and Collector

HSW Haswell

KNL Knights Landing

LAN Local Area Network

OFI OpenFabrics Interfaces

OSI Open Systems Interconnection

OSU Ohio State University

MIMD Multiple Instruction Multiple Data

MPI Message Passing Interface

MISD Multiple Instruction Single Data

NACK Negative Acknowledgement

NERSC National Energy Research Scientific Computing Center

NIC Network Interface Controller

PGAS Partitioned Global Address Space

PMIx Process Management Interface - Exascale

QoS Quality of Service

RFP Request for Proposal

SAI Stalled, Active, Idle

SDLC Systems Development Life Cycle

SHMEM Symmetric Hierarchical Memory

SISD Single Instruction Single Data

SIMD Single Instruction Multiple Data

SPMD Single Program Multiple Data

SST Structural Simulation Toolkit

UGAL Universal Globally Adaptive Load-balanced Routing

VC Virtual Channel

CHAPTER 1

Introduction

Computation provides a vital method for investigating scientific phenomena that are impractical to physically measure. It proves a safe, reliable and repeatable environment to experiment. Some of these simulations require larger computers to provide the computation resources, called supercomputers. Supercomputers are large machines that support calculating large amounts of computation, typically faster than desktop computers. The research field around design, development and support of these software and hardware systems is called High Performance Computing (HPC).

The Atlas Computer (delivered in 1962) is considered one of the first *supercomputers* [82], it could perform a floating-point multiplication operation in around 5ms. Modern supercomputers can perform often 10 orders of magnitude more Floating-Point Operations Per Second (FLOP/s) than these systems although they have shaped how HPC has developed. Some current supercomputers are constructed from commodity of-the-shelf hardware and networked with a low latency interconnection network called an interconnect, such as InfiniBand [1] which is commonly used. Previous interconnections networks have come and gone due to more advanced technology developing, one such network is Myrinet [20], originally a 1Gb/s link specifically designed for supercomputers due to it's low latency and high bandwidth, with a 10Gb version being released in 2005. Machines first appeared with this network in 1995 and the last machine appeared on the June 2014 TOP500 list [121]. Custom proprietary networks are also common place inside of supercomputers with IBM (for BlueGene product range[1]) and Cray/HPE developing specialist interconnects for supercomput-

[1]See: `https://www.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_sm/1/877/ENUS0207-_h01/index.html`

ers. The current Cray/HPE offering in the EX series[2] is a 200Gbps network providing low latency but higher jitter compared to an equivalent InfiniBand solution [47]. Newer networks provide advanced routing algorithms which may not provide the minimal path, this could result in higher latencies and decreased network throughput. This increases the time to solution for an application.

Modern distributed HPC applications typically have to exchange data as the problem sizes make them unfeasible to run on a single node, this is because of power, cost and physical size limitations. One approach is to distribute the problem over multiple nodes thereby achieving more physical memory, somewhere in the order of petabytes. This data exchange either between CPU cores (intra-node) or nodes (inter-node) is limited by latency and bandwidth of the memory and/or network subsystem. As the next era of supercomputers enters (named the exascale-era) we see restrictions on power requirements with the Department of Energy (DOE) (a large purchaser of supercomputers) stating the first generation of exascale supercomputers should consume 20MW at most [81]. This is unachievable with current hardware configurations. In an effort to satisfy this power requirement we see new node architectures developing which include an accelerator(s) such as a Graphics Processing Unit (GPU). Summit (the second fastest supercomputer as of the June 2021 TOP500 list [123]) features 6 GPUs per node [103].

Large supercomputers typically run lots of small jobs rather than one full system job (although this does happen) at one time, these jobs can interfere with each other affecting application runtime and performance [131]. The effects of the inter-job interference is a relatively new field in the HPC research area and is of great interest as this can affect application communication performance by as much as 50% for application communications on highly utilised supercomputers [131]. Figure 1.1 shows how some applications could interfere with each other throughout an application run. The vertical dotted lines highlight the points at which network contention could affect performance and the shapes

[2]See: `https://support.hpe.com/hpesc/public/docDisplay?docId=a00109703en_us&docLocale=en_US`

represent different communication patterns. In reality contending application communication patterns will not lineup as neatly as the abstract figure due to load imbalance and OS jitter, but provides a good visual representation for the problem at hand.

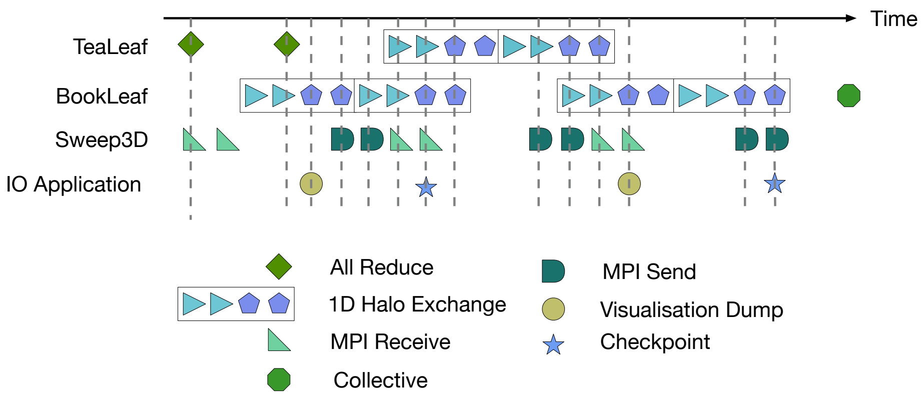


Figure 1.1: Interacting Applications in relation to time

Benchmarking advanced features is widely used to compare and contrast Central Processing Unit (CPU) architectures yet are largely ignored for the networking with current procurement procedures focusing on peak performance numbers to aid decisions, which may not always be attainable in production.

This thesis investigates the benchmarking and modelling modern low latency interconnection networks by focusing on application communication patterns rather than traditional network benchmarks such as latency and bidirectional bandwidth. This work aims to act as a case study for benchmarking and performance modelling activities for future design and procurement processes as the techniques presented in this thesis exercise the advanced networking features that are arising in current and next generation low latency interconnection networks.

1.1 Motivations

Compute hardware has improved at a rapid pace of the last few decades resulting in CPUs allowing results to be computed faster although this data movement has become the bottleneck due to memory and Input Output (I/O) subsystem performance not growing at the same rate [93]. Both of these impact the network performance which results in slower time to solutions for applications, as does larger dense nodes seen in recent supercomputers. The memory bottleneck can have a significant performance implications for data transmitted over the network as it limits the network injection rate, to see in advances in more memory bandwidth core counts are growing per socket to deliver performance. The I/O subsystem can impact the network traffic as latency sensitive (small) messages have to contend with the large bandwidth sensitive (large) messages which leads to increases in the time to solution for applications, thus slowing them down.

As network architectures evolve we are seeing the development of adaptive routing algorithms, congestion control and Quality of Service (QoS) effect the end user experience and time to solution for applications because these features try to act more fairly than providing the minimum path, while the minimum path should deliver the best performance, this may not always be the case when a network becomes congested. These issues will only continue to deteriorate as networks grow in size.

Memory performance limits the network injection rate as memory has to be read and injected in to the network when transmitting, and written to memory once received [89, 90, 124]. While recent developments allow the Network Interface Controller (NIC) to interface with the memory directly removing the need to communicate with the CPU the memory bus latency and bandwidth becomes the issue. This can be further impacted by how the memory is managed by an application, in the case of a simple latency benchmark the send/receive buffers could be contiguous memory where as an application may have to stride across the memory accessing non-contiguous blocks, when accessing multi-dimensional arrays [56].

Slow parallel I/O performance means that I/O traffic has to reside inside of the network while waiting to be relayed to the I/O nodes and depending on the network design and job placement could affect the time to solution for applications. While we explicitly state I/O traffic this could be any type of traffic waiting to propagate through a network; typically I/O traffic has to wait for the I/O node to read/write to disk which is slower than system memory. This type of traffic typically is also routed to the same endpoint further increasing chances of network contention.

1.2 Contributions

This thesis makes the following contributions:

- We present the design and implementation a new Message Passing Interface (MPI) and I/O Benchmark framework (StressBench) capable of driving and stressing network fabrics based on application workloads. Replicated application workloads of four applications have been validated, with a maximum difference of 15%. A full system workload is replicated to demonstrate application performance degradation due to network and I/O contention across two systems. We demonstrate how StressBench can be used for performance studies investigating the effects of network contention with multiple patterns against I/O traffic.

- We present a validation methodology and validated network models for three common network fabrics built on top of the Structural Simulation Toolkit (SST). These validated network models are used to develop accurate performance models for Sweep3D and TeaLeaf. Using these network models the design space of interconnection networks is explored for potential exascale systems, looking at cost, switch radix and runtime performance.

- Contention aware performance models are constructed to understand the performance degradation due to shared resource contention in the network. We demonstrate that with a tapered Fat Tree bandwidth can reduced by as much as 2GB/s when in contention with an 64K AllToAll congestion pattern. StressBench is used to compare the effects of real network performance against the simulated network making use of the full system workload introduced with StressBench.

1.3 Thesis Structure

The structure of the thesis is as follows:

- Chapter 2 describes the terminology and techniques used in the High Performance Computing, Benchmarking and Performance Modelling. This chapter also introduces the design of a low latency interconnection network.
- Chapter 3 presents the hardware and software used throughout this thesis. The three hardware platforms are documented and two applications have been used throughout this thesis Sweep3D and TeaLeaf.
- Chapter 4 documents the design and implementation of StressBench, application replication validation results and two performance studies to understand the effects of network contention on application performance.
- Chapter 5 introduces network simulations with SST, presents the validated network models and performance models for Sweep3D and TeaLeaf, and future network design exploration is explored.
- Chapter 6 outlines contention aware performance modelling and presents contention aware performance models to understand performance degradation through simulation.
- Chapter 7 summaries the research outlined in this thesis.

The work presented in this thesis started from understanding communication patterns in an application [35], the tools and techniques used in that paper have been used throughout this thesis. The work from Chapter 4 has been published as both a poster [32] and research paper [33] at ISC 2021 and IEEE HPEC respectively. The analysis of network degradation inside of the network switches has been published as a poster [34] at MODSIM 2019.

CHAPTER 2
Background

Performance gains can come through two forms, software optimisations or hardware optimisations. For software optimisations ensuring that the hardware is fully utilised in the best way possible. With hardware optimisations ensure that the application is running as fast as the hardware will allow. Both of these optimisations need to be exploited to achieve a high level of performance. This performance is often limited by some underlying theories which are described in this chapter.

This chapter is as follows, Section 2.1 presents some of the underlying laws governing parallelisation. Section 2.2 introduces system engineering concepts and shows the state of the art for benchmarking and performance modelling. Section 2.3 is a primer on network architecture introducing the three key elements in a low latency interconnection network.

2.1 Parallelisation

This section covers the laws governing the behaviour and limits of parallel computation and the classifications and types of parallelisms that can be exploited to obtain high levels of performance from parallel applications.

2.1.1 Speedup

Speedup is a measure of scalability for an application, it is the ratio of serial runtime (T_s) over parallel runtime (T_p) (see Equation (2.1)). This metric allows for a quick comparison to see how an application is performing as the parallel runtime decreases with more computational resources. If an application demonstrates a linear speedup then the application scales well.

$$S = \frac{T_s}{T_p} \tag{2.1}$$

G Amdhal et al. proposes that the maximum limit from parallelisation is governed by the time spent performing serial operations (see equation 2.2) [10]. In equation 2.2 s and p refer to the serial and parallel execution times respectively and n is the number of processes the parallel section is spread across.

$$\text{Speedup} = \frac{s+p}{s+\frac{p}{n}} \tag{2.2}$$

The approach taken by G Amdhal et al. focuses on a varying problem size rather changes in runtimes. Gustafson proposes the following change to calculating speedup as seen in equation 2.3 [62].

$$\text{Speedup} = n + (1-n)s \tag{2.3}$$

This approach focuses on the parts of the application that are typically fixed. This approach is favoured as problem sizes typically grow to match capacity rather than choosing to run the same problem size faster across more resources [62]. This can be considered to demonstrate how well an application weak scales.

2.1.2 Flynn's Taxonomy

Flynn et al. proposes four types of parallelism for computer architectures [55], Figure 2.1 shows the differences in the four ideas proposed. Single Instruction Single Data (SISD) provides a basic building block of computation, one process operates on a single data block with one instruction. Single Instruction Multiple Data (SIMD) applies the same instruction to multiple blocks of data simultaneously, this is similar to a vector processor or instruction set like Intel SSE or AVX instructions. Multiple Instruction Single Data (MISD) allows for multiple instructions to operate on the same data allowing for task repetition supporting fault tolerance. This approach is uncommon but one example of a MISD design are the flight control computers of the space shuttle [118]. Multiple Instruction Multiple Data (MIMD) allows for multiple instructions to operate on multiple blocks of data, most of the implementations for MIMD are shared memory based.

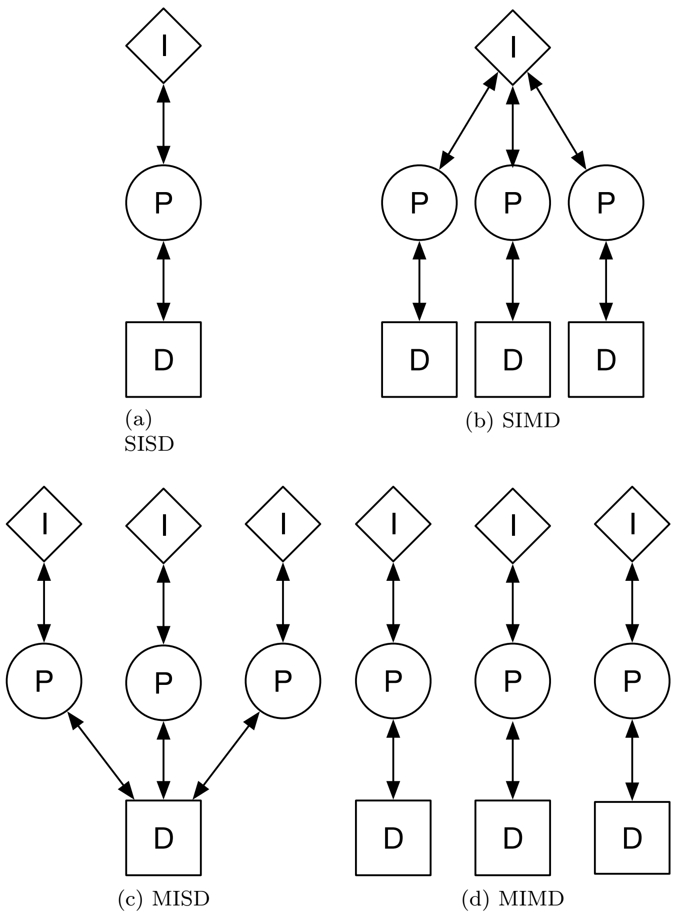


Figure 2.1: Flynn's Taxonomy, D represents Data, P is the Processing Elements and I is the instruction

In addition to these classifications there exists another classification, Single Program Multiple Data (SPMD) [45]. This classification has become synonymous with parallel scientific applications, such as MPI based applications.

2.1.3 Types of Parallelism

Three types of fundamental parallelism exist, Task Level, Data Level and Instruction Level. If all of these are exploited effectively then an application will perform optimally, but often some of these parallelisms are unable to be exploited together due to loss of accuracy. This is because the order of floating-point operations differ between an single threaded and multi-threaded loop, this can be corrected by ensuring that floating-point operations occur in the correct order.

Task Level Parallelism

Task parallelism can take the form of either message passing, or a multi-threaded approaches. There exists a hybrid message passing and multi-threaded approach to ensure resources can be utilised to their fullest, this is useful with certain unstructured applications such as LULESH [74].

Multi-Threading This approach allows communication where memory is located on the same array and can be easily achieved with a variety of libraries, such as PThreads[1] or OpenMP [41]. OpenMP is more common for HPC applications than PThreads.

OpenMP is implemented with the use of compiler directives (called pragmas) and the compiler handles the creation and deletion of the thread. OpenMP uses the fork-join model for thread creation and deletion. At the start of the parallel execution block threads are created and forked and then perform their portion of work; finally joining when the last thread has completed its work [2]. Listing 2.1 shows a simple threaded loop with OpenMP.

```
#pragma omp parallel for
for(int i = 0; i < x; i++)
{
    u[i] = u[i] * constant;
```

[1]See: `https://man7.org/linux/man-pages/man7/pthreads.7.html`

```
}
```

Listing 2.1: OpenMP Example

One disadvantage of multi-threaded applications is that they must reside on the same node as the data cannot be passed between nodes. This limits the problem size given restrictions on RAM and processing power.

Message Passing Message passing allows inter-node communication to take place so messages can be passed around sharing information between nodes. The MPI standard [57] defines the interface for implementations to provide, this is often abstracted away from the underlying network hardware (see Section 2.3.1).

Several studies have looked at the effect of running message passing for intra-node demonstrating that there is a performance impact compared to a shared memory, multi-threaded approach [73, 110].

Data Level Parallelism

Vectorisation is a way to make Data Level Parallelism possible and can be represented by SIMD from Flynn's Taxonomy. Vector processors operate on large one-dimensional arrays of data called vectors. These approaches were common in the early days of supercomputers with the Cray-1 being a vector machine [111].

Vector processing units are now commonly built in to generalised CPU architectures with additional instructions such as AVX512 [109].

Instruction Level Parallelism

Instruction-Level Parallelism (IPL) shows how many of the operations a computer program the hardware can perform simultaneously. To achieve IPL two techniques could be used, firstly increasing the depth of the instruction pipeline, allows more operations to be overlapped. A second approach is to increase the number of instructions that are run at every stage of the instruction pipeline [106].

2.2 Performance Engineering

Performance engineering covers a wide variety of techniques to ensure systems comply to requirements during the Systems Development Life Cycle (SDLC). This section discusses two techniques used throughout the lifetime of a HPC system, benchmarking and modelling. Benchmarking demonstrates that a system can ascertain a level of performance once constructed. Modelling allows for exploration of design spaces to fulfill system requirements, one of the advantages of modelling during the SDLC is that it provides indicative idea of the level of performance a design may provide. In this thesis performance relates to the time to solution for an application or the time for a benchmark to complete.

2.2.1 Benchmarking

A benchmark is a simple application that is run that stresses a a system or subsystem, for example LINPACK [52] a linear algebra benchmark used to stress the computation power of a system and STREAM is designed to only stress the memory subsystem [92].

The TOP500 presents a list of the machines that perform well for LINPACK, more recently High Performance Conjugate Gradients (HPCG) has become increasingly common as a benchmark as this stresses a supercomputer similar to a production application [67]. HPCG uses a preconditioned CG method with a local symmetric Gauss-Seidel preconditioner.

Benchmarks come in varying complexity from micro-benchmarks through to production applications like Sweep3D [80, 100, 53]. Given the complexity of modern parallel applications the rise of proxy/mini applications which capture some of the key resource characteristics (such as network traffic [78], computation [105]). Figure 2.2 shows the relationship of proxy applications to both production applications and microbenchmarks.

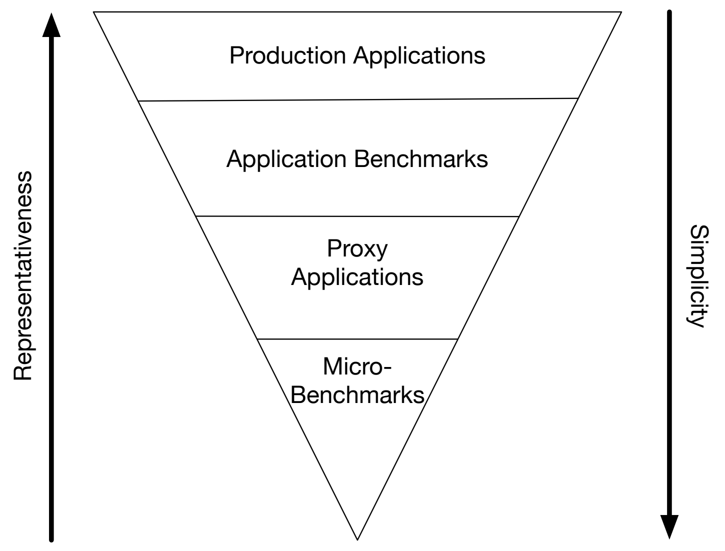


Figure 2.2: Representativeness and Simplicity of Applications Scale

Network Benchmarks

The usual approach to assessing the performance of massively parallel systems consists of executing a large set of benchmarks with a variety of differing communication patterns, often sequentially. How concurrently running applications interact with a machine's shared resources (i.e., the interconnect and parallel file system) is usually difficult to understand. As a result, these benchmarks fail to provide an accurate picture of application performance as they are unable to capture realistic network usage and highlight potential issues such as load imbalances that may affect the performance of collective operations [88].

ScalaBenchGen provides a way to automatically trace and replay applications as a synthetic MPI benchmark [130]. This approach uses the MPI Profiling layer (PMPI) to capture the MPI events which are stored chronologically; these events are then replayed through a custom tool. One limitation of ScalaBenchGen is that it provides no capability to scale communication sizes as multiple application traces must be captured with varying sizes.

File I/O can often interfere with MPI communications as large amounts of traffic are sent and received over the network. Dickson et al. have studied the I/O characteristics of large applications by replicating I/O workloads with MACSio [50]. This is achieved by capturing Darshan [27, 28] logs of applications, parsing the log files and generating input parameters for MACSio. Darshan captures I/O characteristics from applications, this is achieved by wrapping the I/O function calls at link time. These additional functions handle the timing and statistics aggregation. Darshan does not store all I/O events like a an MPI trace tool such as Intel ITAC [70] but rather aggregates the data to characterise the I/O patterns reducing the data generated [27, 28].

Common MPI Benchmarks include the Intel MPI Benchmarks (IMB) [69], OSU Microbenchmarks (OSU) [104], and SKaMPI [110]. These microbenchmarks focus on the performance of singular MPI operations: either point-to-point or collective operations. They are useful when trying to diagnose application performance issues as they often report the average time for MPI operations. SKaMPI is no longer under active development but was extended to cater for complex communication patterns [64].

The NAS Parallel Benchmarks replicate commonly used application patterns to benchmark systems [12]. While this benchmark suite comprises a wide variety of parallel patterns it does not orchestrate them to show how the patterns can interact with or affect one another.

More recent benchmarks such as GPCNeT look at testing network performance in isolation and under load [36]. GPCNeT provides artificial noise in a network with four congestor patterns and is designed to stress a system rather than provide representative communication of a specific workload.

Task Bench is a parameterised benchmark for evaluating parallel systems [117]. It allows for rapid replication of a variety of programming models and applications. Configurable parameters allow the tuning of the length of the benchmark; the degree of parallelism; the type of kernel (such as a stencil or sweep) and tuning of any potential imbalance. This task-based approach is a novel idea that allows for flexibility and customisation. One drawback of this tool is that it only focuses on one task at a time meaning that it is difficult to understand how the chosen benchmark will perform in a production environment.

I/O studies looking at improving performance are not new [5, 132, 14]. These studies typically focus on tuning I/O parameters for specific applications and systems. They often fail to consider the I/O subsystem being a shared resource and as such what contention may be affecting their performance.

Wright et al. have investigated the effects of I/O performance in relation to contention of the I/O nodes within a system [129]. They note that contention within the I/O subsystem can result in a 13% performance decrease on a multi-user system.

Limitations of Current Benchmarks

A large proportion of the benchmarks presented are designed to measure the peak network performance and often fail to test the advanced network features, the exception to this is GPCNeT which is designed inject noise in to the network to see the effects on common benchmarks. One issue with GPCNeT is that it splits the entire MPI Communicator randomly, this means that a congestor could reside on the same node and socket as a the benchmark under test. In reality production systems are configured to not mix applications on the same node or socket. Another limitation with GPCNeT is that it only provides micro-benchmarks which are not representative to production applications and there is a missing understanding to how network congestion impacts application traffic.

In the literature there is no benchmark to stress the network and I/O concurrently, GPCNeT makes use of an incast traffic pattern to replicate I/O traffic which may not be representative as these patterns often neglect to factor a read/write time associated with performing the I/O.

2.2.2 Performance Prediction

Performance predictions allow architects to understand performance characteristics of systems prior to the implementation. These typically come in two methods, analytical models and simulation.

Analytical

An analytic model is a mathematical representation of time required for some operation. This operation for HPC applications is typically the longest part of the execution or the cricitcal path of an application.

Common approaches to analytical network models include Bulk Synchronous Parallel (BSP) [126], LogP [39, 40] and LogGP [9].

The BSP model operates across *supersteps*, these steps consists of computation and a communication phase. At the start of a *superstep* synchronisation takes place then the computation can take place. If the data required is on a remote node then it must be retrieved during a previous *superstep.* In a communication phase then exchange data with other nodes.

The LogP model uses 4 parameters to model communication time:

- L - Communication Delay (flight time of message - latency)
- o - Communication Overhead (time taken to transmit/recevie a message)
- g - gap (minmimum time between consecutive messages)
- P - number of processors

The LogP model assumes all messages are small, and in 1997 Alexandrov suggested the addition of a new parameter to cater for longer messages, called the LogGP model. The new parameter is the time *Gap per byte* and is the time taken to transmit a byte on the network.

In 1993 Adeve et al. showed that the performance of a parallel application could be modelled outside of strict modelling frameworks by taking a generic approach (see equation 2.4) [6].

$$T_{total} = (T_{computation} + T_{communication} + T_{overlap}) + T_{synchronisation} + T_{overhead} \tag{2.4}$$

In Equation 2.4 overlapping time from the communication and computation is taken in to account with $T_{overlap}$. Resource contention and system overhead are considered independently in the model ($T_{overhead}$) and any cost associated to synchronisation during the application run is catered with $T_{synchronisation}$. Each cost is calculated by either benchmarking, or timing sections of a code.

Once disadvantage of an analytical model is that there is an an underlying assumption that blocks of computation and synchronisation that place at the same time in lockstep and are deterministic which may not always be the case.

The use of reusable models has become prevalent with the development of a plug and play wavefront model [100]. This reduces the model development time, and increases the applicability of the model.

Simulation

Simulation is the recreation of some experimental setup with a computer based representation. This is usually performed with a simulator, these fall in to two categories Discrete-event driven and trace driven [72]. In Discreete-event simulations events are typically added to a queue and processed in chronological order. In contrast trace-driven simulations replay a trace of recorded events in which the order of execution is sequential.

There are limitations with a trace-driven simulation, these include:

- Scalability - traces must be created on a physical system
- Flexibility - traces can not be modified

There are many simulators covering different HPC system subsystems such as CPU, Memory and IO. Some notable examples of interconnect simulators include; SST/Macro [79], CODES [38], WARPP [66] and TraceR [71].

SST/Macro provides a macro simulation of the network using an analytical (MACRELS) and packet model (PISCES). MACRELS makes the assumption that data is moved as a single chunk while PISCES models individual packets moving through the network [79]. The packet model provides two levels of simulation: simple which assumes that packet flits travel as a single unit and are not separated; and cut-through which allows for the separation of flits but provides an aggregated latency/bandwidth approximation for flits similar to MACRELS.

CODES [38] was designed to explore design questions around large-scale storage systems. It is built on top of the Rensselaer Optimistic Simulation System (ROSS) [29] which is a parallel discrete event simulation framework similar to SST. CODES has been developed to have a tightly coupled network model and this has been validated against an IBM BlueGene/L. A key feature of CODES is that, while it was designed for storage systems, it can be used to model collective performance for different topologies [98].

TraceR is a scalable simulator built on top of the parallel discrete event capabilities of CODES and ROSS allowing for scalable packet-level simulations of applications [71]. This is done by replaying communication traces through the simulation stack.

WARPP is macro simulation toolkit that predicts application runtime for applications by allowing you to recreate a communication pattern for the simulator and playing this through a system configuration. WARPP allows the modelling of core-to-core, socket-to-socket, and node-to-node configurations which are representative of modern HPC systems [66]. WARPP achieves these by using latency measurements of existing systems to approximate the timing for sending individual messages.

Limitations of Current Modelling Techniques

Analytical models fail to capture the advanced features that are present in modern low latency interconnection networks, for example if we consider two packets traversing a network that are freely available to be routed adaptively one packet could take the minimum path while the second could have additional hops in the network. In the case of a LogP or LogGP model then this would equate to differing values for o (the communication overhead).

Using simulation as a modelling technique allows for advanced network features to be modelling and accurately but this will only be the case if the simulator can reflect the crossbar and the input and output queues. SST/Macro provides some of this functionality but does not provide a cycle accurate router model which can cause implications when using this to design future networks. Given this it is impossible to model applications in a multi-user scenario.

One other area for concern with these modelling techniques is the validation process, while it is a good assumption to measure the latency between the links and compare against the simulation yet this fails to capture all the nuances of a system such as the memory latency and the host to NIC bus.

2.3 Low Latency Interconnect Design

This section explains the different aspects of a low latency interconnection network, it provides a description on the software communication stack that sits on top of the underlying hardware.

2.3.1 Communication Stack

Figure 2.3 shows these different layers, similar to the Open Systems Interconnection (OSI) model.

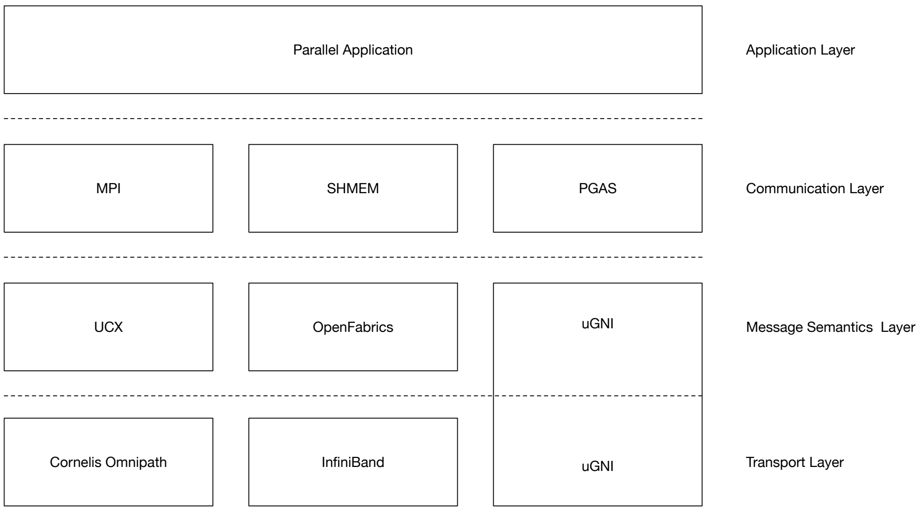


Figure 2.3: Distributed Application Communication Stack

The application layer represents the parallel application. This is designed to exploit the computational resources. This application is built with a communication library such as MPI or SHMEM. The communication library interfaces with the NIC drivers either through the kernel or through an additional library such as OpenFabrics Interfaces (OFI) libfabric [60], UCX [115] or Cray's uGNI [107]. Data is then transmitted over the network.

2.3.2 Network Architecture

In this section we consider the hardware aspects of network design. This falls in to three key areas; firstly the network topology, secondly flow control and finally routing algorithms.

Figure 2.4 outlines an interconnection network. These networks differ from other Local Area Network (LAN) in the fact that they do not use a shared communication medium, instead each link is independent and can communicate without requiring to take control of the transmission medium. Examples of shared medium networks include Ethernet [3] and Wi-Fi [4].

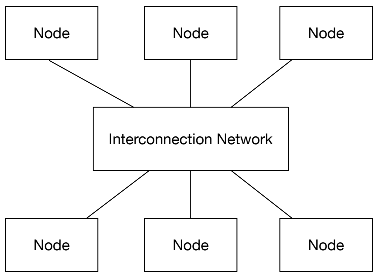


Figure 2.4: Network Architecture

In the OSI model layer 2 frames are transported over the physical layer, multiple frames make up a packet at layer 3. In the case of low latency networks the smallest form of data movement is a flow control digit (flit). A physical digit (phit) is the smallest unit of data processed by a switch; multiple phits are combined to form a flit [44].

Figure 2.5 shows the basic architecture of a simplistic switch. Each of the ports has a input buffer and an output buffer, data then flows across the crossbar. This is common across most interconnects for example Cray Aries [54], Intel Omni-Path [18] (now branded as Cornelis Omni-Path) and InfiniBand [1]. The switch radix refers to the number of ports available on a switch; for example a NVIDIA Mellanox InfiniBand Switch SB7800 consists of 36[2]; therefore has a switch radix of 36. The Cray Aries router has a switch radix of 64 [54].

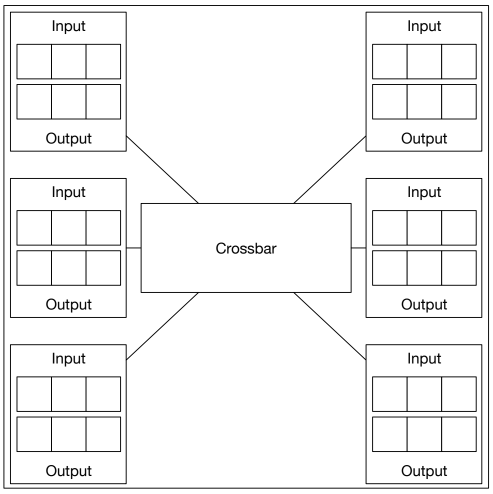


Figure 2.5: Switch Architecture

Topologies

There are several key network topologies employed in low latency networks. The most common is the Fat-Tree [83].

Fat-Trees (also known as a folded Clos) are shaped similarly to a tree, and have root switches and leaf switches (see Figure 2.6). Typical implementations are 2 or 3 level and may be tapered between the root and leaf switches, reducing cost and increasing the number of available endpoints but at the expense of global bandwidth.

The worst case hop count is shown in Equation 2.5. If the left most node and the right most node want to communicate then it must go through each layer up and down in the tree but only hits the root switch once, this is only considered as 1 hop.

[2]See `https://www.mellanox.com/related-docs/prod_ib_switch_systems/pb_sb7800.pdf`

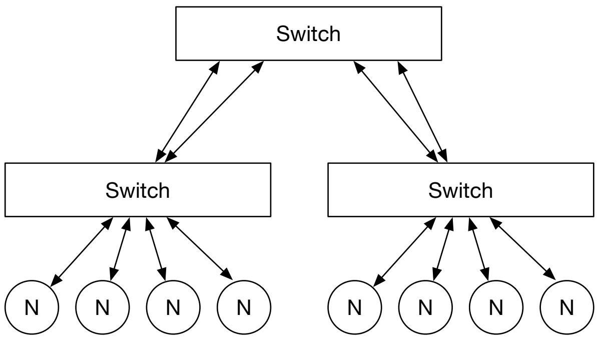


Figure 2.6: Fat-Tree Topology

$$\text{Worst Case Hop Count} = (2 \times \text{Number of Levels}) - 1 \tag{2.5}$$

Dragonfly is another network topology [76]; one key advantage of this topology is that this topology can be more cost effective for more endpoints than a Fat-Tree [76]. Networks based upon a Dragonfly are grouped in to three ranks, firstly the router, secondly a intra-group and thirdly inter-groups (see Figure 2.7). Figure 2.7 shows four nodes per router and three routers per group configuration with one global link per router for inter-group communication. The design of inter-group networks are left to the implementer, in the case of Cray Aries this is an All-To-All 2D mesh while NVIDIA's *Dragonfly+* implementation utilises a tapered Fat-Tree for the intra-group topology [116].

The Aries implementation from Cray of a Dragonfly utilises 4 nodes to 1 router and 96 routers per group connected as an All-To-All mesh electrically; the inter-group connections are optical and can be tapered reducing cost [54].

The average hop count for a Dragonfly is 5, this is where the node needs to communicate with a node from another group. The worst case could be higher than a fat tree for a sufficiently large network because the adaptive routing could make the data travel over multiple routers.

Torus are multidimensional topologies based upon a mesh or a cube network (k-ary n-mesh/cube network).

In 1977 Sullivan et al. document the first implementation of an n-cube net-

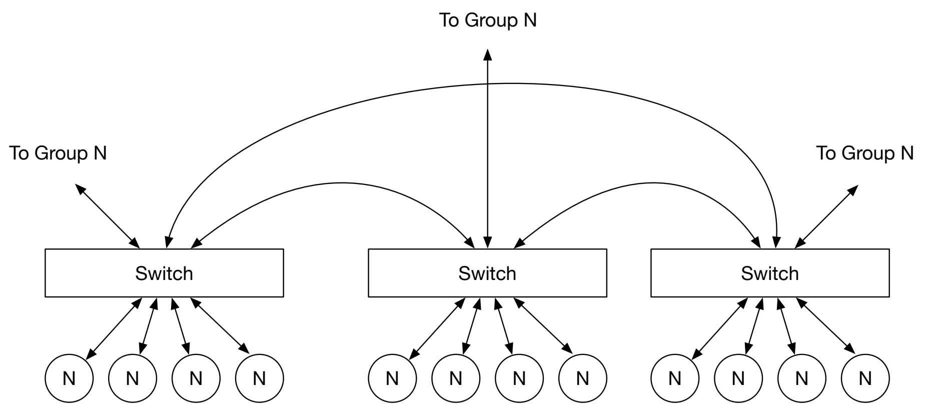


Figure 2.7: Dragonfly Topology - Group All-To-All

work [119, 120]. The requirement for this network design came from the performance constraint of one word being transmitted every instruction. Sullivan et al. show that the average propagation delay is $\frac{n}{2}$, with low link utiilisation. Figure 2.8 shows a 2D torus, in a 3x3 grid, a switch (s in the figure) can then have multiple nodes attached.

More recent implementations of a torus topology include the Cray SeaStar [26, 8]. This features a 3D Torus, common configurations were 25x16x24 in size as seen in Titan [19].

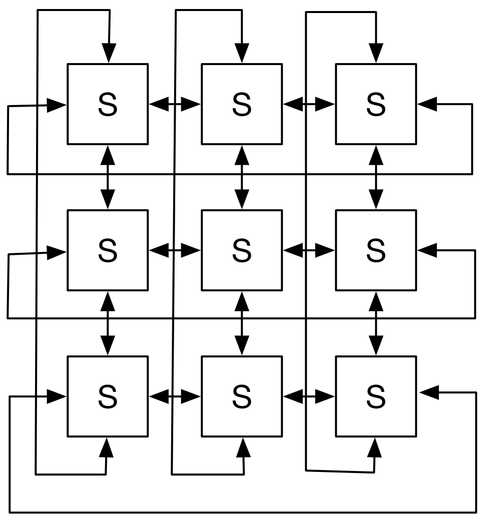


Figure 2.8: Torus Topology, S is a Switch in which multiple nodes can reside

HyperX is an alternative to DragonFly topology for providing a high number of endpoints while reducing cabling and router count [7]. There are currently no commercially available implementations of a HyperX network. Figure 2.9 shows a 2D HyperX configuration, again multiple terminals can be attached to the switches (s in the diagram).

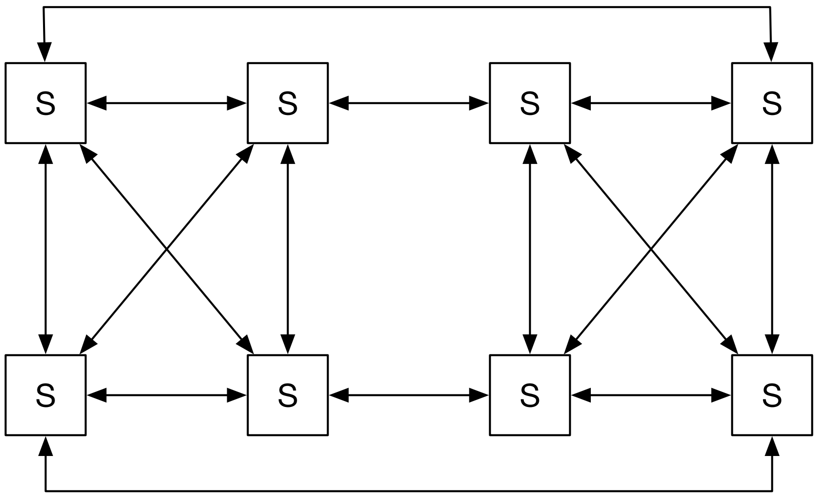


Figure 2.9: HyperX Topology, S is a Switch in which multiple nodes can reside

Flow Control Mechanisms

As low latency networks do not share a medium the transmission of data is controlled by other means. Flow control can be broken down in to two subcategories; buffered and bufferless. Bufferless flow control is the simplest flow control; packets are forwarded is resources are available or dropped/misrouted in the case where resources are not available. Buffered flow control stores the data until resources become available.

Buffered flow control can perform on both packets and flits. Store-and-forward and cut-through methods are examples of packet buffered flow control. Store-and-forward stores the packet until the all of the packet is received and then forwards it [44].

The latency of the packet is given in Equation 2.6.

$$T = Ht_r + H\left(\frac{L}{b}\right) \tag{2.6}$$

Cut-through does not wait for all of the packet to be received instead starts transmitting when resources become available [75]. This is sometimes known as virtual cut-through flow control and should not be confused with virtual channel flow control discussed later on. This approach reduces the latency for the packet compared to the store and forward method; Equation 2.7 shows the serialisation latency for cut-through.

$$T = Ht_r + \frac{L}{b} \tag{2.7}$$

Flit based buffered flow control methods include Wormhole Routing [43] and Virtual Channel Flow [42]. While Wormhole Routing contains routing in the title it is a flow control method. Wormhole routing works much like the packet based approach cut-through yet does not require the storage space of a packet just a flit. Issues arise when the output is blocked because there is insufficient resource available to forward the flit.

Virtual Channel Flow control provides multiple Virtual Channel (VC) per single physical channel overcoming the blocking problems of Wormhole as a flit can just utilise another virtual channel [42]. This approach improves channel bandwidth utilisation as it is not held idle while waiting to forward a blocked packet.

When buffering data at a switch, the switch must also know of the state if upstream buffers, this is controlled by one of three methods; credit-based flow control, on/off flow control and ack/nack flow control. In credit-based flow control the upstream router keeps count of the available flit buffers downstream. This count is decremented when a buffer is used; once this reaches 0 it must wait for the downstream switch to credit the upstream switch when it has forwarded a flit and a VC has become available [44]. For on/off flow control buffer state is handled with a control signal; when the flit has been received and the number of buffers drops below a free buffers threshold the signal is turned off [44]. Once the number of free buffers rises above the threshold for re-enabling it is turned back on. The threshold for turning off the control signal should be greater than the relationship between time of additional flits being sent/received (t_{rt}) after the signal changed state and the length (L_f) of the flit in bits (see Equation 2.8). This ensures that the buffers do not overflow.

$$F_{off} \geq \frac{t_{rt}b}{L_f} \tag{2.8}$$

In the case of ack/nack flow control a switch simply forwards a flit as it does not store upstream information. If the downstream switch has a buffer available then it replies with an Acknowledgement (ACK) [44]. If no buffer space is available then the flit is dropped and the downstream switch replies with a Negative Acknowledgement (NACK). This approach inefficient for both buffer space and bandwidth, due to flits being resent and have to be held on the upstream switch.

Routing Algorithms

Routing algorithms can either be oblivious or adaptive. Oblivious routing algorithms include Valiant's Routing algorithm [127] and minimal oblivious routing [101]. Oblivious routing algorithms route traffic irrespective of network load, compared to adaptive routing algorithms which will change the path data takes. In minimal oblivious routing traffic always takes the shortest path, in comparison Valiant's algorithm chooses a path at random to forward traffic.

Adaptive routing allows the traffic to be routed across different routes on a per flit basis based upon current state information, similar to changing the journey of a car due to traffic. Universal Globally Adaptive Load-balanced Routing (UGAL) is one approach to adaptive routing, traffic with this routing algorithm either take the minimal path or a non-minimal path [77, 108]. Congestion is avoided with this algorithm by using Valiant Load-balanced routing [125]. UGAL is commonly seen on Dragonfly topologies rather than a fat tree where minimal static routing provides the best performance.

2.4 Summary

This chapter has introduced types of parallelisation and the concept of speedup for parallel applications. Subtopics of performance engineering (benchmarking and performance modelling) have been introduced, the current state of the art for performance modelling and benchmarking has been discussed. The building blocks of a low latency interconnection network have been explained, which is further motivates the work presented in this thesis.

This thesis moves the state of the art forward for benchmarking by introducing a new flexible network and I/O benchmark called StressBench that allows for communication patterns to be concurrently across a system. Simulation is progressed with a reusable validation technique that been used to validate four systems and build performance models for two applications. Application performance predictions are then generated for a multi-user shared system demonstrating the usefulness this has on system procurement and network design.

CHAPTER 3
Compute Platforms and Applications

The hardware and software marry together to make the simulation of physics possible. In this chapter we discuss the hardware specifications of the computing resources used and the in depth view of the applications used throughout this thesis.

This chapter is as follows:

- Section 3.1 outlines the hardware configurations of the systems used throughout this thesis.
- Section 3.2 discusses the two primary applications used throughout this thesis in detail.
- Section 3.3 demonstrates common MPI implementations for communication patterns used by the applications mentioned in Section 3.2.

3.1 Compute Platforms

Modern HPC systems are typically interconnected servers which offer computation, storage and management. These servers are typically referred to as a node or blade. A node usually has 3 key elements, a CPU, RAM and NIC. The storage of a single node is typically irrelevant given that file systems are typically shared amongst all nodes. Common parallel filesystems include General Parallel File System (GPFS) [114] and Lustre [25].

Throughout this thesis the following compute platforms have been used extensively. These platforms differ in size, stage of lifecycle, CPU architecture as well as network architecture. The oldest system in use is Tinis, entering production in October 2015 and the newest system is Isambard entering production in November 2018.

The machines used in this thesis are significant because they cover a wide spectrum of what are considered supercomputers rather than a standalone compute server, from small clusters (Orac) to large machines such as Cori which was number 5 in the top500 when it entered production in 2016 [122]. One notable feature of these machines is that smaller machines typically use a Fat Tree network topology while larger machines may use a Dragonfly.

In the UK there are two tiers of HPC resources used by both academia and industry. Firstly we have the Tier 1 machine, currently called Archer2. This is one of the largest systems in the UK which features 5,860 nodes. Tier 2 machines are typically smaller in the 64-512 node sizes, for example Isambard. Universities may have their own smaller clusters such as Orac and Tinis. Given the large machine sizes there are typically resource limits dictated to stop resources being starved by large jobs. In the case for Tinis a user can request 32 nodes at one time without a reservation, where as on Archer2 the maximum job size is 1024 nodes. These constraints can be used to aid procurement decisions and scheduling requirements on a machine, for example on Cori the job scheduler will not allocate job sizes less than a group (384 nodes) to multiple groups as this significantly slows down the applications [37].

3.1.1 Tinis

Tinis is a 212 node cluster featuring Intel Haswell (HSW) CPU and an InfiniBand interconnect. This machine is based at the University of Warwick. Table 3.1 shows the hardware specifications for Tinis. The systems is built using Lenovo NeXtScale nx360 M5 servers.

The software stack used on Tinis is GCC 8.3 and OpenMPI v4.0.3. The job scheduler for this system is Slurm.

Table 3.1: Tinis Specification

Component	
CPU	2 x Intel Xeon E5-2630 v3 2.4 GHz (Haswell)
Memory Per Node	64GB
Network	QLogic TrueScale InfiniBand (QDR)
Network Topology	2-Level Tapered Fat Tree (2:1)
Parallel Filesystem	GPFS

3.1.2 Orac

Orac is a 84 node at the University of Warwick featuring Intel Broadwell (BDW) CPUs and an Intel Omni-Path low latency network. The specifications for Orac can be seen in Table 3.2. Orac is constructed from Lenovo NeXtScale nx360 M5 servers.

Table 3.2: Orac Specification

Component	
CPU	2 x Intel Xeon E5-2680 v4 (Broadwell) 2.4 GHz
Memory Per Node	128GB
Network	Intel Omni-Path
Network Topology	2-Level Tapered Fat Tree (2:1)
Parallel Filesystem	GPFS

The Intel compiler (v2020.4.304) was used for Orac with Intel MPI 2019 (2019.9.304).

3.1.3 Isambard

Isambard is a tier 2 machine based on the Cray XC series; this machine makes use of the Marvell ThunderX2 CPU. This system uses XC50 blades. It features 329 nodes residing in 1 Dragonfly group.

OpenMPI 4.0.3 was used on Isambard, compiled with GCC using the Cray Programming Environment (9.0.6).

Table 3.3: Isambard Specification

Component	
CPU	2 x 32-core Marvell ThunderX2 2.1 GHz
Memory Per Node	Phase 1: 256GB Phase 2: 512GB
Network	Cray Aries
Network Topology	Dragonfly
Parallel Filesystem	Lustre

3.1.4 Cori

Cori (based at National Energy Research Scientific Computing Center (NERSC)) is a Cray XC40 System featuring both HSW and Knights Landing (KNL) CPU; the machine totaling 12,076 nodes (2388 HSW, and 9688 KNL nodes) makes use of 34 Dragonfly groups.

This split system design is similar to other large scale machines such as Trinity [51].

Table 3.4: Cori HSW Specification

Component	
CPU	2 x Intel Xeon Processor E5-2698 v3 2.3 GHz
Memory Per Node	128GB
Network	Cray Aries
Network Topology	Dragonfly
Parallel Filesystem	Lustre

Table 3.5: Cori KNL Specification

Component	
CPU	Intel Xeon Phi Processor 7250 1.4GHz
Memory Per Node	96GB DDR4 & 16GB MCDRAM
Network	Cray Aries
Network Topology	Dragonfly
Parallel Filesystem	Lustre

3.1.5 Astra

Astra (based at Sandia National Laboratories) was the first petascale ARM machine utilising the ThunderX2 CPU from Marvell.

Table 3.6: Astra Specification

Component	
CPU	2 x 32-core Marvell ThunderX2 2 GHz
Memory Per Node	128GB
Network	Mellanox Infiniband (EDR)
Network Topology	3-Level Tapered Fat Tree (2:1)
Parallel Filesystem	Lustre

The compiler used on Astra was GCC (v9.0) and OpenMPI (v4.0.1) was used for the MPI library.

3.2 Applications

Applications for HPC systems take many forms and solve different problems, although at the heart of these applications lay some primitive communication patterns.

The two applications (TeaLeaf and Sweep3D) chosen for this thesis contribute to a large proportion of the runtime of production codes at Atomic Weapons Establishment (AWE) and the DOE. In the case of some production applications at AWE the time solving equations using linear solvers can be 20-50% of the overall runtime, as linear solvers are strong scaled they become predominantly communication bound which limits the scale due to the communication overhead [35]. The performance of sparse linear solvers limits the size, fidelity and quality of the computational results [11]. The wavefront communication pattern used by Sweep3D can consume as much as 50-80% of the runtime for some production applications used both AWE and DOE. Given the high percentage of runtime and scaling issues, it is prudent to focus on these applications so that improvements can be seen as networks continue to evolve.

The message profiles for the communication patterns inside of TeaLeaf and Sweep3D are deterministic and fixed during initialisation. They do not change as the applications progresses as is the case with some Adaptive Mesh Refinement algorithms or an unstructured mesh applications.

The communication patterns used in this applications are not just used in multi-physics applications but other applications such as a Halo Exchange is used in GROMACS [128] which is a molecular dynamics application which can model chemical bonding interactions.

The decomposition schemes used in these applications leads to a regular computation/communication pipeline that are deterministic with regular synchronisation points such as calculating a residual in a linear solver. This makes them easy to model as there is very little load imbalanced generated as the problem is evenly distributed across all the computational resources.

3.2.1 TeaLeaf

TeaLeaf is a proxy application for solving the heat conduction equations [94] it provides four iterative solvers. TeaLeaf solves the heat conduction equations in both 2D and 3D using a 5 and 7 point stencil respectively. The temperatures are cell-centred. The linear solvers provided include a Conjugate Gradient (CG), Jacobi, Chebyshev and a communication avoiding CG algorithm.

Linear Solvers typically solve systems in the form:

$$Ax = B \tag{3.1}$$

Algorithm 1 describes a preconditioned conjugate gradient algorithm. This preconditioned algorithm solves $b - Ap_0 = r_i$. The first residual reduction (r_0) is calculated, then carries on calculating until it has converged. The solution is then x_i. The preconditioner is the matrix M^{-1} which is applied to the residual reduction [112].

Practically this algorithm can be broken down to two key communication patterns; firstly a reduction for example the calculating of the residual needs to be performed across all ranks. The second is a halo exchange in which boundary data is exchanged after a calculation.

As this type of application scales it becomes predominately communication

Algorithm 1 Preconditioned Conjugate Gradient Algorithm

```
  $r_0 := b - Ax_0$
  $z_0 := M^{-1}r_0$
  $p_0 := z_o$
  for $i = 1, 2, ...$ until convergence do
      $\alpha_i := \frac{r_i \bullet z_i}{Ap_i \bullet p_i}$
      $x_{i+1} := x_i + \alpha_i p_i$
      $r_{i+1} := r_i - \alpha_i A p_i$
      $z_{i+1} := M^{-1}r_{i+1}$
      $\beta_i := \frac{r_{i+1} \bullet z_{i+1}}{r_i \bullet z_i}$
      $p_{i+1} := z_{i+1} + \beta_i p_i$
  end for
```

bound due to decreasing computation intensity across the MPI ranks.

Five input problems sets are provided called benchmarks, these vary in size from a small 10x10 grid to a 4000x4000 grid size. Typical problem sizes ran in production for linear solvers vary from 500x500 (Benchmark 3) through to 4000x4000 (Benchmark 5).

3.2.2 Sweep3D

Sweep3D is a discrete ordinates transport code [80, 100, 53], solving the multi-group Boltzmann transport equation. Sweep3D operates over three dimensions (N_x, N_y, N_z) which are decomposed over a 2D processor array (n $\times$ m). This decomposition occurs over the X and Y dimensions, each process receives all of the Z dimension (see Figure 3.1).

The algorithm employed by Sweep3D can be seen in Algorithm 2. The most north east tile has to wait for the all other tiles to be processed. In total 8 sweeps through the arrays are required for Sweep3D, one for each of the vertices of the 3D cube. Sweep3D performance has been improved by the inclusion of the blocking factor, this allows a group of tiles to be computed prior to the communication phase.

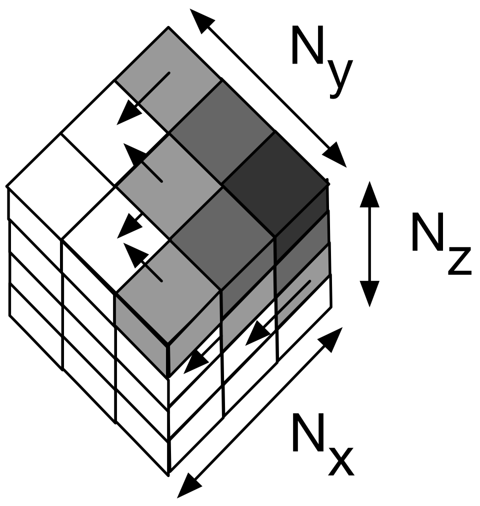


Figure 3.1: Three Dimensional Wavefront Decomposition

Algorithm 2 Sweep3D Algorithm

```
for Each energy group do
    for Each Sweep do
        for Each Angle Block do
            for Tile Block in Z do
                Receive form West Neighbour
                Receive form South Neighbour
                for Each Angle do
                    Compute Tile
                end for
                Send to East Neighbour
                Send to North Neighbour
            end for
        end for
    end for
end for
```

3.3 Communication Patterns

This section briefly describes some of the common communication patterns commonly used in parallel applications. These patterns form the foundations to the distributed parallel applications described in Section 3.2.

These communication patterns can be constructed as small *motifs*. *Motifs* are small runnable communication patterns that are typically taken from applications and can be run in isolation without the overhead of having to run the entire application. Sections 3.3.1, 3.3.2 and 3.3.3 explain the communication patterns with code snippets on how they are performed in MPI.

3.3.1 Halo Exchange

A halo exchange exchanges the data at the boundary cells. An implementation may typically vary the number of rows/columns that are exchanged. Figure 3.2 shows a structured 2D halo exchange with a depth of 2 (green cells are exchanged with neighbours).

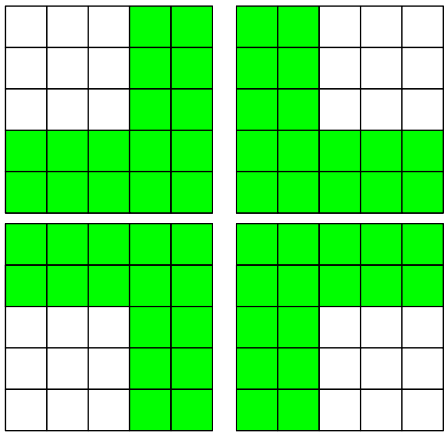


Figure 3.2: Halo Exchange

A typical blocking MPI implementation for a 2D halo exchange can be seen in Listing 3.1.

An unstructured halo exchange is similar but the boundary lengths are not uniform (depending on the partitioning scheme used). The data accessed may not be from contiguous memory which has performance implications, MPI provides functionality to address some of these non-contiguous memory access through the creation of custom MPI types which can handle the memory addressing cleanly.

```
if(y_up) {
    MPI_Status status;
    MPI_Send(y_buffer, length, MPI_DOUBLE, y_up, 1, MPI_COMM_WORLD);
    MPI_Recv(y_buffer, length, MPI_DOUBLE, y_up, 1, MPI_COMM_WORLD, &status);
}
if(y_down) {
    MPI_Status status;
    MPI_Send(y_buffer, length, MPI_DOUBLE, y_down, 1, MPI_COMM_WORLD);
    MPI_Recv(y_buffer, length, MPI_DOUBLE, y_down, 1, MPI_COMM_WORLD, &status);
}

if(x_left) {
    MPI_Status status;
    MPI_Send(x_buffer, length, MPI_DOUBLE, x_left, 1, MPI_COMM_WORLD);
    MPI_Recv(x_buffer, length, MPI_DOUBLE, x_left, 1, MPI_COMM_WORLD, &status);
}
if(x_right) {
    MPI_Status status;
    MPI_Send(x_buffer, length, MPI_DOUBLE, x_right, 1, MPI_COMM_WORLD);
    MPI_Recv(x_buffer, length, MPI_DOUBLE, x_right, 1, MPI_COMM_WORLD, &status);
}
```

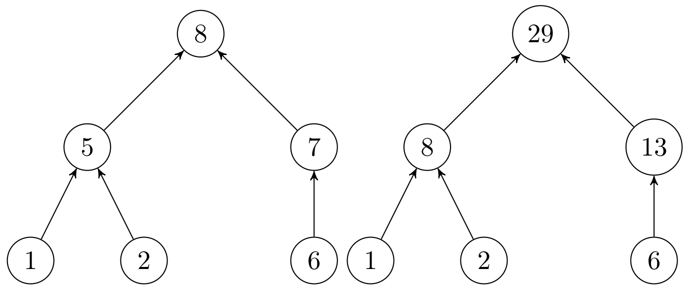
Listing 3.1: Example 2D Halo Exchange

3.3.2 Reduction

A reduction combines the elements in an array with a specific operation. The MPI standard provides standard operations (e.g. minimum, maximum, summation). MPI implementations typically provide multiple algorithms for performing these operations, such as recursive doubling or a binary tree. These implementations have an impact the performance of the operation as some generate more communication traffic.

Figure 3.3 shows how values propagate up a binary tree to perform a reduction.

An AllReduce is a modification to this in which the result is broadcast to all



(a) Starting Values (b) Computed Results

Figure 3.3: Reduction - Binary Tree Example

MPI ranks when calculated. In this thesis an AllReduce is tuned to be a Binary Tree Reduction and Broadcast.

3.3.3 Wavefront

In the case of Sweep3D the communication pattern requires the result from a previous cell prior to calculate its own value.

Listing 3.2 demonstrates the sweep across one octant of a 3D cube. A check is made to see if the neighbouring cell is an edge, if not it will wait until the data has been received before progressing to perform the tile computation and then finally send the data on to its neighbours.

Wavefronts are seen in other types of applications in use at the DOE, such as Computational Fluid Dynamics (CFD) and linear solver applications [97].

3.4 Summary

This chapter has covered the hardware platforms that are used throughout this thesis. This chapter has also provided an explanation of the software applications used and how they are implemented with MPI.

The patterns discussed in this chapter form the basis of many parallel applications and are applicable to many scientific applications.

```
for (int k = 0; k < z_dimension; k += kba) {
  if (xDown > -1) {
    MPI_Recv(xRecvBuffer, nx_len, MPI_DOUBLE, xDown, 1000,
    MPI_COMM_WORLD, &status);
  }

  if (yDown > -1) {
    MPI_Recv(yRecvBuffer, ny_len, MPI_DOUBLE, yDown, 1000,
    MPI_COMM_WORLD, &status);
  }

  physics(x,y,z);

  if (xUp > -1) {
    MPI_Send(xSendBuffer, nx_len, MPI_DOUBLE, xUp, 1000,
    MPI_COMM_WORLD);
  }

  if (yUp > -1) {
    MPI_Send(ySendBuffer, ny_len, MPI_DOUBLE, yUp, 1000,
    MPI_COMM_WORLD);
  }
}
```

Listing 3.2: Example Octant Sweep

CHAPTER 4
Design and Implementation of a modern network benchmark

Predicting the performance of supercomputers is vitally important in evaluating their suitability for applications and for informing the procurement process. Time to solution is usually the primary metric of consideration, and can be impacted by OS jitter [46], network contention [16], and resource allocation [15]. While there exist a variety of simulators focused on modeling the computational aspects of supercomputers, consideration of the networking infrastructure has been less thorough. Since network contention can cause variability in communication time [37]; it is prudent to develop a benchmarking tool that is capable of reproducing traffic patterns commonly seen in scientific applications, thereby allowing for more faithful replication of these workloads on new and existing machines. By benchmarking systems using higher level communication patterns, such a benchmarking tool enables applications to be evaluated on a variety of architectures without source code being released, which is beneficial in the case of commercially sensitive and/or restricted codes, where the underlying application architecture cannot be exposed. Common MPI benchmark tools either take a generic approach to network congestion [36] or focus on performance of individual MPI operations [69, 104]. This generic approach to network congestion may not be representitive of what can be expected from a shared multi-user system.

Understanding the interactions and impact between applications on a multi-user system can be used to improve both resource scheduling and allocation; and the communication patterns themselves for example the development of communication avoiding algorithms such as those used within linear solver applications [30].

In this chapter we address these shortcomings with the development of a novel network replication framework called StressBench, that is capable of executing complex communication patterns concurrently and reproducing application workflows.

Specifically, this chapter documents the development of a customisable network and I/O benchmarking tool that uses traffic patterns to evaluate architectures; evaluates the tool against commonly used network microbenchmarks and validate application workflow replication with four proxy applications all within 20% difference; replicates a full system run and use this to demonstrate the impact of network contention on the time-to-solution of multiple proxy applications. Finally, we extend our full system replication to present a novel case study assessing the communication performance while in contention with common I/O strategies.

This chapter is as follows Section 4.1 outlines the design of a modern MPI benchmark for evaluating network architectures; Section 4.2 demonstrates replay functionality inside of StressBench and Section 4.3 looks at performing performance studies looking at how I/O traffic interferes with application communication traffic.

4.1 StressBench Design

StressBench was designed to be flexible and applicable to all parallel workloads; this is done through a portable interface. This allows for extension and additions to the communication patterns. Multiple communication patterns can be chained together to create a *job* which can resemble a production application. A communication pattern is applied in three phases:

Decomposition In this phase the proxy application breaks up the global MPI communicator world into the relevant MPI groups which each have multiple communication patterns associated with them. Once the MPI groups have been constructed the communication patterns themselves perform a decomposition if required to establish their nearest neighbours in the case of a halo exchange.

Perform During the perform phase the communication patterns execute as if they were a standalone application using their MPI communicator to communicate. Each pattern is timed individually and the total job is timed.

Cleanup The cleanup phase allows the patterns to safely clean up any resources that have been consumed. Each job also collates the timings from each of its group's ranks and then prints these to standard out.

These three phases are separated by global barriers to ensure they begin at the same time such that the patterns under test are controlled tightly to ensure that they are performed concurrently.

Listing 4.1 shows an example TeaLeaf iteration with I/O write after. Inputs for StressBench must take the following form: *Job Name*, *Node List* and then a list of *motifs*.

```
[JOB_NAME] TeaLeaf_CG
[NID_LIST] 6,9,17,18,33,41,58,67,72,75,83,84,87,90,102,103
[MOTIF] AllReduce
[MOTIF] Compute -m 350000
[MOTIF] AllReduce
[MOTIF] Compute -m 350000
[MOTIF] halo2d -x 4000 -y 4000
[MOTIF] MPIIO -s 1500000 -i 1 -m 256 -f <file_path> -n
    MPI_File_write_all
```

Listing 4.1: Example Input

Figure 4.1 shows the architecture for StressBench. Job creation is the construction of the motifs in to a *job*. The job list is given to StressBench which executes each of the motifs consecutively per job and the jobs concurrently. The three phases of a motif are globally barriered and then the results aggregation occurs during the clean up.

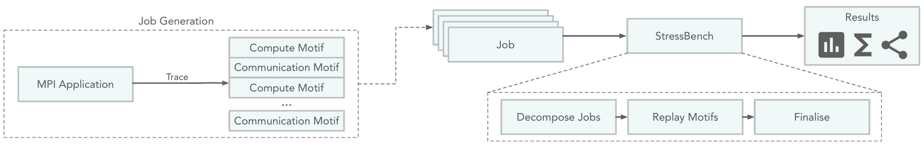


Figure 4.1: StressBench Architecture

4.1.1 Motifs

Patterns can be written by providing implementations for each of the key phases of a given proxy application. The example communication patterns are taken from mini-applications and can be seen in production applications. As these patterns try to interact with the network (a shared resource) contention increases which can degrade application performance. These examples have been taken from the following proxy applications; TeaLeaf [94], CloverLeaf [87], Sweep3D [80], LULESH [74] and Hardware/Hybrid Accelerated Cosmology Code (HACC) [63].

A “compute” pattern is provided so that more intricate workloads can be built. The compute pattern includes the capability to emulate load imbalance; this has been achieved by generating a value from a specified distribution.

The design of the emulated patterns depends on the communications and computation pipeline. In the case of a TeaLeaf there is no overlap between the communications and computation pipeline. For Sweep3D the communication and computation pipeline are tightly coupled so therefore the computation has to be integrated in to the design of the motif. By understanding the communication and computation pipeline the information can then be built in to a motif by providing implementations to the 3 phases. For the implementation of the 2D halo-exchange the problem decomposition was extracted from TeaLeaf and then rebuilt inside of the decompose functionality. The perform functionality involved inspecting how the 2D halo-exchanges take place and replicating the MPI calls based upon the provided communicator. In the deletion phase all buffers used are freed; this is independent of the pattern being replicated.

Multiple motifs have already been implemented in StressBench:

Halo Exchanges A 2D structured halo exchange and 3D unstructured halo exchange are provided.

AllReduce Support for two reduction operations; sum and minimum operations.

Computation A computation motif is provided to emulate computation. A distribution can be provided to generate a load imbalance.

Incast A file I/O motif providing N-1 communications.

AllToAll An AllToAll pattern is provided in the default package.

Sweep3D A Sweep3D motif is provided offering a Sweep communications pattern.

PingPong A PingPong style motif is also provided for measuring the latency while in contention.

I/O Motifs

I/O motifs have been integrated in to StressBench to allow I/O subsystem to be benchmarked simultaneously to the network. Thereby giving the ability to understand the interactions between I/O traffic and MPI application traffic; while these may be configured to avoid interaction the underlying network has fixed resources that are shared by both types of traffic.

Currently two I/O strategies are implemented, namely N-1 and N-N. These two approaches are the most widely used within HPC applications. StressBench uses N-1 for reading input files.

Currently these two I/O strategies can be executed through HDF5, MPI-IO or POSIX file operations. Writes are typically of more interest than reads but both have been developed for StressBench [28].

The I/O bandwidth reported by the motifs is calculated with Equation (4.1).

$$BW = \frac{\text{Bytes Read/Written}}{\text{Time Taken}} \tag{4.1}$$

In an effort to validate this implementation, StressBench was run with Darshan profiling the I/O operations. The cumulative timings were taken from the Darshan log for the MPI-IO operations and compared to the cumulative timings from the the output of the motif. The 'MPIIO_F_WRITE_TIME' counter from the Darshan log was used for the comparison. The difference in the timings was less than 0.5% for 1GB files and less than 0.01% for 10GB files. This difference is caused by two factors. Firstly, the resolution of timers and secondly the position in which the timer is placed. Darshan provides wrappers around the I/O function calls which insert the timers within the call while the MPI-IO motif places the timing calls around the I/O function call which includes the calls to capture the information for Darshan; resulting in a marginal difference between the times reported by Darshan and MPI-IO motif.

Traditional MPI benchmarks focus on peak performance for example IMB and OSU are designed to run on a quiet system.

One of the motifs built inside of StressBench is AllPingPong which runs from 0 to 4MB message sizes in the same way as PingPong in IMB and Latency in OSU. This motif was used in the comparison against IMB and OSU. It is possible to measure the latency of a specific message size inside of StressBench.

The default compiler and linker flags were used for building the microbenchmark suites. In the case of StressBench the default optimisation level is -O0. The latency benchmarks were scheduled to use one core across two nodes; each of these values were repeated 10 times across different days to establish the average latency the benchmark may achieve. This is because adaptive routing may change the path that the traffic follows and increase the latency.

Figure 4.2 shows how traditional PingPong benchmarks compare against StressBench, for Cori, Isambard and Tinis.

StressBench performed similarly to the existing MPI microbenchmarks such as PingPong resulting in slight increase in the returned latencies.

For Tinis the average increase in the reported latency was 3.9% (maximum 11.4%) for IMB and 2.75% (maximum 8.2%) for OSU. On Isambard this average difference in the latency was a decrease of -7.9% (maximum -26.3%) for IMB and -29% (maximum -49.2%) for OSU. The large variability on Isambard comes from the adaptive routing inside of the network as each PingPong message may take a different route to reach the desired endpoint.

Stressbench demonstrates a negligible difference between traditional microbenchmarks thus is a suitable replacement for these microbenchmarks.

4.2 Application Replay Functionality

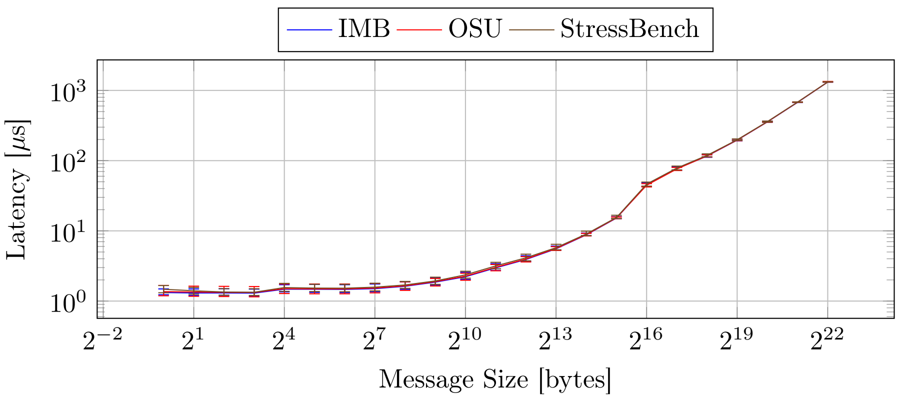
To build a representative workload we extracted key characteristics from a proxy application flow in order to replicate these patterns with StressBench. TeaLeaf is a linear solver proxy application that has a variety of solvers.

The proxy application is instrumented to capture computation timings, using Caliper [21]. Caliper allows for application's source code to be annotated and records snapshots during application execution.

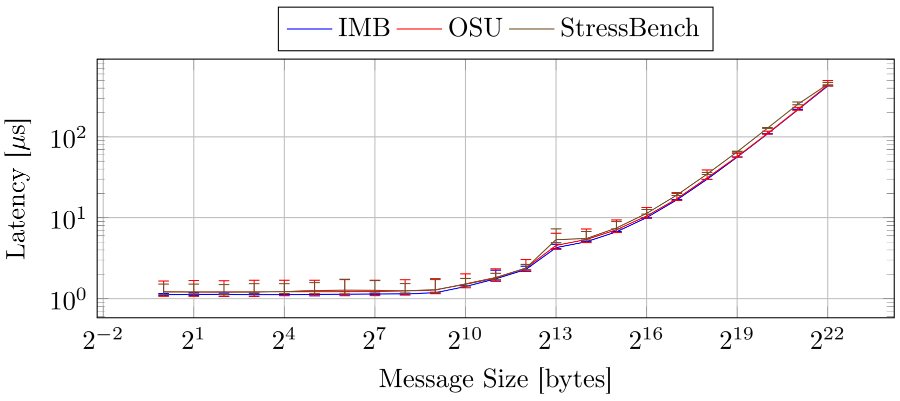
The motifs can be rebuilt either independently such as TeaLeaf in which the CG iteration is replicated as all reduces; computation and the halo exchange. In the case of the Sweep communications we have coupled the computation with the communications and it results in one motif.

To verify the communication patterns match they were traced with Intel Trace Analyzer and Collector (ITAC) and the point-to-point message profiles captured. To further validate the workload we compare the times captured inside of StressBench with the timings from the proxy application. The compute timings were generated using a Gaussian distribution function parameterised to model OS jitter.

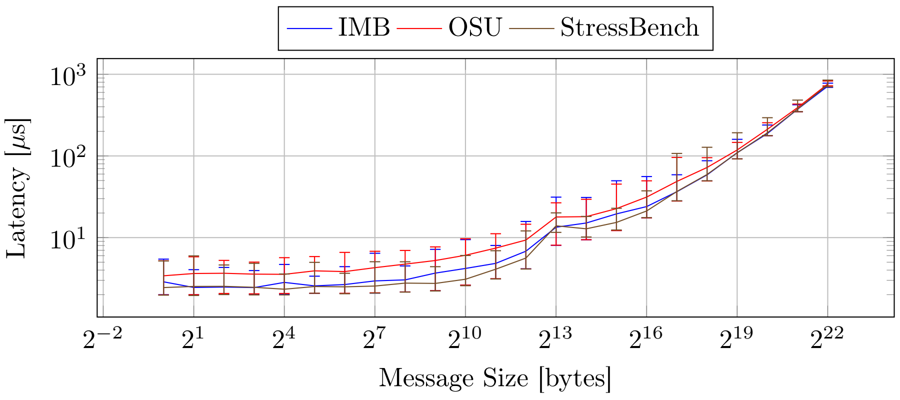
In the case of TeaLeaf we emulate one Conjugate Gradient (CG) iteration. Benchmark (BM) 5 was chosen for the selected problem and has been strong-scaled. This problem is the crooked pipe problem in which a pipe has a lower density than its surroundings and therefore heat travels faster through this part of the problem domain. A conjugate gradient iteration in TeaLeaf consists of two



(a) Tinis



(b) Cori



(c) Isambard

Figure 4.2: PingPong Comparisons for Three Machines

reductions with computation and a 2D halo exchange with post computation; this was confirmed by tracing the application with Intel ITAC and reading through the application source code.

Figure 4.3 shows how a Conjugate Gradient Iteration inside TeaLeaf compares with StressBench. The difference in the measured and emulated runtimes for TeaLeaf was less than 10% difference. When the communication patterns were traced with Intel ITAC the message profiles did not differ. The computational motif runtime provided less than 1% of the variability compared to the measured TeaLeaf run. The large variations came from the communications notably the halo exchange. In our emulation the message packing was treated as additional computation rather than part of the communication directly. This was chosen because the message packing and unpacking occurs after the communication has taken place and the memory is sent/received from contiguous blocks. When TeaLeaf is strong scaled like this at larger scales the communications can dominate the execution of an iteration; the computation roughly halves as MPI ranks double.

The message profile shows how many bytes were sent and received by each of the MPI ranks. The message profile from the halo exchange matched for the measured and emulated, Figure 4.4. These plots show the message sizes match the real communication pattern and the *motif*, any difference in the plots would indicate the communication patterns not matching.

The largest difference in runtime between the proxy application and the emulation is less than 10% for Tinis in the comparison.

Sweep3D is a discrete ordinates transport code [80, 100]. We have emulated a typical application run which consists of 12 iterations. The problem was weak scaled for each rank to have a grid size of 40x40x100. The measured compute time was 61us per octant.

Figure 4.6 shows how there is no difference in the message profiles between Sweep3D and the *motif*.

Figure 4.5 shows how StressBench compares to the weak-scaled input deck. The average difference was 3.2% with the largest difference 6% for 16 nodes. As the input deck was weak scaled; we felt this was the best approach to use

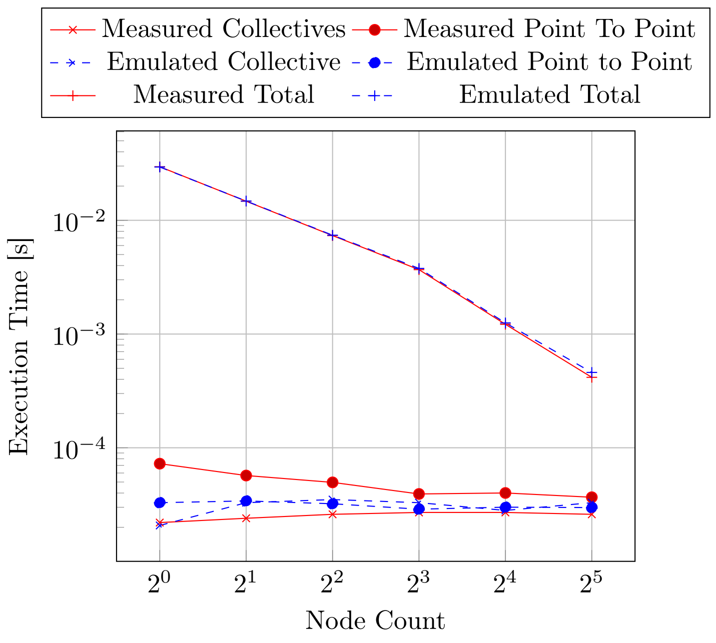


Figure 4.3: TeaLeaf Runtime Validation

the average computation time. The point-to-point message profiles for both the measured and emulation matched when compared.

LULESH is a 3D Unstructured Lagrangian Explicit Shock Hydrodynamics proxy application [74]. As with TeaLeaf we have emulated one iteration; we chose to emulate the 50th iteration to provide some warm up iterations. We have performed a weak scaling study across a mesh of 81^3. The computation time was 0.81s for all runs. The maximum difference was 15% between the measured and emulated; this was for the a single node with 8 MPI ranks. For multi-node runs the average difference was less than 5%. Figure 4.7 shows the comparison between the runtimes for the measured and emulated results. This demonstrates that StressBench can be used to effectively emulate communication patterns found in production applications.

Figure 4.8 displays the message profiles for the 50th iteration of LULESH and the *motif*.

SWFFT is a Fast Fourier Transform (FFT) proxy application. This type of communication pattern features heavily in astronomy related simulation codes

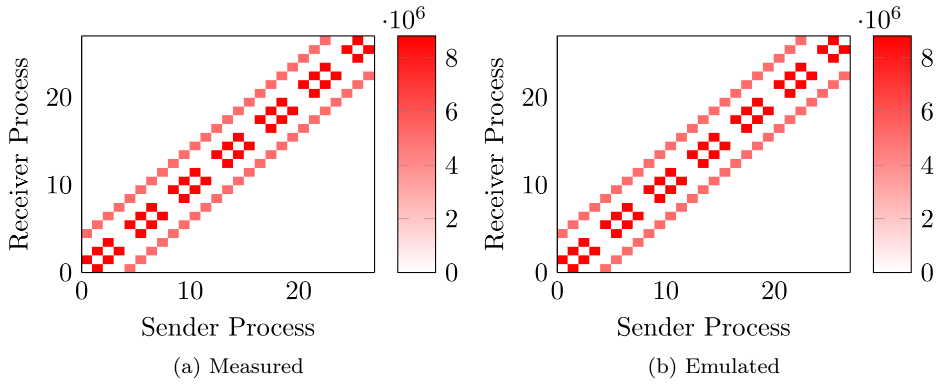


Figure 4.4: Measured and Emulated Communication Pattern for Halo2D

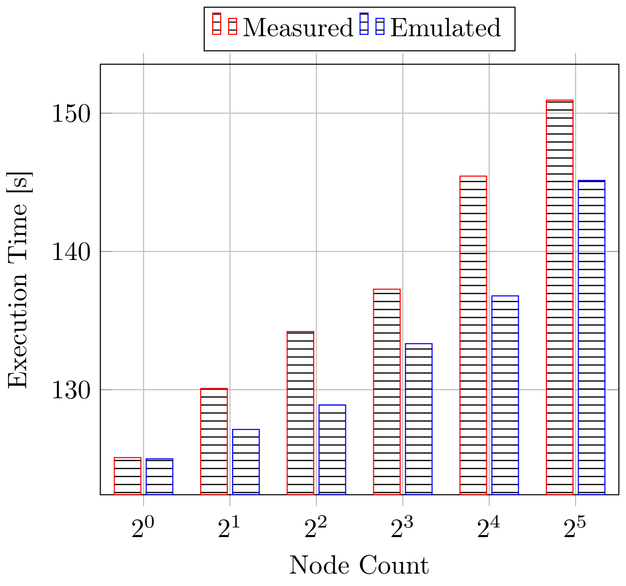


Figure 4.5: Sweep Runtime Validation

for example Hardware/Hybrid Accelerated Cosmology Code (HACC) [63]. The FFT implementation in SWFFT is from HACC; this operates on 1D FFT steps which are interleaved as transposition and sequential steps. This approach reduces communication overhead. We have emulated a forwards FFT and backwards FFT during the emulation of the pattern. The MPI communications were traced and the number of bytes for the point-to-point communications were recorded. Figure 4.9 shows that the point-to-point message sizes for the real and emulated runs match; as does the send/receive processes. The mea-

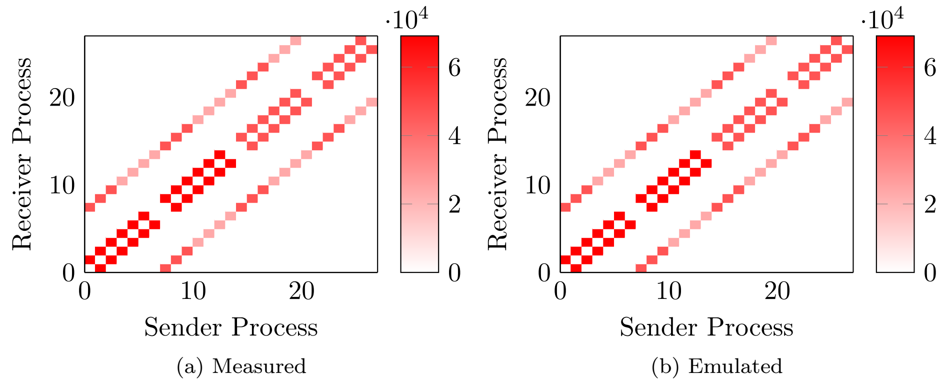


Figure 4.6: Measured and Emulated Communication Pattern for Sweep3D

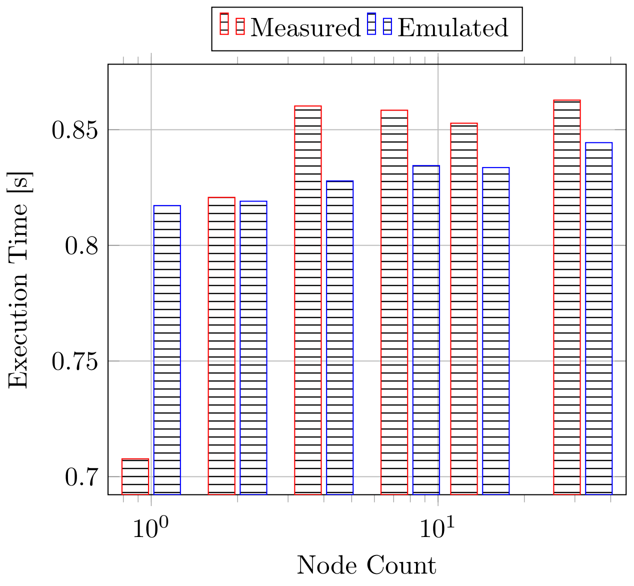
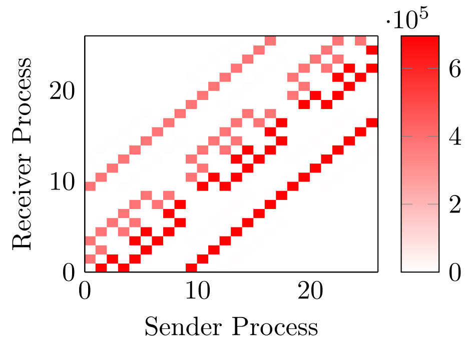


Figure 4.7: LULESH Runtime Validation

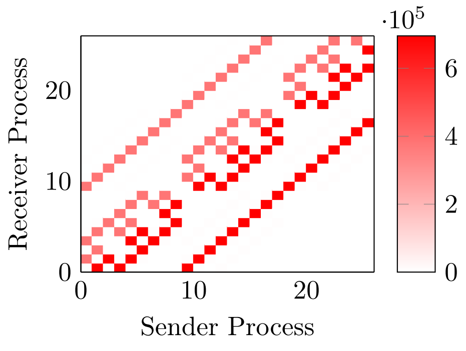
sured and emulated runtimes for SWFFT differed by as much as 11.4% for a strong scaled problem of 160x160x160. The average error was -7.8% for this problem, results can be seen in Table 4.1.

4.2.1 Applications Inside of StressBench

Building an application in to StressBench is simpler than constructing patterns as the application can be treated as an Application Programming Interface (API) which can be called. Given the proposed portable interface for

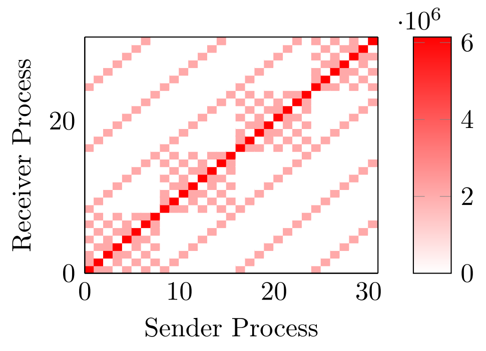


(a) Measured

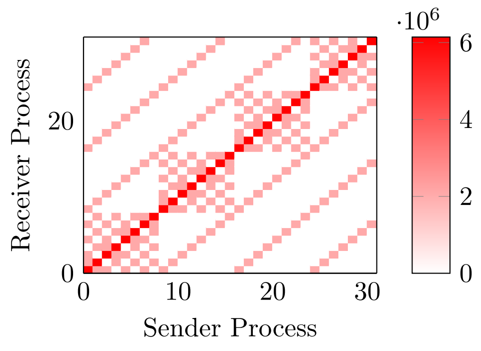


(b) Emulated

Figure 4.8: Measured and Emulated Communication Pattern for LULESH



(a) Measured



(b) Emulated

Figure 4.9: Measured and Emulated Communication Pattern for SWFFT

StressBench (section 4.1) the API calls (to the application) can be called in the relevant phases. The easiest applications to interface with StressBench are applications developed in C as there are no type differences. In section 4.2.1 we show how a C application is interfaced with StressBench and in section 4.2.1 we show how a C++ application has been interfaced with StressBench.

C Application

TeaLeaf was chosen as a C application to interface with StressBench. The TeaLeaf application needed to modified to store the communicator provided from StressBench, this was added to the Settings structure. The communication routines were then changed to use the communicator stored rather than the

Table 4.1: SWFFT Runtime Validation

Nodes	Measured	Emulated	Difference
1	0.1383	0.1340	-3.1045
2	0.0555	0.0530	-4.4551
4	0.0357	0.0316	-11.3894
8	0.0190	0.0180	-5.4324
16	0.0266	0.0250	-6.1350
32	0.0263	0.0253	-3.8023

MPI_ COMM_ World.

Another modification to TeaLeaf was the addition to support a different input and output files, by default TeaLeaf looks for a file called 'tea.in' for the input and outputs to 'tea.out'. This allows different problem sizes for each job within StressBench.

The application is then usable as any other motif, Listings 4.2 shows how to the input for running the TeaLeaf application in StressBench.

```
[JOB_NAME] TeaLeaf
[NID_LIST] 0,1,2,3
[MOTIF] TeaLeaf -i /home/user/tealeaf/tea.in -o /home/user/tealeaf/
    tea.out
```

Listing 4.2: StressBench Input for TeaLeaf

To validate this implementation still operated as before with no overhead each problem set from TeaLeaf was run across 4 nodes on Isambard. Each of the problems validated against the solutions contained within 'tea.problems', the computed solutions matched for the application run independently and the application running in StressBench. Figure 4.10 shows how the runtime of the application and the reported times from StressBench compare, there is a negligible difference between the time reported by the application and StressBench's course-grained aggregated timing. The average time is presented from five runs within StressBench for each of the problem sets, with the minimum and maximum times as the error bars.

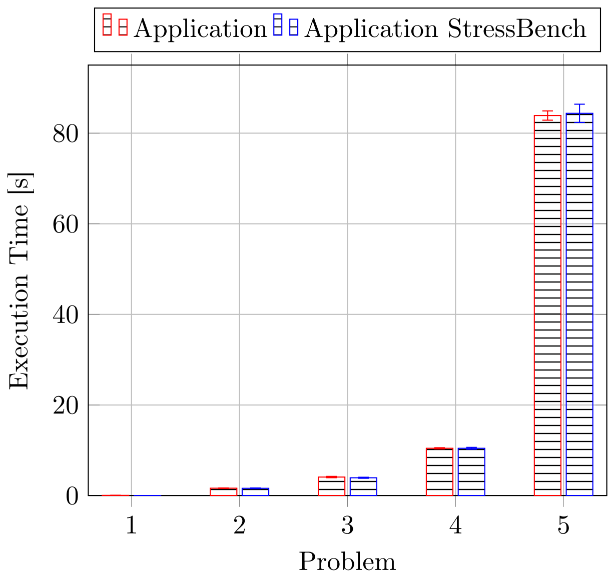


Figure 4.10: TeaLeaf Application Problem Sizes

C++ Application

CloverLeaf [87] was chosen as the C++ application to integrate with StressBench, because the patterns are similar to those in the I/O study in Section 4.3.2. CloverLeaf is a two dimensional Lagrangian-Eulerian explicit hydrodynamics proxy application [87]. Interfacing with a C++ application requires a C/C++ interface to be developed so that StressBench can call the functions from the application. The C/C++ interface implemented allows the functions and objects within the C++ application to be treated as C functions.

Again with this application the communicator from StressBench was stored and used in the communicator functions. CloverLeaf already supports different input and output files and did not require modification like TeaLeaf.

To validate the implementation 5 datasets were run through the CloverLeaf application and the CloverLeaf implementation in StressBench. This was performed on Isambard across 4 nodes, the comparison of the runtimes can be seen in Figure 4.11, similar to the TeaLeaf timings there is no significant difference in the runtimes when running the application through StressBench.

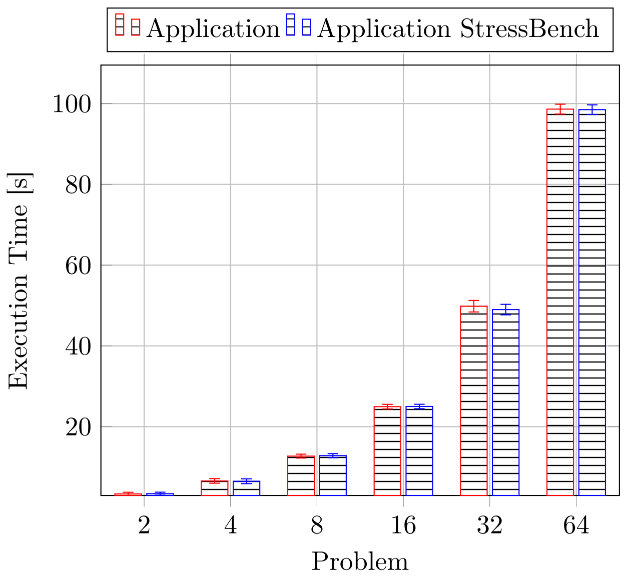


Figure 4.11: CloverLeaf Application Problem Sizes

4.3 Application Communication Traffic and I/O Performance Studies

This section uses StressBench to investigate two scenarios, firstly a full system workload is constructed and run across a cluster orchestrated with StressBench. The second performs an I/O study looking at the effects of I/O traffic on communication patterns.

4.3.1 Full System Orchestration

The validated applications can be composed to mimic a representative system workload. The mimicked workload allows explorations into potential slowdowns as a result of job placement and communication interactions affecting performance.

The workload we have examined utilised the communication patterns in TeaLeaf and Sweep. Applications often perform I/O to either checkpoint or to output a visualisation [50].

Existing benchmark suites [36, 113] use an incast like communication pattern to replicate I/O traffic. While this communication pattern can induce similar network traffic it often lacks the ability to replicate I/O bandwidth which means that it could artificially clear the network. When evaluating a mimicked workload we have used both Incast traffic and real I/O traffic through the use of the Incast and I/O motifs.

I/O traffic patterns were added to these workloads after a number of iterations to emulate this. The size of the I/O has been approximated using Equation (4.2), where domain size is the problem size in the case of TeaLeaf it is 4000×4000.

$$\text{I/O Message Size} = \frac{\text{Domain Size} \times \text{variables}}{\text{MPI Ranks}} \tag{4.2}$$

The system was randomly distributed to have eight jobs of sixteen nodes each; two Sweep3D with an Incast, two TeaLeaf with an Incast, two repeated file I/O and finally two AllToAll Traffic. The problem domain for Sweep3D was configured as 100^3 cube and 5 iterations. The blocking factor for the Z dimension was set to 10 and Incast message size was set to 7,111,111 bytes per rank.

For TeaLeaf the problem was configured similarly to that used in the above validation: a domain size of 4000×4000; five iterations were performed with a incast motif at the end having a message size of 1,500,000.

A representative Incast size of 480,469 bytes per MPI rank was configured for five iterations.

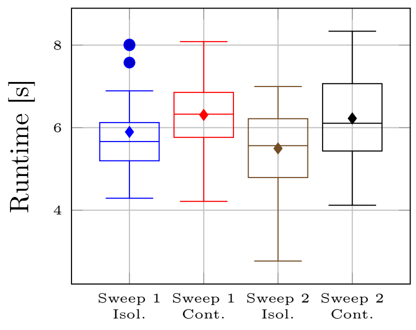
The file I/O was performed using an 'MPI_File_write_all' call with the MPI-IO motif.

Figure 4.12 shows how each of the jobs performed in isolation and under contention on Tinis. The mean is shown with the diamond inside of the box plots and the line represents the median. The distance between the whiskers is the Inter-Quartile Range (IQR), the value for the upper and lower whiskers using Equation 4.3. Outliers are represented as coloured shapes outside of the whiskers. The boxplots show that with the network under contention these runtimes generally shift upwards and the boxplots elongates showing wider variability in the time to solution for the jobs. Each *job* was run in isolation for 20 runs under the same resource allocation of 128 nodes with all other nodes not being provided jobs. All jobs were then combined to execute concurrently across the 128 jobs 30 times. All these runs were preformed on a system during a maintenance window to ensure no external factors affected communication times for the workload.

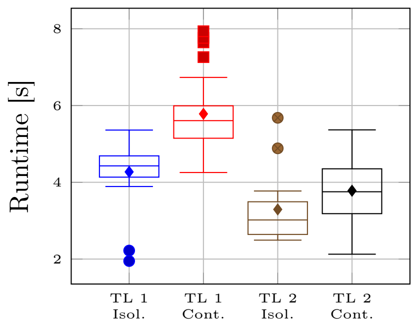
$$\text{Whisker} = \text{Quartile} \pm 1.5 \times \text{IQR} \tag{4.3}$$

For Sweep3D on Tinis the worst case slowdown was 1.2×; for TeaLeaf we show a 1.8× slowdown when the applications are trying to run in contention with other applications on the system. There are some anomalies in which the traffic under contention could under some circumstances perform faster than isolation, such as the *job* TL2. This is because the network contention is a temporal and one of the contending jobs could have slowed down due to CPU throttling creating a load imbalance and TL2 being able to transmit optimally, the messages from the application do not have to wait for resources across the network. The static routing deployed in the Fat Tree inside of Tinis means that the outliers are primarily caused by the aforementioned reasons.

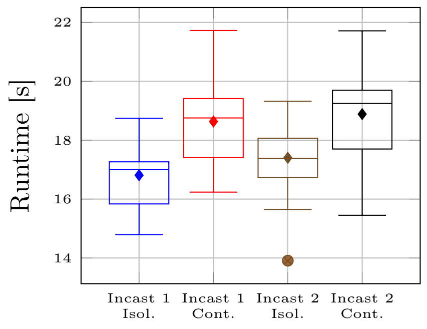
Figure 4.13 shows the how Sweep3D and TeaLeaf performed in isolation and under contention on Isambard. The slowdown worst case slowdown for Sweep3D was 1.02, TeaLeaf 1.02 and AllToAll was 1.2. Outliers for the jobs running on Isambard are of more interest, because the network makes use of adaptive routing [54]. The adaptive routing algorithm employed by Aries adaptively on a per packet basis, so packets part of the same message may not take the same path and can arrive out of order. This means that for the larger the messages (such as the AllToAll) could take many different paths, which sees benefit for this *job* but increases the communication times for other jobs as this application begins to congest all the links across the network.



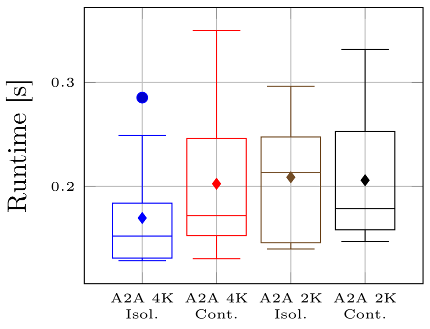
(a) Sweep Comparison



(b) TeaLeaf Comparison



(c) Incast Comparison



(d) All-To-All Comparison

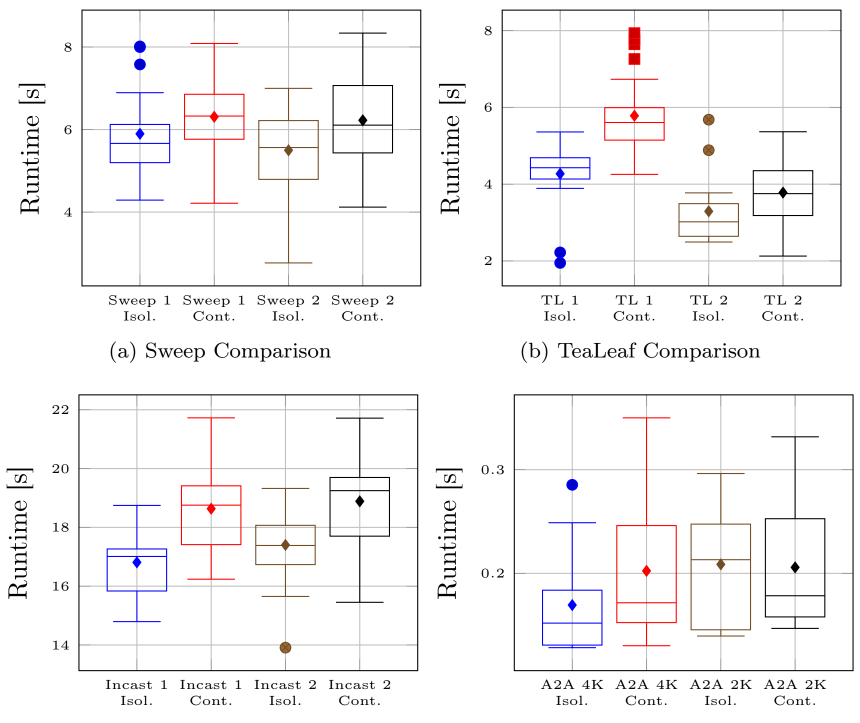
Figure 4.12: Application Runtimes in Isolation and in Contention on Tinis

Chunduri et al. present a congestion impact (CI) metric [36], shown in Equation (4.4).

$$\mathrm{CI} = \frac{\mathrm{t_{congested}}}{\mathrm{t_{isolated}}} \tag{4.4}$$

As discussed previously, an Incast pattern can provide a similar communication pattern to file I/O. As such we ran the same jobs with the Incast motif rather than an MPI-IO Write motif. Table 4.2 shows how the CI differs between Incast and the use of file I/O on both systems. It is clear that the impact of using real file I/O is greater than using an Incast motif.

Due to the transient nature of the traffic hotspots it is possible that messages are unaffected by network contention resulting in no congestion impact for motifs.



(a) Sweep Comparison

(b) TeaLeaf Comparison

(c) Incast Comparison

(d) All-To-All Comparison

Figure 4.13: Application Runtimes in Isolation and in Contention on Isambard

Table 4.2: Comparison of CI for Incast and File I/O for Applications

	Tinis		Isambard	
Pattern	Incast	MPIIO	Incast	MPIIO
Sweep 1	1.0068	1.0744	1.0146	1.4664
Sweep 2	1.0010	1.1356	1.0331	1.4266
TeaLeaf 1	1.0172	1.4094	1.0157	1.7936
TeaLeaf 2	1.0707	1.1345	1.0196	1.4392
Incast 1	1.0088	1.1314	1.0004	1.6109
Incast 2	1.0426	1.1161	1.0021	1.6868
All To All 2K	1.0157	1.1917	1.0137	1.0327
All To All 4K	1.0587	1.0000	1.2327	1.0014

4.3.2 I/O Study

This study looks at the interactions between I/O traffic and application traffic, and the performance degradation of both of these with StressBench.

There is a presumption that I/O traffic interferes with application traffic yet no such study exists quantifying this interference. As such we have designed this study to cover breadth rather than depth into a specific interaction. The study focuses on some common communication patterns from applications; we use the validated patterns mentioned above.

To understand the interactions between application communications and I/O traffic we have run a range of application patterns against some large file sizes which would cause congestion inside of the network. In order to ascertain suitable file sizes we analysed the file sizes on four storage systems at NERSC. The data was collected using RobinHood policy engine [48] and inserting the POSIX information in to a mySQL database [85, 59]. The data provided consisted of the size of files as reported by the inode for each files [59]. We grouped the data in to 5 buckets: 0GB, 1GB, 10GB, 100GB and 500GB. Figure 4.14 shows how many files are situated in each of the 5 groups.

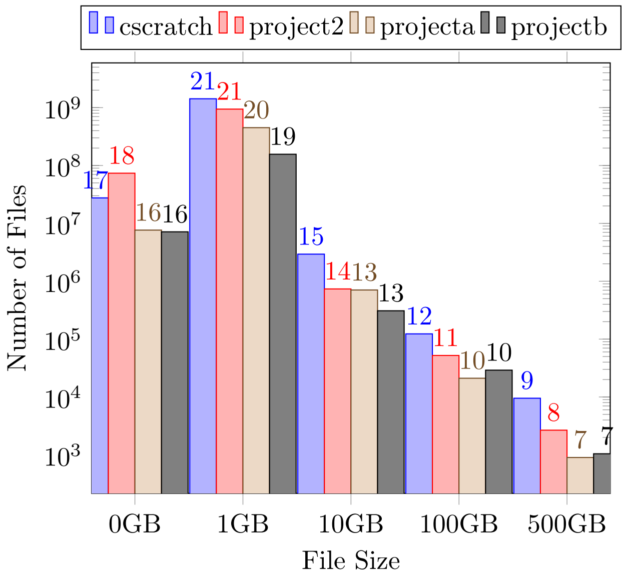


Figure 4.14: Measured File Sizes From Four Production Storage Systems

In an effort to negate background network noise the runs performed the pattern in isolation and then in contention; each job then completed this 10 times. This was done to ensure that the isolated and contended runs performed as close as possible to each other such that they would have an equivalent background noise. These jobs were repeated at differing times across a week to achieve best and worst case background network noise. Each motif in the run was configured to run for at least 30 seconds so that the network can get fully congested and links can be exhausted. This ensures that any adaptive routing algorithms has time to take affect on communication patterns. Previous work has shown that system load can interfere with latency sensitive messages [35].

The study not only looked at the the effects of pattern and file size but also the effects on job placement. Three job placement schemes were used; linear, interleaved and random; Figure 4.15 shows how the three placement schemes differ.

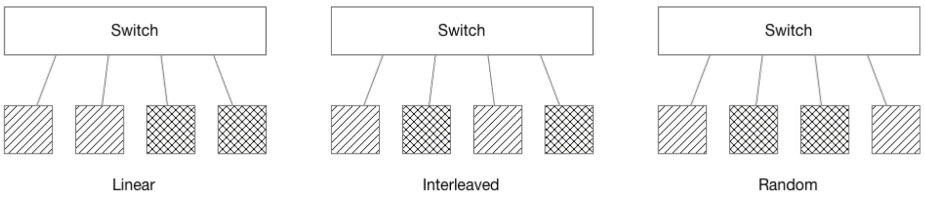


Figure 4.15: Job Placement, Diagonal and Hash lines represent different applications

For runs on Tinis 32 nodes were utilised while on Isambard 256 node runs were used. Both sets of runs consisted of a 50:50 split between the pattern of interested and offending I/O traffic.

To assess the performance degradation we compare the CI. Table 4.3 shows how the CI differs against the four file sizes tested for Tinis. Table 4.4 shows how the CI differs against the four file sizes tested for Isambard.

In the case of Tinis the file size seems to have negligible difference in the impact on the performance rather the job locality has a greater impact. This is due to the Fat tree network topology deployed in Tinis. Traffic that can be routed between nodes across the same switch are unlikely to suffer because of the file I/O traffic, such as the linear job placement. This results in a slowdown of communication time thus increasing the application runtime.

I/O traffic generated on Isambard has a greater effect on application communication traffic (shown in Table 4.4). This is most notable with a linear placement; with this network it is also observed that I/O traffic size impacts the application communication traffic.

Table 4.3: Comparison of Application Workload against Congestion Impact - Tinis

		AllReduce	Halo Exchange	Sweep3D
Interleaved	1GB	1.026	1.006	1.001
	10GB	1.154	1.007	1.000
	100GB	1.172	1.003	1.006
	500GB	1.222	1.004	1.001
Linear	1GB	1.028	1.006	1.000
	10GB	1.023	1.006	1.002
	100GB	1.015	1.004	1.000
	500GB	1.024	1.000	1.002
Random	1GB	1.015	1.008	1.002
	10GB	1.132	1.000	1.004
	100GB	1.152	1.010	1.003
	500GB	1.135	1.008	1.001

Table 4.4: Comparison of Application Workload against Congestion Impact - Isambard

		AllReduce	Halo Exchange	Sweep3D
Interleaved	1GB	1.000	1.100	1.001
	10GB	1.000	1.015	1.036
	100GB	1.122	1.018	1.070
	500GB	1.145	1.059	1.045
Linear	1GB	1.031	2.026	1.189
	10GB	1.050	1.159	1.139
	100GB	1.104	1.125	1.286
	500GB	1.212	1.157	1.118
Random	1GB	1.101	1.073	1.001
	10GB	1.119	1.096	1.046
	100GB	1.262	1.138	1.064
	500GB	1.304	1.137	1.199

4.4 Summary

In this chapter the development of a modern reconfigurable network benchmark built on top of MPI has been presented. The approach presented allows for domain complexity to be abstracted away so that the underlying network performance can be studied using real-world communication patterns. The patterns being studied can be implemented with MPI directly requiring no external infrastructure. These patterns can then be connected together to look at how applications utilise the network while in contention with other communication patterns.

By chaining multiple motifs together applications can easily be replicated within StressBench, we demonstrate that runtime differences are less than 15% for a variety of applications. Applications can also be plugged in to the flexible to run applications instead of separate motifs.

The presented orchestration of several applications running concurrently shows that StressBench is a suitable tool for evaluating network performance; with applications such as TeaLeaf running 1.4× on a fat tree slower while in contention with other applications. We also show that real I/O traffic has a greater impact on application communication performance resulting in larger slowdowns when compared to Incast like application traffic. These slow downs results longer time to solutions and more computation resources being used to facilitate the runs as wallclock times are increased to ensure jobs complete. This can impact on research budget where the computation resource is fixed.

We have shown how on systems with adaptive routing that file size affects the performance rather than the placement of the jobs; while on systems with static routing such as those in fat trees we have shown that job location is more likely to cause issues with contention. To mitigate this contention jobs could be scheduled to avoid being placed linearly and instead interleaved; this may improve the performance of the application communication patterns.

Possible extensions to this work include more validated communication patterns from other scientific disciplines and machine learning applications, as well as the addition of I/O libraries to allow for better replication of applications.

CHAPTER 5
Validation of a Network Micro-Simulator

As large scale systems increase in heterogeneity it is becoming more important to consider the various components and their contributions to bottlenecks within systems. Modeling efforts have previously lacked fidelity in aspects such as topological awareness, as is the case for LogP [39, 40]. Analytical modeling techniques such as these are severely limited due to the significant assumptions made in arriving at results, and as such, they pose a challenge for exploring the design space for interconnection of supercomputers for the Exascale era.

Another approach for such exploration is simulation, which falls in to two distinct categories: macro- and cycle accurate simulation. A macro-simulation, similarly to an analytical model, makes assumptions about the behaviour of the various components in the system, but offers more configurability, hence allowing for enhanced exploration of the design space than an analytical model. Cycle accurate simulators are typically developed by network vendors and simulate networks in detail down to moving each FLIT through the network. The major trade-off between these two types of simulator come down to accuracy vs time. While cycle-accurate simulators are accurate they are often slow; compared to macro-simulators/analytical models that are considerably faster.

While macro simulators largely make assumptions for system parameters, some customisation can be applied to better reflect system characteristics when evaluating how changes might impact performance. One notable case study looks at the jitter between two operating systems and feeds this into the macro simulator WARPP to evaluate application system performance at scale [65].

In this chapter we present a micro-simulator for simulating interconnection networks and validation with microbenchmarks across three systems. Two application performance models are constructed on top of the system models. Finally this chapter looks at pushing the system models in larger networks looking at the performance and cost tradeoffs of larger switches.

This chapter is structured as follows, Section 5.1 describes the simulator built on top of SST. Section 5.2 shows the validation for the hardware platforms and the performance models for Sweep3D and TeaLeaf. Section 5.3 speculates about how design of fat trees and dragonfly networks influences designs.

5.1 Simulator Design

The design of SST compliments the development of a network simulator as an interconnected stack; similar to a real HPC system. For this four components that interlink and connect to provide network simulation where each component represents a different layer in the networking stack. Figure 5.1 shows the elements stack.

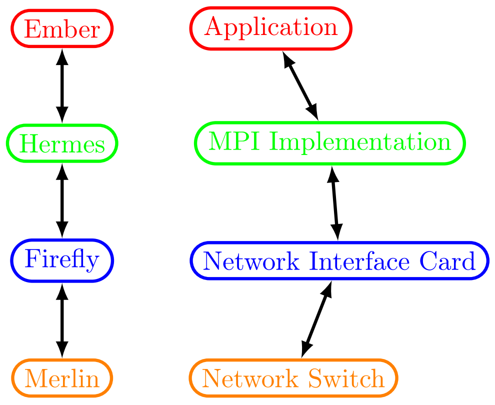


Figure 5.1: Network Simulation Stack

This stacked design allows for customisability and high fidelity without compromising the simulation of other elements in the stack; such as evaluating collective algorithms for a known network topology and interconnect.

Building on top of SST means that it inherits a large feature set; such as statistic collection being handled and aggregated at the core instead of the components. The components presented utilize statistic collection and aggregation so detailed analysis can be conducted at all levels in the networking stack.

5.1.1 Ember

Ember provides high-level logic for constructing communication patterns which we have named *motifs*. Motifs provide a synonyms mechanism with developing MPI or Symmetric Hierarchical Memory (SHMEM) applications so that they can be recreated for a scalable communications patterns.

A *motif* is constructed in three phases; decomposition, generate and complete. In the decomposition phase the pattern calculates in neighbours (if required) and creates the send/receive buffers such as for a halo exchange. During the generate phase the *motifs* feeds communication events in to a queue which are processed in chronological order, by the *Ember Engine*. The *Ember Engine* processes the event queue to ensure all events complete and ensures that the state machine flows correctly.

This approach of a usable API provides scalable communication traces that are not limited to traces alone, meaning that motifs scaling behaviour can be observed for large scale simulations. This is because traces store the number of bytes sent and received and then replayed with the same function call, to scale this approach requires working knowledge of the application so that the communications are scaled appropriately. Listing 5.1 shows how to make an MPI call for an application and Listing 5.2 shows an equivalent call in Ember.

```
MPI_Allreduce(&send, &recv, length, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD);
```

Listing 5.1: MPI Call

```
enQ_allreduce( evQ, m_sendBuf, m_recvBuf, m_count, DOUBLE, m_op,
    GroupWorld );
```

Listing 5.2: Ember Call

5.1.2 Hermes

Hermes is the SST element which provides state machines to ensure correct simulation of MPI and SHMEM semantics. The principle activity of the state machines is to convert communication activities from application patterns into MPI/SHMEM operations and have them progress correctly according to the programming model specification. Delays can be added in to the Hermes layer to represent software delay inside of the MPI and SHMEM layer of an application, see Listing 5.3.

```
    'sendStateDelay_ps' : 0,
    'recvStateDelay_ps' : 0,
    'waitallStateDelay_ps' : 0,
    'waitanyStateDelay_ps' : 0,
```

Listing 5.3: Hermes Latency Parameters

5.1.3 Firefly

Firefly provides the functionality of the NIC. This functionality allows for high fidelity simulation of the key components within a NIC; such as the bus used between the host and NIC. This component also handles the packetization and byte movement engine. Firefly is able to handle multiple MPI ranks connected to one firefly NIC, this is similar to current multicore systems.

The memory latencies are modelled under the assumption there is a base latency with an addition per byte. This calculation can either be linear or a multiplication of the size for a given size range. This is similar real systems, where latency can increase as size increases [23]. Listing 5.4 demonstrates how configured latencies may vary for different access sizes. In this example if 249 bytes was being read from the memory read to send over the network the latency to read the memory is 77.23ns.

```
    'txMemcpyMod': 'firefly.LatencyMod',
    "txMemcpyModParams.op" : "Mult",
    "txMemcpyModParams.base" : "10ns",
    "txMemcpyModParams.range.0" : "0-15:0ps",
    "txMemcpyModParams.range.1" : "16-63:250ps",
    "txMemcpyModParams.range.2" : "64-:270ps",
```

Listing 5.4: Transmit Copy Send Parameters for FireFly

5.1.4 Merlin

Merlin supplies topological and networking infrastructure. At the heart of Merlin lies a high radix router implementation that provides a extensible feature set to aid investigate how varying routing and Quality of Service (QoS) affect the overall performance of the system. Routers inside of HPC networks typically have a crossbar; this crossbar applies an algorithm that manages how packets flow across the crossbar. The high radix router models the input and output queues which means that the crossbar can be modelled cycle accurately without the time complexity of a cycle accurate simulator. The high radix router provides multiple cross-bar arbitration policies to investigate performance these include; round robin allocation, least recently used; age and random.

Network topology can affect network performance, as such when a new system is under design all topologies should be evaluated. Topologies that are currently supported include; Dragonfly, Fat tree, HyperX, Mesh, and a Torus.

The Fat Tree topology parameter is denoted as:
“<downlinks_L1>,<uplinks_L1>:<downlinks_L2>,<uplinks_L2>" for the simulator. This notation is used throughout this chapter when discussing Fat Tree topologies.

Dragonfly topologies take in a different set of parameters, these include the number of hosts per gorup, routers per group and the number groups. Listing 5.5 show the topology parameters for Isambard.

```
platdef.addParamSet("topology",{
    "hosts_per_router": 4,
    "routers_per_group": 96,
    "intergroup_links":10,
    "num_groups" : 1,
    "link_latency" : "120ns",
    "host_link_latency": "120ns",
})
```

Listing 5.5: Topology Parameters for Cray Aries

5.1.5 Simulator Performance

SST is a parallel-discrete event simulation toolkit and provides a variety of ways to parallelise and partition simulations running inside of the core. As with any parallel application how the job is run has significant impact on the runtime. Some considerations when running a simulation at scale are the simulated message sizes and network topology. These two things have the largest impact on the simulation time given they require knowledge of how the simulator will perform. Large messages (typically those over the rendezvous size) that have to traverse farthest across the simulated network and are poorly placed partitioned means that the runtime is excessively long.

Parallelisation

Threading and MPI are parallelisations techniques supported by the SST core. Threading is provided by PThreads[1] rather than OpenMP [41] threads.

There are some drawbacks and tradeoffs for a hybrid vs MPI approach. One main issue is the synchronisation that takes place between all the MPI ranks. If the simulator is ran as a fully MPI application the message sizes of the synchronisation decreases but there are more ranks for the collective operation to occur on which can increase the time of the operation.

This is of course dependant on what type of system you are running on, large simulations presented in this thesis have been run on Isambard, where as

[1]See: `https://man7.org/linux/man-pages/man7/pthreads.7.html`

smaller simulations have been run on a dual-socket BDW system and a Intel Core i7 Apple MacBook Pro. The run configuration for Isambard has been a full MPI approach, on the dual-socket system hybrid and solely threading have been used. Threading with the 20 threads on the BDW system offered some performance gain over MPI, presumably because the synchronisation did not have to take place and memory could be directly accessed.

Partitioning

The SST core provides two partitioners with the default installation, linear and round-robin. It also provides an interface to build with Zoltan [24], although this work only focuses on the use of the linear and round-robin partitioners as Zoltan's runtime far exceeded the time to run the simulation and therefore provided no benefit to the simulations.

Figure 5.2 shows linear and round-robin partitioning schemes for 8 MPI Ranks across two nodes. Linear fills the resources first before moving on to the next node. Round-robin iterates over all the nodes placing a component on the next available resource.

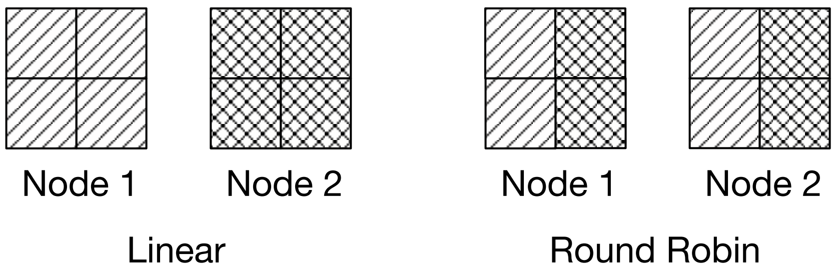


Figure 5.2: SST Partitioning Schemes

The round-robin partitioning approach has significant benefits for when simulating large Dragonfly networks because the network infrastructure is situated on the nodes where the NIC maybe located, this improves the data locality for the simulation improving the time to solution.

In this work Fat Trees have been simulated with linear partitioning and Dragonfly networks have been simulated using round-robin partitioning to improve the performance of the simulator.

5.2 Simulator Validation

We present a methodology to validate the decomposed simulator to provide assurances that the layers accurately reflect systems, and finally present the validated performance models. Validation of the simulator comes in two forms firstly the validation of the hardware model using low level benchmarks, such as PingPong, Streaming Bandwidth and collectives. Secondly the validation of the performance models built on top of the simulator. The performance models sit on a well benchmarked accurate platforms for the simulator and require changes at the *Ember* layer. Given the layered design this provides confidence that the platforms accurately reflect system architecture.

Existing validations for performance models primarily use micro-benchmarks to ensure that the point to point latency is modelled correctly [17, 38] and then make assumptions that all communication sit on top of this latency which is not always the case as with different implementations for performing an AllReduce. Newer approaches to trying to validate performance model include the use of machine learning [86]. This approach provides a fast performance model, compared to a simulation but can result in inaccurate models when extrapolating data given a training set [86]. This approach has been used across four machines with 3 different interconnects, it is shown that with a machine learning based performance model it can accurately predict runtimes for applications given the models high coefficient of determination (R^2). Both of these approaches are fine for high level models such as a macro simulations but lack the detail to truly test the components that comprise a low latency interconnection network. To test each of the sub-components inside of the networking stack further benchmarks will be required to ensure that parts of the networking subsystem are stressed which will push the performance model, this will provide assurance that each of the layers are being modelled closer to the real system.

This section demonstrates the highly accurate platforms for four systems, Astra, Isambard, Orac and Tinis. Performance models are then shown for three systems.

5.2.1 Validation Methodology

Mubarak et al. explain that network benchmarking should occurs over a variety of scenarios, such as separate switches, groups to ensure that the distance between the nodes is increased [99]. Benchmarks used for validation must also take in to account the message rates to ensure that they cater for a wide variety of messages sizes and that benchmarks should use a variety of node sizes, not just two in the case of a PingPong (or latency) benchmark [99].

Given the design of the simulator benchmarks have been chosen to stress the components, firstly the memory subsystem is benchmarked using LMBench [96]. LMbench measures the memory read latency by Back-to-back load; this is where each instruction is a cache-missing load so has to retrieve it from this next level of memory [96]. This benchmark was chosen because the benchmark was validated against a SGI Indy machine.

A latency benchmark is then run across 2 nodes across the same switch and separate switches. This allows the a comparison that link latencies reflect the system. This configuration is one core across both nodes. The benchmark must be run on different days to alleviate any background noise and must be repeated a significant number of times.

The bandwidth of the network must be measured, this is achieved using the Ohio State University (OSU) Bandwidth benchmark. As with the latency benchmark this must be run on separate days across the same switch and separate switches, running on separate switches provides insight in to how the network tapering affects the message rate.

Collective performance can then be ascertained with an AllReduce benchmark for the system which can be compared against the simulation. This should be performed with a packed node and one MPI rank per node; this will further confirm that the network model matches the system.

Once this data has been collected the model can be constructed and compared against this data, an accurate model will demonstrate similar trends through the results obtained with few deviations.

5.2.2 Modelling Hardware

To model a system first the system must be benchmarked to establish the performance of the key subsystems, these include the network performance and memory subsystem. The results of these benchmarks feed in to a platform.

Figure 5.3 shows the memory read latency for Isambard and Tinis. The interest is the latency after the L3 cache as this is where the data hits the main system memory. The L3 cache size of Intel HSW CPU is 15MB after which the latency is approximately 100ns rising with a slight include, which starts to flaten off towards as more memory is being read from this is because the bandwidth of the memory bus can be saturated. For Isambard the L3 cache size is 32MB, the latency to the main memory is around 64ns, this climbs quickly to reach the 100ns similar to Tinis. The bandwidth of the memory takes longer to get saturated as there is more bandwidth available on this newer CPU which is why it does not flatten as quickly as CPU in Tinis. Listing 5.6 documents the configured memory parameters for the Isambard platform.

Benchmarking the network performance involves benchmarking the entire network stack to establish any software overhead there might be limiting the true network performance. This additional software delay can be accounted for inside of Hermes in the simulation. The Intel MPI Benchmarks [69] and OSU Microbenchmark suite [104] have been used to benchmark the systems.

To construct a platform the values from a latency benchmark need to be used; although this only provides part of the picture to ensure the platform truly represents a system. We have also used a streaming bandwidth benchmark (part of the OSU microbenchmark suite) to ensure that the sustained bandwidth matches the system. A poorly designed platform focusing on just point-to-point

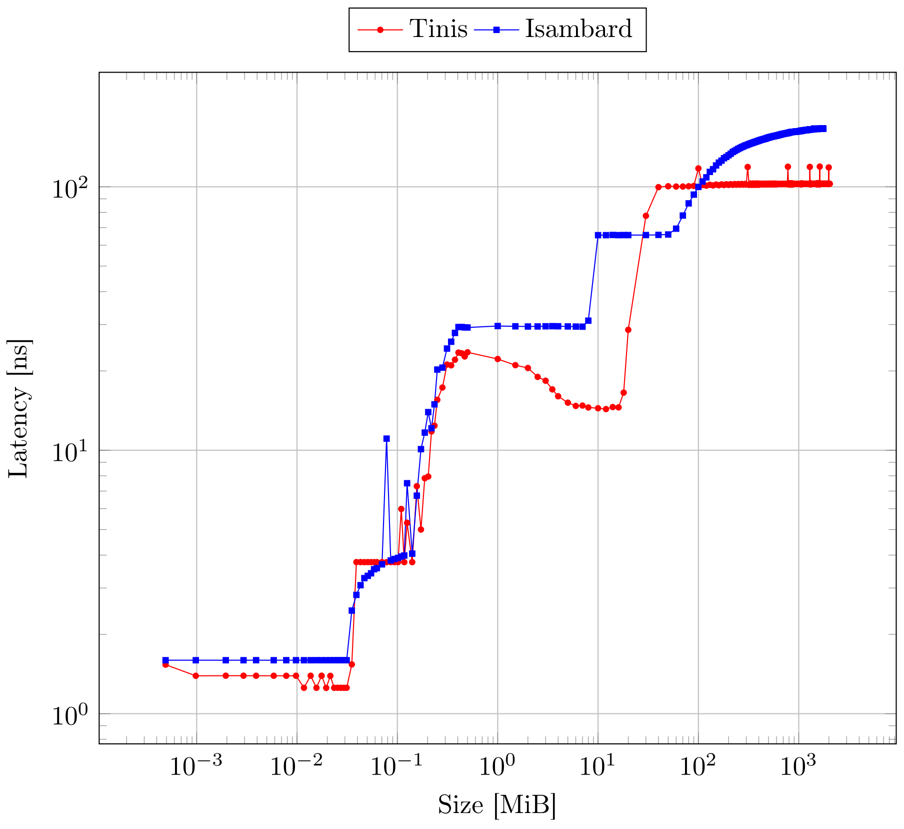


Figure 5.3: Measured Memory Latency for Tinis and Isambard

```
'txMemcpyMod': 'firefly.LatencyMod',
"txMemcpyModParams.op" : "Mult",
"txMemcpyModParams.base" : "65ns",
'txMemcpyModParams.range.0': '0-1024:500ps',
'txMemcpyModParams.range.1': '1024-8191:250ps',
```

Listing 5.6: Isambard Memory Parameters

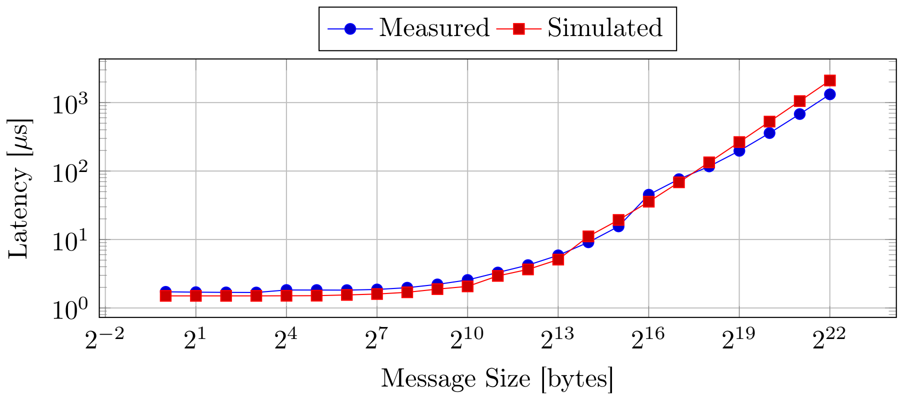
latency can result in higher sustained bandwidth, as the streaming benchmark exposes hardware latencies that can remain hidden. A implementation of the OSU Bandwidth benchmark has been developed for Ember.

Figure 5.4 shows how the simulated systems compares against the measured results for a latency benchmark. There is a large difference after the transition point between eager and rendezvous protocols is because the simulator is unable to utilise the buffers correctly, this gets resolved as the messages size increases, this increase appears across all systems modelled although most noticeable with the Astra platform. Errors are also introduced in the model as there are still some approximations in the model that are not catered for, one such assumption is that latency over the host to NIC bus is the same irrespective of the data transfer, this may not be the case yet it is difficult to capture the parameters for such a model without understanding the internals of a proprietary NIC. Another source of error in the model includes processing time within the switch, while the parameters for the simulator are broken down in to the input and output latencies in reality these are unknown in the real world and need to be approximated. In the case of Omni-Path the switch latency is around 100-110ns but does not state how this is split between the input and output [18]. Another factor that would affect the switch latency is the crossbar arbitration policy which again needs to be approximated as it is not publicly available, this has implications for two inputs sending to the same output port as to which would reach the output queue first.

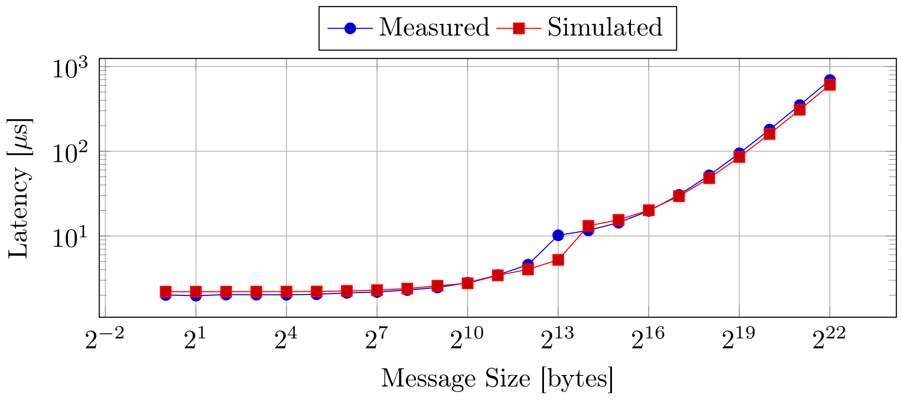
To mitigate against measurement error the micro-benchmarks have been repeated ten times across ten different points in time, this negates any background noise in the system and tends towards a value with little or no variance. With these results the arithmetic mean can be calculated, the use of the average time is suitable because of the law of large numbers shows that as the number of samples increase they tend to a specific value [49].

The application results presented were also ran a large number of times to ascertain the average time for the communications. Given the structured nature of the decomposition the computation time did not vary greatly. To mitigate against any background noise these jobs were typically during a maintenance window with a job size utilising at least 95% of the system. The applications were also ran on sequential nodes to minimise number of hops for the communications improving the runtime of the applications.

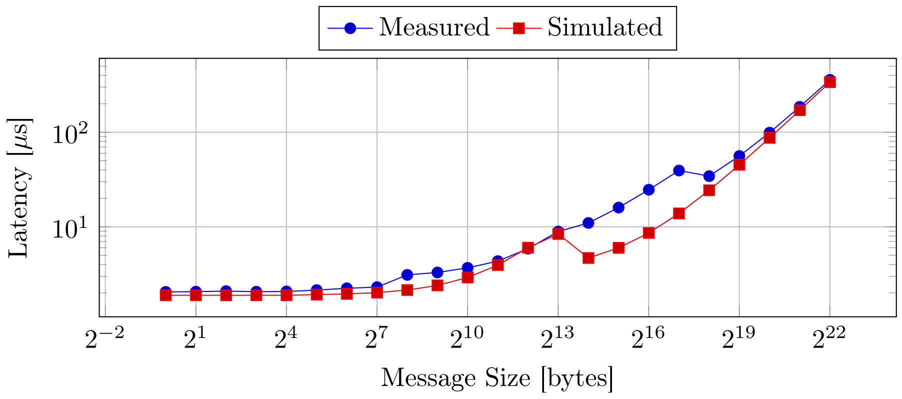
Figure 5.5 shows how the simulated systems streaming bandwidth benchmark compares to the measured results. The streaming bandwidth benchmarks



(a) Tinis



(b) Isambard



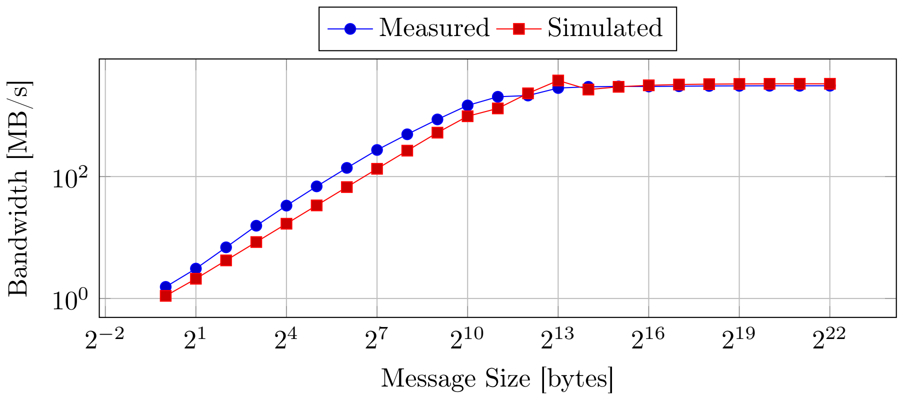
(c) Astra

Figure 5.4: Measured and Simulation PingPong Comparison for Three Systems

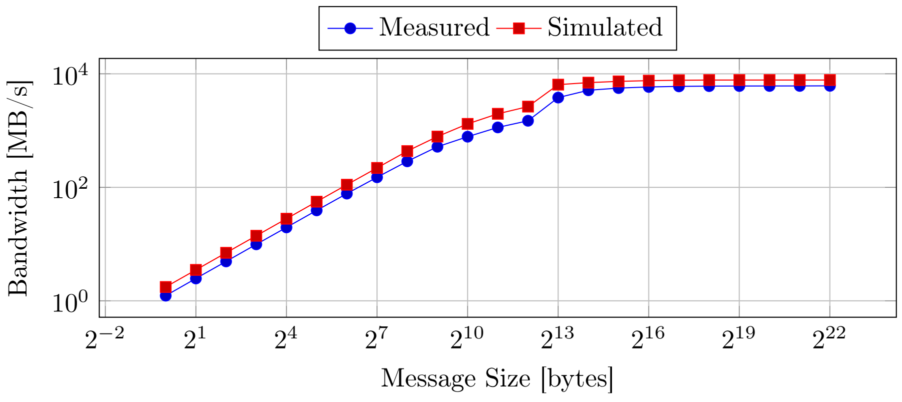
sends a set of messages (known as the window size) to an MPI rank using non-blocking MPI communications, once the receiver (usually rank 0) has received all of these messages it replies with a blocking send and receive. The time is captured for all of these communications is used to calculate the Bandwidth. This benchmark is capable of stressing the CPU/NIC bus as the messages sent with rapid succession [13]. The message rate is calculated from the measured bandwidth and can be seen in Equation 5.1, confirming that the message rates are accurate.

$$\text{Message Rate} = \frac{\text{Sustained Bandwidth}}{\text{Message Size}} \tag{5.1}$$

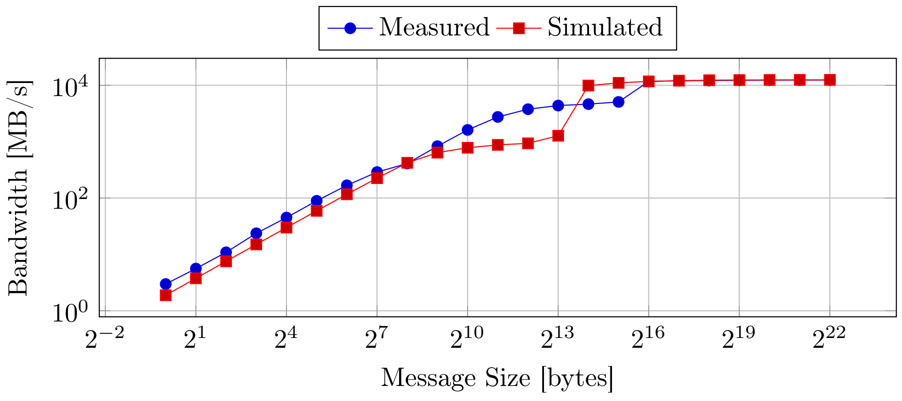
Mubarak et al. suggest that the collective performance is difficult to model as that they are implementation dependant and likely to show a shortcoming of the simulator [99]. Given this we have modeled our AllReduce applications as binary trees which do a reduce then a broadcast. All measured results presented were configured to ensure that AllReduce operations used a binary tree reduction and a broadcast, this was achieved by setting the following OpenMPI parameters: *coll_ tuned_ allreduce_ algorithm*, *coll_ tuned_ bcast_ algorithm* and *coll_ tuned_ reduce_ algorithm.* The comparison of an AllReduce between the benchmarks on Isambard and the simulation can be seen in Figure 5.6. One node with 1 PPN is not shown for clarity, but the measured time was 0.12μs and the simulated time was 0.1 μs resulting in an error of 17%. The time for an AllReduce operation with the default Cray MPI runtime was 54.84μs for 256 nodes, this tuned algorithm performs nearly 3.5$\times$ slower that the standard Cray collective operation but provides a useful comparisons against other network technologies that may not support optimisations made by the Cray MPI runtime. Astra's results showed less than 15% difference for the AllReduce timings when configured to use a binary tree, Figure 5.7 shows the comparison of the reduction for Tinis for a packed node.



(a) Tinis



(b) Isambard



(c) Astra

Figure 5.5: Measured and Simulation Bandwidth Comparison for Three Systems

Orac Modelling Case Study

As described in Section 3.1.2 is an Intel Omni-Path system based at Warwick. Initial work to model this system was abandoned to favour Tinis an InfiniBand

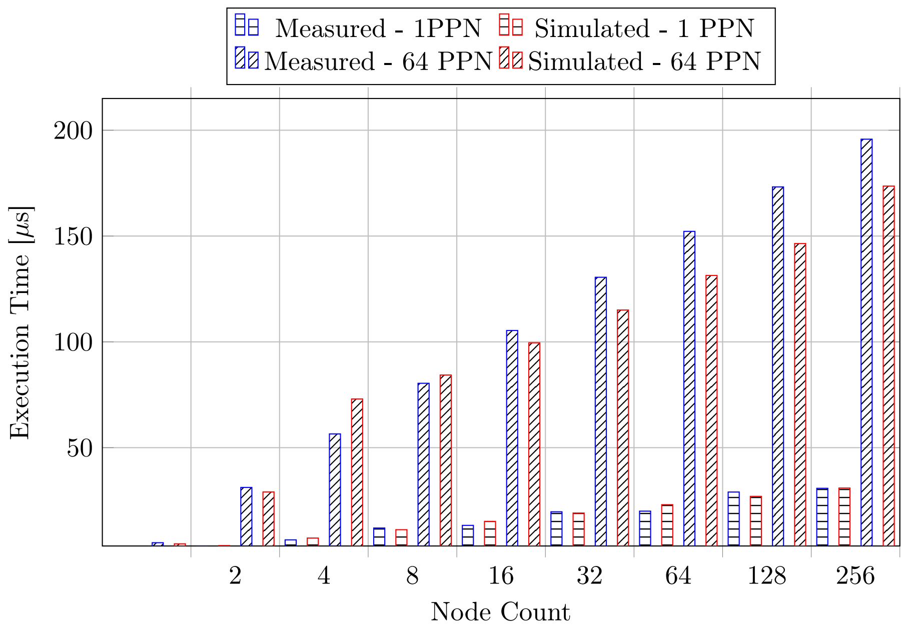


Figure 5.6: Measured and Simulated AllReduce Operation Comparison - Isambard

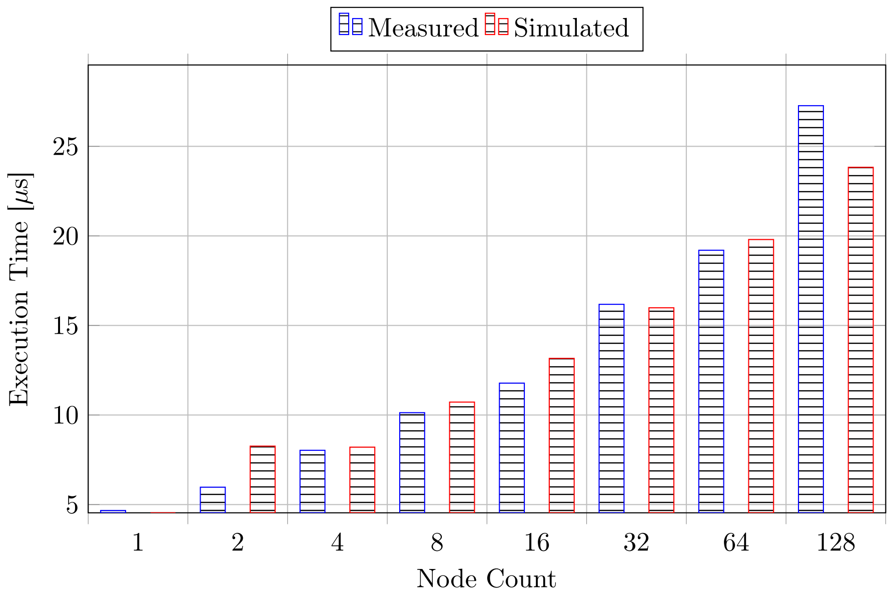


Figure 5.7: Measured and Simulated AllReduce Operation Comparison - Tinis 16 PPN

system given software configurations provided poor performance, latencies for small messages are approximately 5x slower with the configuration of OpenMPI

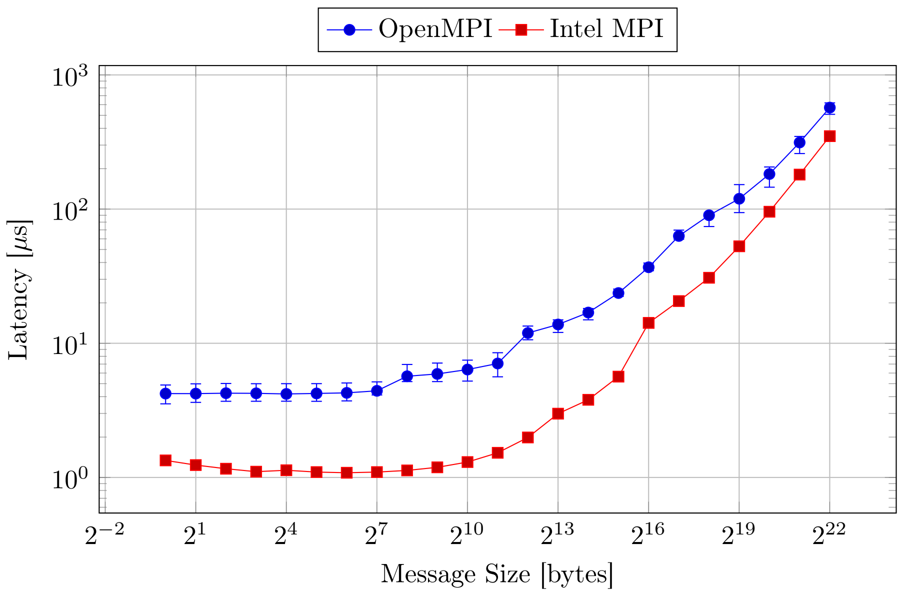


Figure 5.8: Orac MPI Latency - OpenMPI vs Intel MPI

compared to a newly built Intel MPI, see Figure 5.8.

The default OpenMPI (v4.0.5) is configured to use UCX which is poorly optimised for the Omni-Path network architecture, given the latencies seen across other systems are less than 2μS the 5μS for small messages shows that there is a performance implication - slower time to solution. With Intel MPI performing sufficiently well, we present the data from the hardware model using Intel MPI rather than OpenMPI.

The validation methodology (presented in Section 5.2.1) is applied to the development of the Orac model. The measured base latency was 96ns for the main memory latency for the BDW CPUs in Orac. Figure 5.9 shows how the simulated and measured latency compare, it is worth noting for the message size after the eager/rendezvous transition point is is 85% yet for the sustained bandwidth measurements this is only a -15.8% under prediction. The trend for the sustained bandwidth closely fits the measured results demonstrating that the message injection rate is correct, the differences in the latencies comes from the link latency (latency of a cable) not being matched to the system, see Figure 5.10.

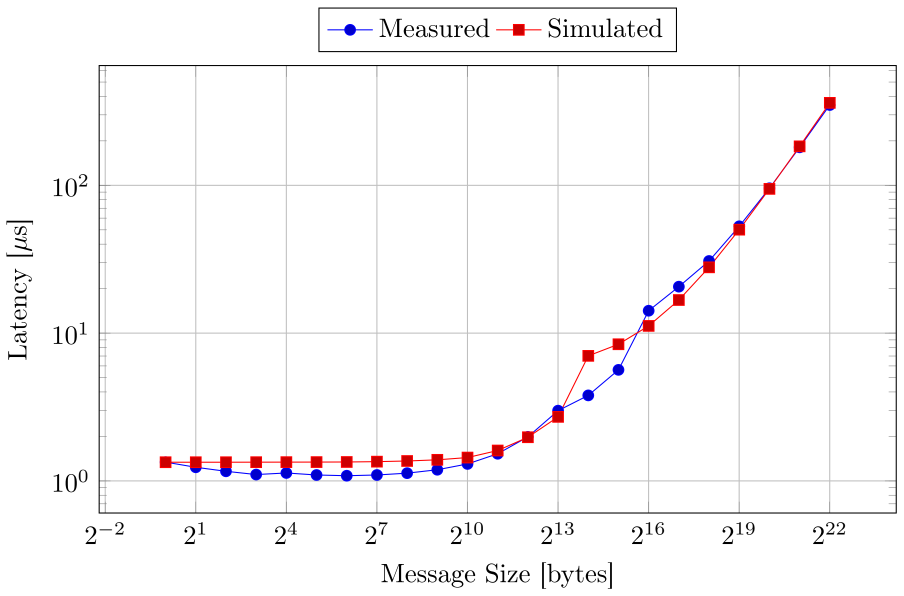


Figure 5.9: Orac Latency Simulation vs Measured (Intel MPI)

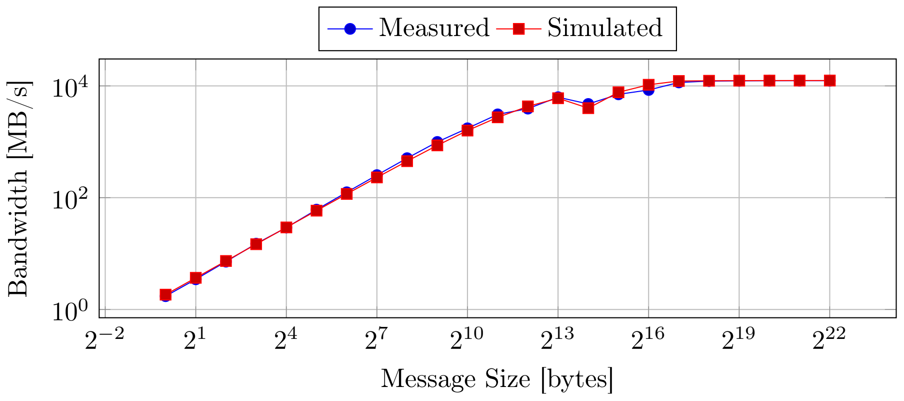


Figure 5.10: Orac Bandwidth Simulation vs Measured (Intel MPI)

Figure 5.11 shows how the model performs for an 8 byte AllReduce for both 1 PPN and 28 PPN. The average error for 1 PPN was -9.99% and for 28 PPN it was 10.78%. The 50% under prediction for 1 node with 28 PPN comes from the assumption in the model that there is more core-to-core bandwidth that the BDW CPU provides.

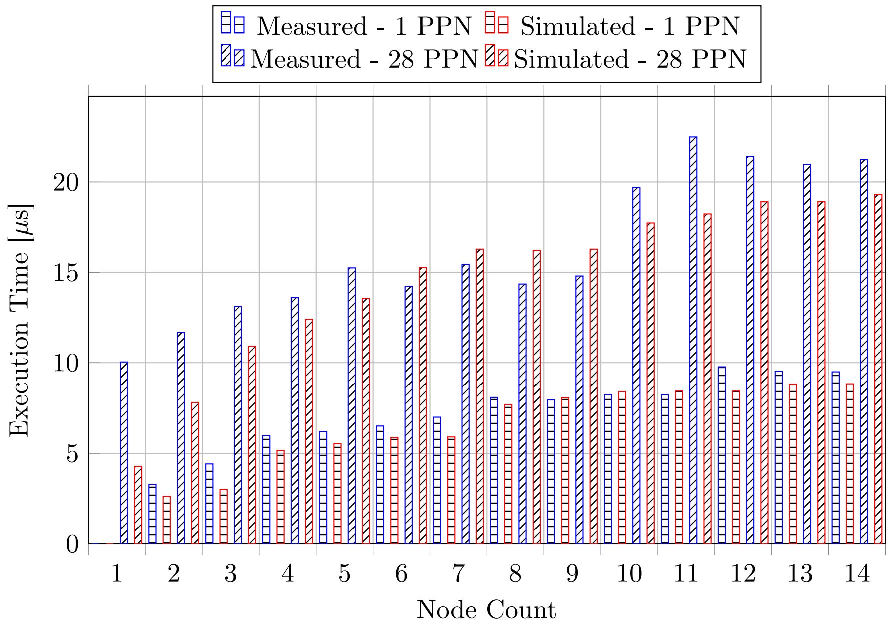


Figure 5.11: Measured and Simulated AllReduce Operation Comparison - Orac

5.2.3 Modelling Software

With the combination of hardware simulation and software simulation we can make predictions of the runtime of applications on system. We have chosen two proxy applications to model, firstly Sweep3D which is a three-dimensional discrete ordinates neutron transport benchmark [68, 80, 100]. Secondly we model TeaLeaf [95] a linear solver proxy application.

The two proxy applications were instrumented so that the key computation times could be established; this was achieved with Caliper [22]. Caliper is a power instrumentation framework that allows performance to be put in to applications during development and then activate the features at runtime. Caliper allows for application source code to be annotated and records snapshots during application execution.

Benchmark 3 and 5 were chosen for the selected problems for TeaLeaf and has been strong-scaled, TeaLeaf does not weak scale due to the decomposition scheme implemented [95]. The difference between the benchmarks is the grid size, for benchmark 3 a grid size of 500×500 is configured, for benchmark the grid size is 4000×4000. This problem is the crooked pipe problem in which a pipe has a lower density than its surroundings and therefore heat travels faster through this part of the problem domain. A Conjugate Gradient (CG) iteration in TeaLeaf consists of two reductions with computation and a 2D halo exchange with post computation, Listing 5.7 shows the input for the Tinis run for benchmark 3. When TeaLeaf is strong scaled like this at larger scales the communications can dominate the execution of an iteration; the computation roughly halves as MPI ranks double. This occurs because the problem size is fixed and distributed over more MPI ranks. We simulate one CG Iteration avoiding the convergence criteria of the application.

```
PlatformDefinition.loadPlatformFile("tinis_platform")
PlatformDefinition.setCurrentPlatform("tinis")

system = System()

ep = EmberMPIJob(0,128, 16, 1)
ep.addMotif("Init")
ep.addMotif("Allreduce iterations=1 compute=16169")
ep.addMotif("Allreduce iterations=1 compute=18818")
ep.addMotif("tealeafMotifs.2DHalo nx=500 ny=500 iterations=1
    post_compute_time=20522")
ep.addMotif("Fini")
system.allocateNodes(ep,"linear")

system.build()
```

Listing 5.7: TeaLeaf Performance Model Input for Benchmark 3 128 Nodes

Equation 5.2 defines our metric for the error between simulated and measured results. This version was chosen given that a minus error shows an under prediction rather than loosing the direction of the error.

$$\text{Error} = \frac{\text{Simulated} - \text{Measured}}{\text{Measured}} \times 100 \tag{5.2}$$

To model Sweep3D a motif was constructed to represent the communications from within the *inner_ auto* loop. This wavefront code provides multiple blocking angles per octant. This motif provides the ability to support weak scaling for an odd number of MPI ranks. This was achieved by causing the extra ranks provided inside of the simulator to exit from the motif early. Listing 5.8 shows how the motif exits early for ranks that sit outside of the run size when the performance model has been weak scaled across an odd number of cores, such as on Astra. Listing 5.9 shows how the angle blocking has been implemented for the (0,0) to (Px, Py) sweep.

```
if( rank() >= (px*py) ) {
    return true;
}
```

Listing 5.8: Exit Early Condition for Sweep3D Motif

```
// Sweep from (0, 0) outwards towards (Px, Py)
for(uint32_t kk = 0; kk < mm; kk += mmi) {
 for(uint32_t i = 0; i < nz; i+= kba) {
  if(x_down >= 0) {
   enQ_recv(evQ, x_down, m_xDownSendLen, 1000, GroupWorld);
  }

  if(y_down >= 0) {
   enQ_recv(evQ, y_down, m_yDownSendLen, 1000, GroupWorld);
  }

  enQ_compute( evQ, nsCompute );

  if(x_up >= 0) {
   enQ_send( evQ, x_up, m_xUpSendLen, 1000, GroupWorld);
  }

  if(y_up >= 0) {
   enQ_send( evQ, y_up, m_yUpSendLen, 1000, GroupWorld);
  }
 }
}
```

Listing 5.9: Angle Blocking for Sweep3D Motif

The 2D halo exchange motif was developed to model the decomposition and halo exchange exhibited in TeaLeaf. The decomposition scheme used is the same as the TeaLeaf application; the communications used are also the same. Listing 5.10 shows how the motif implements the Y direction for communication.

```
if (m_neighbour_top != EXTERNAL_FACE) {
    enQ_isend(evQ, NULL, y_send_length, DOUBLE, m_neighbour_top,
    1000, GroupWorld, &m_requests[msgRequest++]);
    enQ_irecv(evQ, NULL, y_send_length, DOUBLE, m_neighbour_top,
    1000, GroupWorld, &m_requests[msgRequest++]);
}

if (m_neighbour_bottom != EXTERNAL_FACE) {
    enQ_isend(evQ, NULL, y_send_length, DOUBLE,  m_neighbour_bottom
    , 1000, GroupWorld, &m_requests[msgRequest++]);
    enQ_irecv(evQ, NULL, y_send_length, DOUBLE,  m_neighbour_bottom
    , 1000, GroupWorld, &m_requests[msgRequest++]);
}

enQ_waitall(evQ, msgRequest, &m_requests[0], NULL);
```

Listing 5.10: TeaLeaf Halo Exchange Y Direction Motif

For benchmark 3 the largest error on Isambard was 12% and 10.37% on Tinis, see Table 5.1. Errors were slightly larger for benchmark 5 12.5% and 9.8% for Isambard and Tinis respectively.

Table 5.1: TeaLeaf Benchmark 3 - Model Validation

Nodes	Isambard			Tinis		
	Measured	Simulated	Error	Measured	Simulated	Error
1	231.23	229.15	-0.90	255.53	229.02	-10.37
2	239.76	263.51	9.91	180.67	162.96	-9.80
4	277.93	334.59	20.39	144.08	134.13	-6.91
8	315.29	370.55	17.53	126.46	122.76	-2.93
16	405.39	370.545	-8.60	117.71	124.35	5.64
32	455.37	449.35	-1.32	121.09	132.487	9.41
64	545.41	512.85	-5.97	124.38	130.241	4.71
128	590.61	553.39	-6.30	144.28	141.24	-2.11
256	656.05	634.83	-3.23	-	-	-

Sweep3D is a discrete ordinates transport code [80, 100]. We have simulated both strong scaled and weak scaling on Tinis and Isambard, and weak scaling results for Astra. The strong scaled runs saturated the nodes (64 processes per node (PPN) on Isambard and 16 PPN on Tinis). The weak scaled runs did not always saturate nodes although did try to keep the problem size as square as possible. For Astra the largest simulation consisted of 114,582

Table 5.2: TeaLeaf Benchmark 5

Nodes	Isambard			Tinis		
	Measured	Simulated	Error	Measured	Simulated	Error
1	12159.66	12155.90	-0.03	30085.96	29886.40	-0.66
2	5979.25	6000.53	0.36	14948.17	14831.800	-0.78
4	2974.79	3028.40	1.80	7425.83	7346.870	-1.06
8	1057.50	1088.24	2.91	3754.85	3685.990	-1.83
16	722.33	703.97	-2.54	1255.25	1194.900	-4.81
32	626.44	594.88	-5.04	447.08	411.942	-7.86
64	585.27	512.45	-12.44	285.25	268.982	-5.70
128	-	-	-	222.86	204.878	-8.07
256	712.59	629.66	-11.64	-	-	-

processes (2048 nodes). On Tinis, the problem size for strong-scaling was 150x150x150, the weak-scaling problem size was 50x50x800 per process. Isambard used 1000x1000x1000 for strong-scaling, weak scaling used 50x50x800. The Astra weak-scaled problem size was 50x50x800 per process. The blocking factor for all of the runs was 10.

The measured computation for weak scaling was 785μs, 24.9ms and 7.8ms per octant for Astra, Isambard and Tinis respectively. The standard deviation on the computation timings was 71μs for Isambard. The weak scaling predictions showed the largest error on Astra was -17.67% for 16 nodes, for Isambard this was -10.79% for 256 nodes and Tinis showed an error of -9.51% for 16 nodes. Table 5.3 shows how the simulator performed on Astra. Results for Isambard can be see in Table 5.4.

5.3 Towards Exascale Networks

Cost vs performance is one of the driving forces behind design decisions when designing a network. Dragonfly networks typically require significantly less routers and cables compared to a fat tree for large node counts. This is because the typical switch radix for a fat tree implementation is 36 or 48 this results in multiple layers to facilitate large node counts rather than the typical 2/3 level trees. Network vendors (such as NVIDIA/Mellanox and Cornelis) are starting

Table 5.3: Sweep3D Weak Scale - 50x50x800 Per Process - Astra

Nodes	Measured	Simulated	Error
1	13.89	12.62	-9.17
2	14.24	12.98	-8.83
4	14.91	13.25	-11.17
8	15.75	13.79	-12.47
16	17.48	14.39	-17.67
32	18.67	15.40	-17.53
64	19.97	16.72	-16.29
128	21.94	18.63	-15.07
256	24.55	21.31	-13.19
512	28.55	25.12	-12.00
1024	34.06	30.47	-10.55
2048	42.08	38.11	-9.43

Table 5.4: Sweep3D Weak Scale - 50x50x800 Per Process - Isambard

Nodes	Measured	Simulated	Error
1	18.10	17.48	-3.42
2.	18.38	18.09	-1.58
4	19.39	1.10	-1.50
8	20.30	20.41	0.55
16	-	-	-
32	24.10	24.93	3.46
64	27.35	28.75	5.11
128	31.91	34.08	6.81
256	46.65	41.62	-10.79

to provide support for multiple network topologies for their product lines. In the case of Mellanox 400Gbps network two network topologies are supported, a Fat Tree and Dragonfly+. When considering a next generation network the targeted network topologies could have significant implications on cost and performance. As the research and development costs grow as boundaries are pushed to deliver next generation networks this could have implications on the cost of different solutions as such different network toplogoies should be evaluated.

We consider three sized networks 256, 4096 and 131,072 nodes, and consider 5 switch radii looking at how cost and performance are affected. As the switch radix changes the network design should also change to reflect the larger switches. These three network sizes were chosen because they reflect representa-

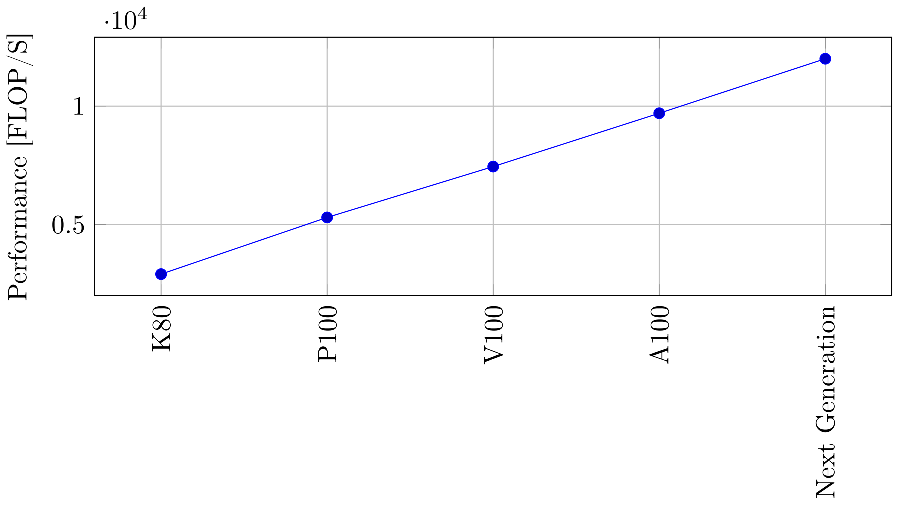


Figure 5.12: GPU Performance Trend

tive system sizes, if we consider that Tinis based at Warwick is 212 nodes with its replacement (Avon) being 180 compute nodes. ARCHER2 is 5,848 nodes and the 131,072 reflect a system that could be equivalent to a 1EFLOP system.

The node performance is based upon the current trend of NVIDIA GPU, see figure 5.12. The current trend shows that the the next generation after the A100 could provide 12TFLOPs per GPU (for double precision floating-point operations), if one GPU per node was chosen as a node architecture this would require 83,333 nodes to reach 1EFLOP. It is worth mentioning that a common design is to have multiple GPU, 6 GPUs per node is not uncommon across the top 10 of the TOP500, this would bring the node count down to 13,889 to build a system with theoretical peak performance of 1EFLOP.

5.3.1 Fat Tree

Fat Trees need further levels to facilitate more nodes, this is typically done with a high radix director switch which internally is a 2-level Fat Tree. In the following scenarios we consider all switches to be of the same radix rather than including a director switch. The designed networks are none-blocking.

Table 5.5 shows the network configurations for the networks for the desired node sizes.

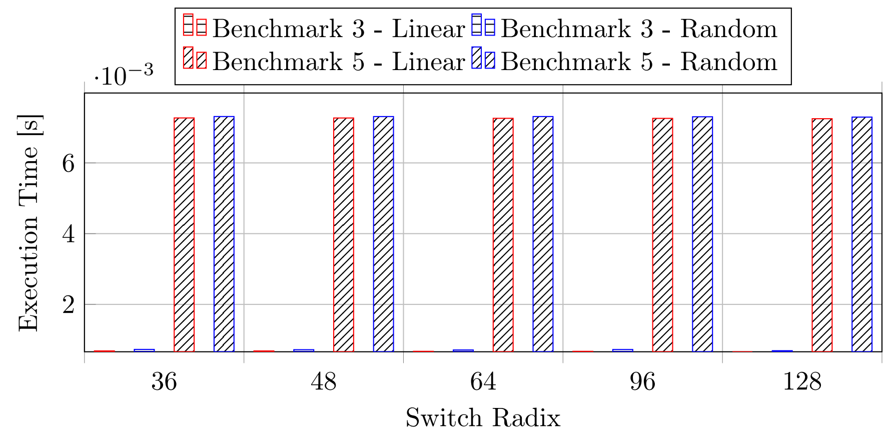
Table 5.5: Fat Tree Network Configurations

Switch Radix	Nodes		
	256	4096	131,072
36	18,18:15	18,18:18,18:13	18,18:18,18:18,18:18,18:2
48	24,24:11	24,24:24,24:8	24,24:24,24:24,24:10
64	32,32:8	32,32:32,32:4	32,32:32,32:32,32:4
96	48,48:6	48,48:48,48:2	48,48:48,48:48,48:2
128	64,64:4	64,64:64	64,64:64,64:32

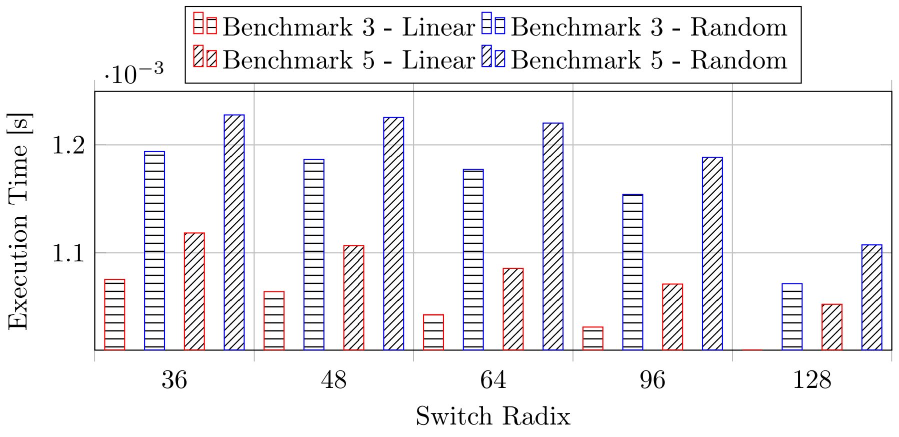
For the performance aspect both linear and random job placements are considered, we look at the time to solution as a measure of the performance. Random placements can provide an insight in to the worst case run time for a communication pattern. For smaller TeaLeaf problem sizes 3.5% improvements can be seen for linearly placed jobs, and can be as high as 5% for random placed jobs (see Figure 5.13). At 131,072 nodes the pattern is heavily communication bound, for random placement the small problem size for TeaLeaf can perform nearly 19% slower when placed randomly compared to linear job placement. For the small network random placement has a negligible effect on the time to solution for TeaLeaf.

In the case for Sweep3D improves can be seen as high as 7% for linearly placed jobs at 4096 nodes, Figure 5.14. Larger switches with random job placement suffer less performance degradation as there are more nodes connected to leaf switches and less layers in the Fat Tree for the traffic to traverse. The 131,072 sizes network can see a performance degradation of 25% for smaller hosts per router when random placement is considered over a linear placed job.

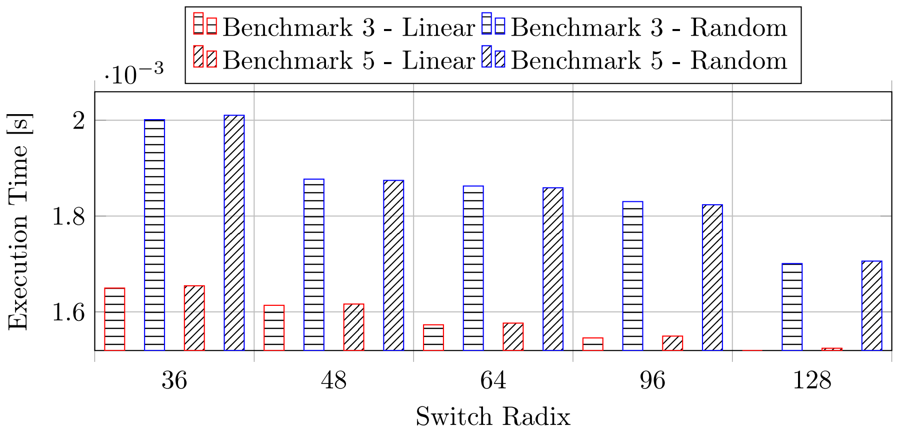
The following costings are as follows: switch is equivalent to 1 and a cable is equal to 0.02. We explore whether or not larger switches are more cost effective for a Fat Tree network. As the switch radix increases the number of switches requires decreases, in the case of the 256 node network the reduction is by $\frac{1}{3}$, in the case of the 131,072 node network this is around $\frac{1}{10}^{\text{th}}$ of the routers (see Table 5.6). Given that there is such a reduction in switches from 46,656 to



(a) 256 Nodes

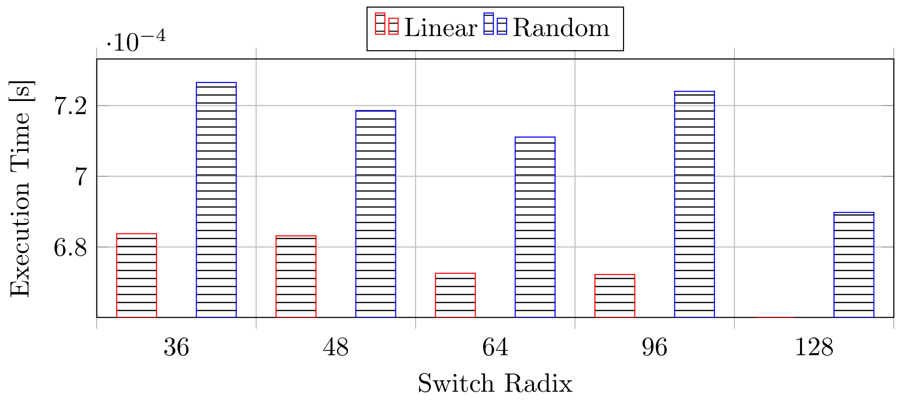


(b) 4096 Nodes

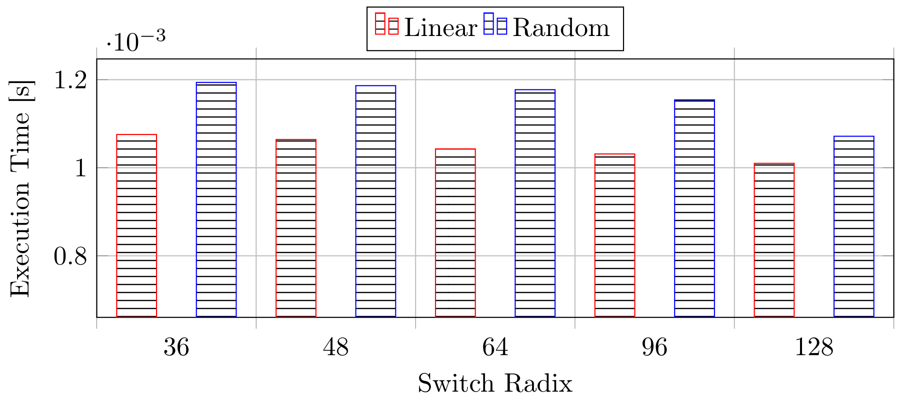


(c) 131072 Nodes

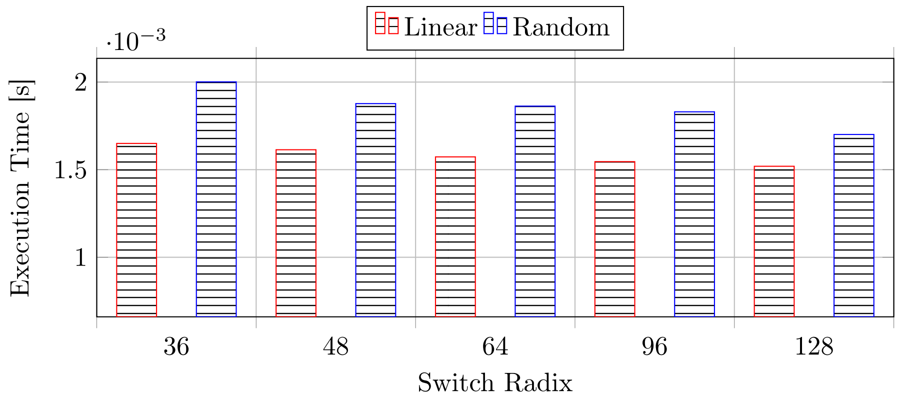
Figure 5.13: TeaLeaf Runtime Performance for varying Switch Radix - Fat Tree



(a) 256 Nodes



(b) 4096 Nodes



(c) 131072 Nodes

Figure 5.14: Sweep3D Runtime Performance for varying Switch Radix - Fat Tree

4096 routers required to implement a large network (131,072) when increasing the switch radix the cost reduction (see Figure 5.15) also becomes noticeable driving the motivation for manufacturers to design larger switches.

Table 5.6: Fat Tree Switch Count Per Switch Radix

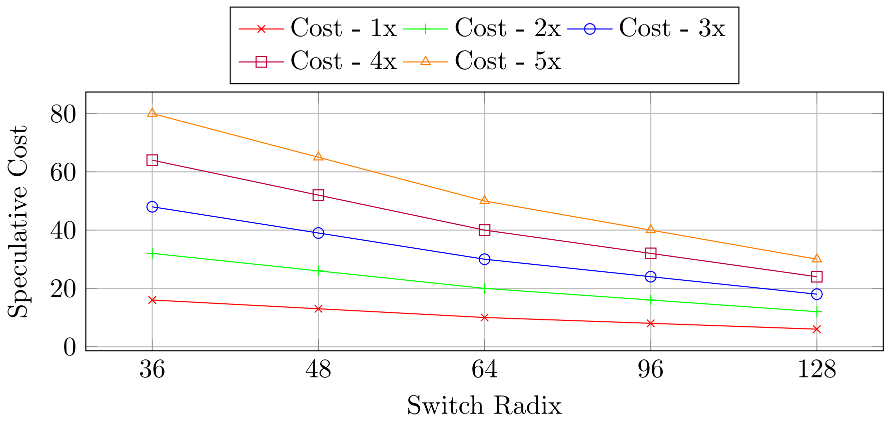
Switch Radix	Nodes		
	256	4096	131072
36	16	468	46656
48	13	384	17280
64	10	256	12288
96	8	192	13824
128	6	66	4096

5.3.2 Dragonfly

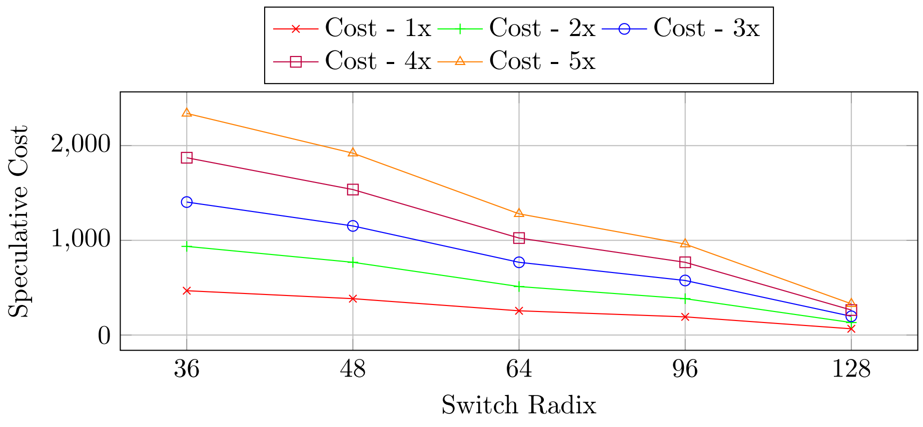
Dragonfly networks consist of groups that are interconnected, the one benefit of the Dragonfly is the reduced cost for large networks when compared to Fat Tree networks. Similar to Fat Trees we look at designing dragonfly networks for three networks; although now we consider the switch radix to be the number of hosts per router. This because while the Cray Aries is a 64 port router only 4 of those ports go to hosts. Considering just the switch radix instead of number of hosts limits the maximum network size, which may be insufficient to achieve the 131,072 network size. As an example if a Dragonfly is configured with a 48 port switch the maximum number of hosts is 4096 if the ports are divided by equally, with 16 hosts per router, 16 routers per group, and 16 groups and 1 inter-group connection.

Table 5.7 shows the designs of the Dragonfly networks used for analysis. The number of inter-group links was configured as 10; this is sufficient as the test patterns for Sweep3D and TeaLeaf are latency bound rather than bandwidth bound.

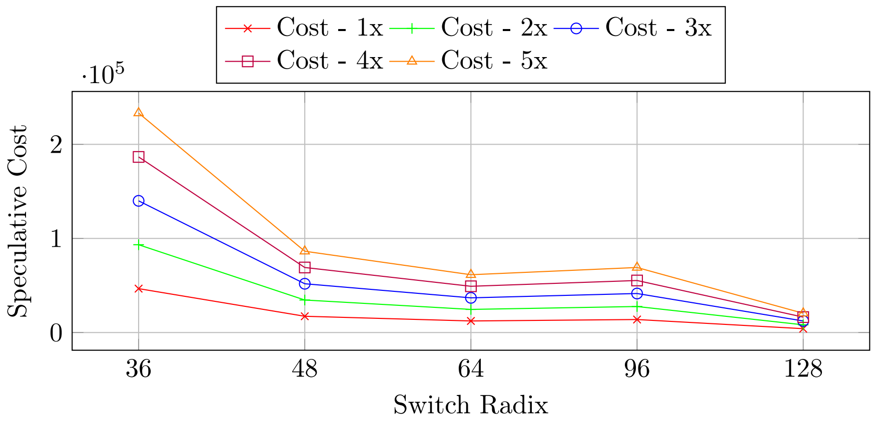
For a small Dragonfly (256 nodes) the job placement of a TeaLeaf application has no effect on the runtime, compared to larger Dragonfly (131,072 nodes) networks where the placement makes TeaLeaf perform around 9% worse compared



(a) 256 Nodes



(b) 4096 Nodes



(c) 131072 Nodes

Figure 5.15: Costs Trends for Fat Tree Network

Table 5.7: Dragonfly Network Configurations

Hosts Per Router	Nodes					
	256		4096		131072	
	Routers Per Group	Number of Groups	Routers Per Group	Number of Groups	Routers Per Group	Number of Groups
36	4	2	64	2	64	64
48	3	2	43	2	43	64
64	2	2	32	2	32	64
96	2	2	22	2	22	64
128	1	2	16	2	16	64

to a linear placed job (see Figure 5.16). Large jobs on the Dragonfly randomly placed the problem size have a negligible effect on the runtime.

Performance gains are seen for all network sizes with an increased switch radix, Figure 5.17. For large Dragonfly networks performance degradation can be as high as 13% for randomly placed jobs

Figure 5.18 shows the costs for the Dragonfly network configurations. For 131,072 nodes increasing the switch radix from 36 nodes to 128 could see the cost decrease by as much as 4×.

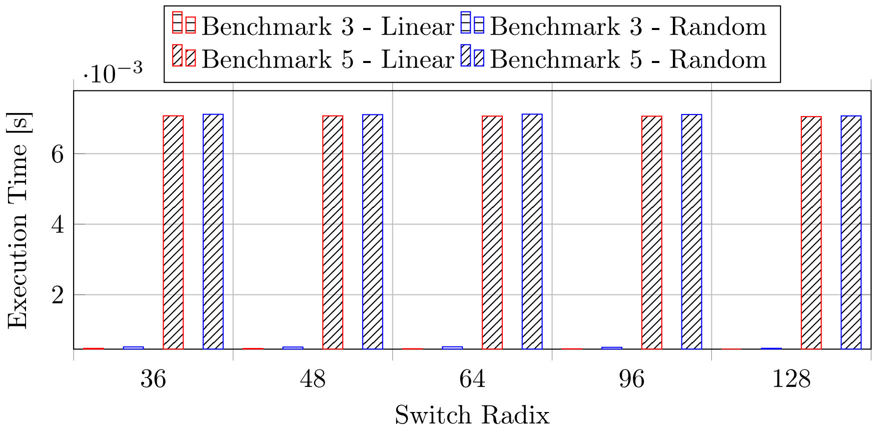
5.4 Summary

This chapter has demonstrated three validated hardware models for network simulations built with SST. A case study looking at how poorly configured software can lead to issues something that modelling can assist with diagnosing. Three *motifs* have been implemented for *ember* to produce the models presented. The OSU Bandwidth benchmark has been implemented to ensure that the models accurately reflect the system. We present accurate models of the hardware by demonstrating that micro-benchmarks are less than 10% different. This chapter has shown the development of two motifs for modelling Sweep3D and TeaLeaf using Ember. We show that two applications can be modelled accurately with these three platforms with all three systems showing less than 20% error.

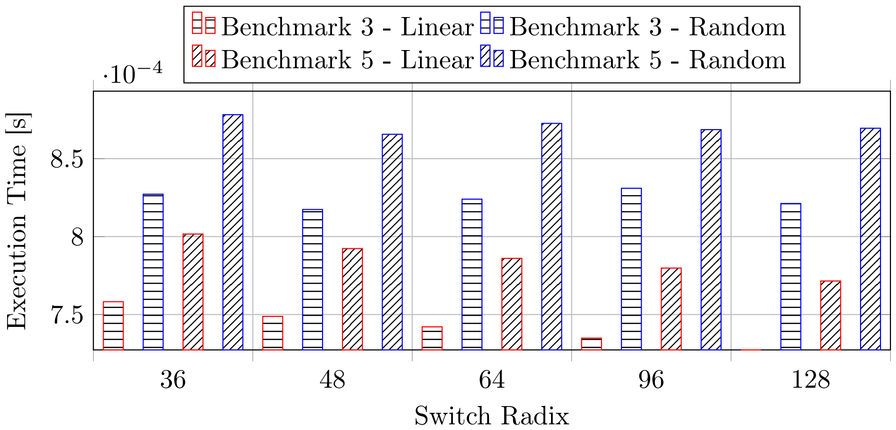
The performance of the simulator is dependant on the simulation being run, care needs to be taken to ensure resources are utilised correctly with respect to how big to scale the simulation and run configuration such that it effectively utilises the hardware.

These models have been used to speculate about what future networks could look like with growing switch radii. Cost and performance have been considered, showing that there are performance gains and cost savings to be made from larger network switches. The performance improves for Fat Trees could be as high as 7% for linearly placed jobs if vendors provide larger switch radii. The time to solution is improved with the design for a Dragonfly presented over the competing Fat Tree for 131,072 nodes. A Dragonfly solution can be 10× cheaper than an equivalent sized Fat Tree network for 131,072 networks, irrespective of the switch radix used. The Dragonfly topology is good solution to minimise cost and improve performance when a full system job is considered to be the primary target.

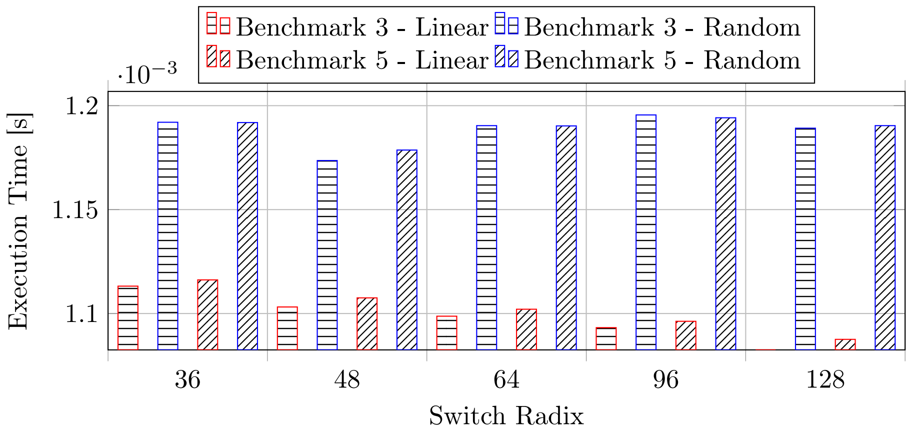
Increasing the switch radix shows performance improvements regardless of the network topology being chosen, this is because typically applications do significant near neighbour communications and reducing the hop counts can improve the communication time improving application runtimes. One implication of increasing the switch radix is that the physical circuitry for encoding the date on to the wire (called a SerDes) cannot be made smaller inside of the switch Application Specific Integrated Circuit (ASIC) resulting in a physical limit on the number of ports on a switch. One alternative to this is to bifurcate the port and half the available bandwidth, this could be beneficial for applications which do not need the available injection bandwidth, such as TeaLeaf and Sweep3D. This would allow for a higher switch radii and negligible effects on the time to solution at scale.



(a) 256 Nodes

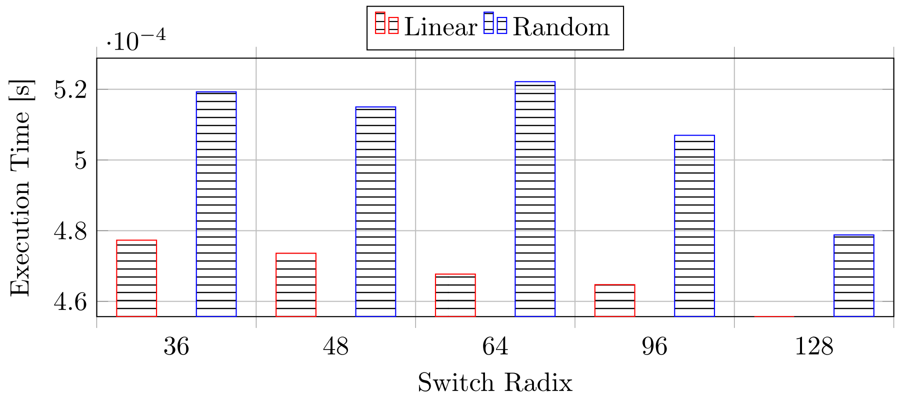


(b) 4096 Nodes

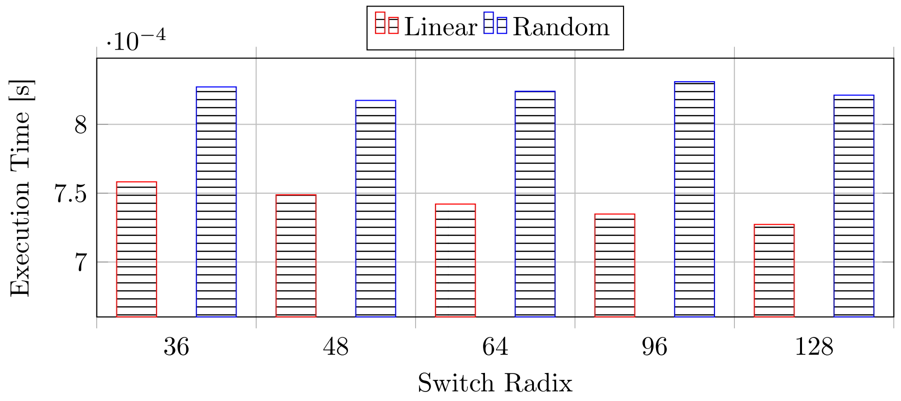


(c) 131072 Nodes

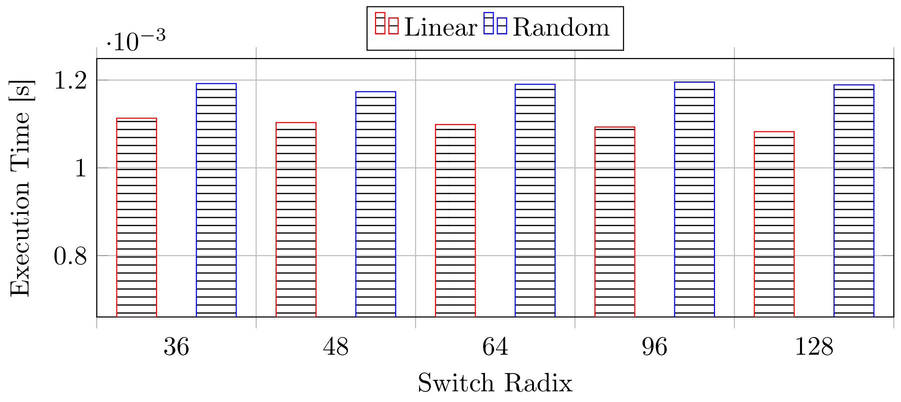
Figure 5.16: TeaLeaf Runtime Performance for varying Switch Radix - Dragonfly



(a) 256 Nodes

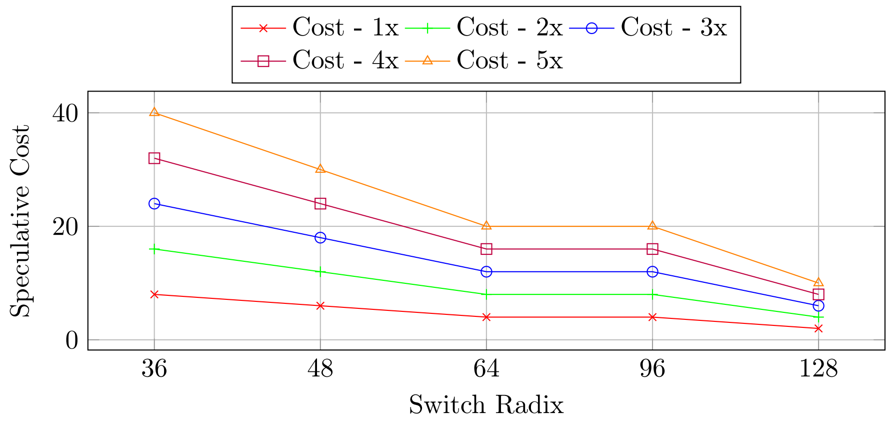


(b) 4096 Nodes

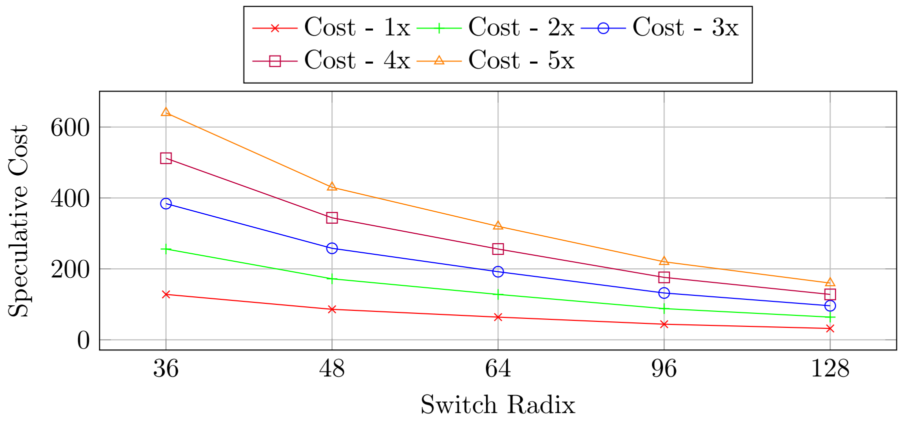


(c) 131072 Nodes

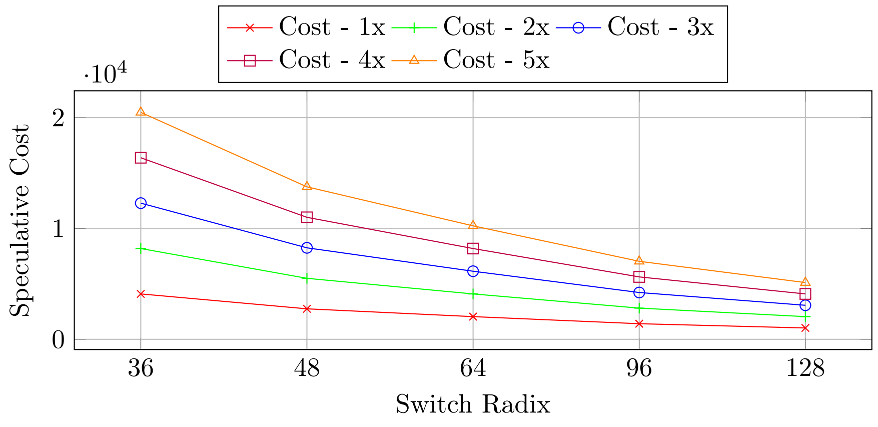
Figure 5.17: Sweep3D Runtime Performance for varying Switch Radix - Dragonfly



(a) 256 Nodes



(b) 4096 Nodes



(c) 131072 Nodes

Figure 5.18: Costs Trends for Dragonfly Network

CHAPTER 6
Contention Aware Performance Modelling

Until now performance modelling has focused on simplistic models predicting the peak performance of applications and allowing us to extrapolate data to understand what limitations arise in larger systems. While this peak performance is useful in diagnosing hardware/software issues that occur during commissioning (such as Section 5.2.2) it is still not reflective of how applications will perform in a multi-user shared environment. This could be beneficial with designing systems for a Request for Proposal (RFP) as a multi-user shared environment can be demonstrated prior to implementation of a system. It would allow for rapid prototyping of the network design space such that cost vs performance can be evaluated for capacity systems (for example those procured by the DOE under the Commodity Technology Systems (CTS-1) platform[1]). Commodity Technology Systems are designed to run many small jobs, compared with a capability system which may run a few large jobs (typical full machine/partition runs). Another option is to consider that time is finite and to achieve the most throughput from a system should multiple smaller jobs be run (like a capacity system) or allow applications to run at full system scale (capability system), this is dependant on the use-case for a HPC system and the applications that are being run. The modelling approach used in Section 6.1 can be used to evaluate whether a system could provide a better throughput for a capacity or capability jobs.

Modelling network contention is difficult given the temporal nature of the issue. Modern approaches have looked at using analytical models for modelling contention [91], or have focused purely on one application [61] to look at what is going on inside of the network. The work presented in this chapter looks at

[1]See: `https://asc.llnl.gov/computers/commodity`

simulating multiple applications to understand the interactions between applications for both time to solution and network utilisation, both of which are of interest when designing a network as it can provide insight in to the expected level of performance that may be seen in a multi-user environment.

The simulator (presented in Chapter 5) has been designed to simulate multiple jobs across a network as well as a singular job. This is achieved by assigning the endpoints different jobs. If endpoints inside of the simulator are not provided a job, they do not drive any network traffic.

In this chapter we use the simulator to investigate three scenarios looking at effects inside of the application level and time to solution as well as a view inside of the network. Firstly we evaluate the effects of tapering a Fat Tree in an effort to increase computation while keeping network costs the same. Secondly network utilisation is studied through simulation from contending applications. Finally we investigate using StressBench as a tool to validate the simulators multi-user simulations.

6.1 Network Tapering

One key consideration when designing a network for a system is the global bandwidth. A non-blocking fat tree has full global bandwidth, although this is costly to implement as it requires more switches and cabling. In this section we evaluate the effects of network tapering through simulation, by understanding whether there is a net gain from increasing node count and network taper.

This work focuses on tapered Fat Trees, although the Dragonfly implementation from Cray can be tapered by removing the global links between groups. The network switch in question is a 36 port leaf network switches, similar to switch design inside of Tinis. The hardware platforms are based upon the Tinis model with a modified Fat Tree shape. Non-blocking this network supports 72 nodes, this can increase to 128 nodes with an 8:1 taper. Table 6.1 shows the shape of the tapered networks making use of a 36 port switch. Figure 6.1 shows a leaf switch with a non-blocking configuration.

Table 6.1: Network Tapers

Blocking Factor	Network Shape
Non Blocking (1:1)	18,18:4
2:1	24,12:4
3:1	27,9:4
5:1	30,6:4
8:1	32,4:4

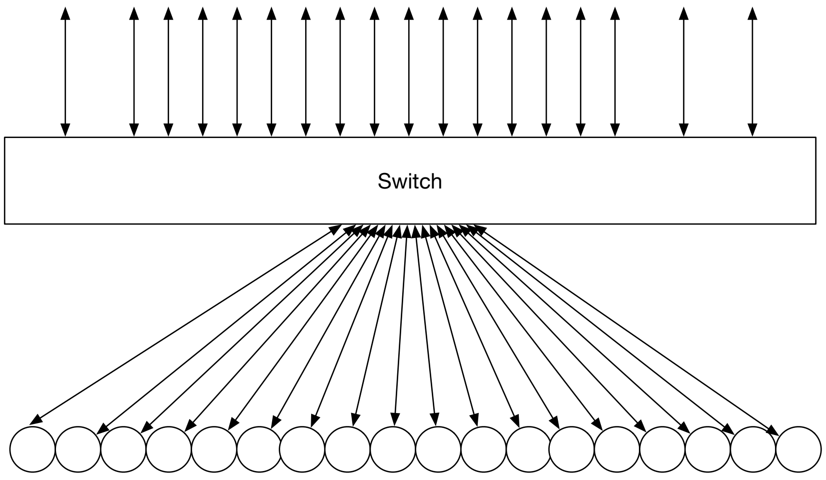


Figure 6.1: Non-Blocking Leaf Switch - Nodes represented by circles

The effects of tapering on three communication patterns, Sweep3D, ten TeaLeaf CG iterations and an AllToAll. For Sweep3D a cube of 100 and 1000 was chosen to run, benchmark 3 and 5 for CG was used for the problem sizes for the TeaLeaf CG iterations, and a small (1024B) and large (4MB) AllToAll was used.

Linear and Random job placement was used to understand the effects on the runtime of the communication pattern. Random job placement provides insight in to the worst case runtime as jobs may not be situated on consecutive nodes, resulting in an sub-optimal job placement.

6.1.1 Entire System

In this section we consider how the communication patterns are affected by the network tapering for Tinis if the network taper was changed to increase the computation.

Table 6.2 and 6.3 shows how the patterns perform for linear and random job placements at 72 nodes. For larger problem sizes random job placement increases the time to solution of the communication patterns, this is because the jobs may be further away and have to travel further across the network. In the case of linear job placement improvements in the runtime for the patterns is seen because the traffic is located on the same switch so less traffic has to flow through the root switches. For the AllToAll the reduction in the global bandwidth has a significant effect on the runtime increasing by as much as 33.2% for a linearly placed AllToAll pattern. The effects of tapering lessen for nearest neighbour communications like those seen in TeaLeaf and Sweep3D.

Table 6.2: Network Tapering for Linear Job Placement

Taper	AllToAll		Sweep3D		TeaLeaf	
	1024	4MB	100	1000	BM3	BM5
Non-Blocking (1:1)	2.7944	3.9523	0.0393	6.5438	0.5416	2.7514
2:1	2.7944	4.1500	0.0393	6.5438	0.5416	2.7514
3:1	2.7944	4.3024	0.0393	6.5440	0.5416	2.7514
5:1	2.7944	4.6764	0.0393	6.5439	0.5416	2.7514
8:1	2.7944	5.2651	0.0393	6.5439	0.5416	2.7514

Table 6.3: Network Tapering for Random Job Placement

Taper	AllToAll		Sweep3D		TeaLeaf	
	1024	4MB	100	1000	BM3	BM5
Non-Blocking (1:1)	2.7944	4.0603	0.0396	6.5443	0.5417	2.7514
2:1	2.7944	4.2330	0.0395	6.5443	0.5417	2.7515
3:1	2.7944	4.4142	0.0396	6.5444	0.5417	2.7514
5:1	2.7944	4.7417	0.0396	6.5449	0.5417	2.7515
8:1	2.7944	5.1786	0.0396	6.5448	0.5417	2.7515

6.1.2 Contended Applications

In this section application contention is studied to understand if network taper affects the time to solution for applications.

The system is configured to use nine 8-node jobs rather than the 72 nodes in Section 6.1.1, this is similar to how a real system maybe used in a multi-user shared environment. An RFP *may* state that jobs are less than a specific size (e.g. eight nodes). One key design criteria may to be have a multiple of job size for the number of nodes off a leaf switch to minimise the distance that traffic has to travel. It is also worth nothing that the throughput of the system is improved for a tapered network, so for a fixed period of time the system can run more jobs. In this simple configuration 8 more jobs could be processed in the time frame when heavily tapered (e.g. 8:1) compared to the non-blocking design, this is beneficial if the target applications for a system do not require global bandwidth.

The applications are configured to use the same parameters (problem sizes are the same) this is similar to a parameter sweep in which a property of the physics (such as the initial density or temperature) could change but the problem size remains the same.

Table 6.5 and 6.6 show how the applications perform under contention in a linear and random job. These results are the total time to run all jobs. The isolated times to run 1 job on a none blocking fat tree can be seen in Table 6.4, this could be considered the best time to solution.

For applications like Sweep3D and TeaLeaf there is a negligible difference in the runtime when considering a tapered network design rather than none blocking design. While if the applications in use feature large AllToAlls then a tapered network design will hinder the performance by as much as 11.7% for random job placement. When the linear job placement is considered the performance suffers degradation for a non-blocking Fat Tree design by as much as 7.4%.

Table 6.4: Isolated Runtimes for Tapered Fat Tree

Pattern	Problem Size	Runtime [s]
AllToAll	1024	0.2848
	4MB	0.7758
Sweep3D	100	0.0227
	1000	6.8216
TeaLeaf	Benchmark 3	0.2508
	Benchmark 5	20.4734

Table 6.5: Network Tapering on multiple applications - Linear Job Placement

Taper	AllToAll		Sweep3D		TeaLeaf	
	1024	4MB	100	1000	BM 3	BM 5
Non-Blocking (1:1)	0.285	0.833	0.023	6.851	0.251	20.474
2:1	0.285	0.776	0.023	6.822	0.251	20.473
3:1	0.285	0.833	0.023	6.851	0.251	20.474
5:1	0.285	0.841	0.023	6.851	0.251	20.473
8:1	0.285	0.776	0.023	6.822	0.251	20.473

6.2 Network Utilization

HPC systems are rarely single job machines and often run a variety of applications concurrently. These concurrent jobs can interact with each other generating contention. As the contention increases inside of the switch has to buffer some of the traffic.

Given that the simulator models the queues on the input and output ports this contention can be modelled easily by simulating multiple communication patterns simultaneously.

A tapered Fat Tree is evaluated to understand how congestion affects application communication traffic through simulation. The tapered Fat Tree is a subtree from the Astra Topology, this is because the statistics collection results in vast amounts of data generated from the simulation.

PingPong has been contended against an AllToAll congestion pattern with a varying congestion message size, given the 2:1 taper inside of the Fat Tree in Astra network performance can drop by as much 2GB/s for very large congestion message sizes (64K Bytes), see Figure 6.2. The drop in the available bandwidth

Table 6.6: Network Tapering on multiple applications - Random Job Placement

Taper	AllToAll		Sweep3D		TeaLeaf	
	1024	4MB	100	1000	BM 3	BM 5
Non-Blocking (1:1)	0.285	0.841	0.023	6.888	0.251	20.474
2:1	0.285	0.855	0.003	6.891	0.251	20.474
3:1	0.285	0.863	0.023	6.886	0.251	20.474
5:1	0.285	0.885	0.023	6.897	0.251	20.474
8:1	0.285	0.939	0.023	6.912	0.251	20.474

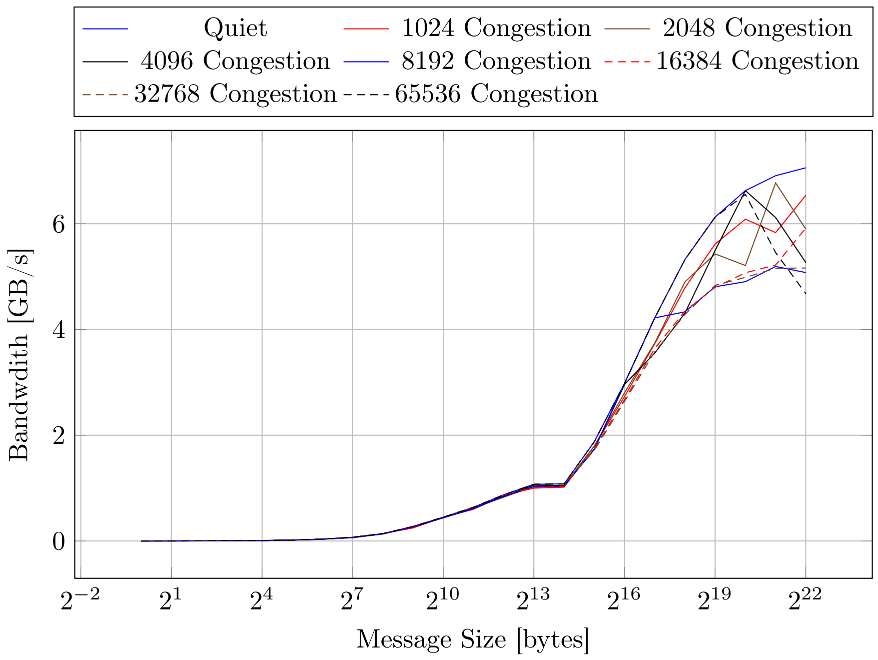


Figure 6.2: Simulated Bandwidth with varying congestion size

is because the AllToAll consumes all the bandwidth on the uplinks inside of the fat tree, this means that the smaller PingPong messages have to wait for the AllToAll iteration to complete before these messages can complete. One option to try and improve the performance could be preemption, which interleaves the smaller messages with the larger messages and can provide a fairer share of the communication channel, like the saturated uplinks inside of the Fat Tree. The presented bandwidth was calculated using Equation 6.1. This reduction in available bandwidth can have a profound effect on patterns that utilise large message sizes, such a I/O traffic.

$$BW = \frac{\text{Bytes}}{\text{Latency}} \tag{6.1}$$

Timing information can be captured by the simulator to time the three possible states for an output port on the high-radix switch. The three states are Stalled, Active, Idle (SAI). Active represents data flowing, idle means that there is no communication taking place and could be performing computation. The stalled state is where there is data to send but can not send the data. Stalling can occur either because of insufficient credit (where there is data to send but not enough credit to send the data) or the arbitration policy of the crossbar prohibiting the transmission of data.

The times captured can then normalised to represent a portion of the run-time, Equations 6.2, 6.3, 6.4. The sum of the normalised times will equate to 1.

$$\text{Stalled} = 1 - \frac{\text{Active} + \text{Idle}}{\text{Runtime}} \tag{6.2}$$

$$\text{Idle} = 1 - \frac{\text{Active} + \text{Stalled}}{\text{Runtime}} \tag{6.3}$$

$$\text{Active} = 1 - \frac{\text{Idle} + \text{Stalled}}{\text{Runtime}} \tag{6.4}$$

To represent these three states a ternary plot is used, Figure 6.3 shows an example ternary plot. In the figure the blue circle represents active all of the time, the red square represents fully stalled and the brown circle is fully idle. The black star represents the values $\frac{1}{3}$ Active, $\frac{1}{3}$ Idle, $\frac{1}{3}$ Stalled. The blue diamond is 0.8 Active, 0.05 Stalled and 0.15 Idle.

An AllReduce, 2D Halo Exchange, Sweep3D and AllToAll congestion pattern were simulating varying the size of the AllToAll congestion pattern. The jobs were placed randomly. Listing 6.1 shows the input workload for the communication patterns. The input workload is similar to the input for StressBench and inspiration was taken from this file for the development of StressBench.

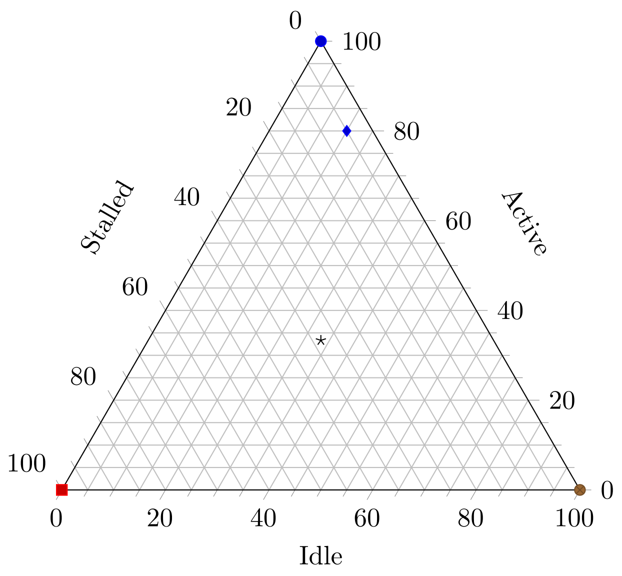


Figure 6.3: Example Ternary Plot

The state of the network ports is evaluated using the SAI metrics. Figure 6.4 shows the portion of time each of the switch ports spent in each of the SAI states for two congestion sizes at four time intervals, 0.5ms, 331ms and 662ms and the last time stamp of the simulation. The final simulated time for the 1K congestor size was 993.30ms and the simulated time for the 64K congestor was 993.31ms. In the figure the colour represents the level inside of the fat tree (green is the leaf switch, orange is the root switch). The direction of the triangle represents whether the switch port is an uplink or a downlink in the Fat Tree. The general trend as the congestor message size increases is a traffic on the links is shifted towards the stalled axis. This is because there becomes less credit for data transmission to occur so data has to reside on the switch until credit becomes available. The observed affect of increasing the congestor message size is that the links move to the stalled axis as the congestor message size increases, triangles move from the right to the left of the ternary plot.

At timestamp 331ms stalling for the leaf switches increases for the 1K congestor sizes compared to the 64K congestor sizes. The NIC ports are more active compared to a 64K congestion size which are more idle for this timestamp.

```
#Sweep
[JOB_ID] 1
[NID_LIST] 0-15
[MOTIF] Init
[MOTIF] Sweep3D pex=28 pey=32 nx=448 ny=512 nz=800 iterations=2 kba
    =10 computetime=10
[MOTIF] Fini

#Halo Exchange
[JOB_ID] 2
[NID_LIST] 21-47,72-83,99-122
[MOTIF] Init
[MOTIF] Halo2D iterations=10 compute=10 messagesizey=4000
    messagesizex=4000
[MOTIF] Fini

#AllReduce
[JOB_ID] 1
[NID_LIST] 48-71,90-97,125-140
[MOTIF] Init
[MOTIF] Allreduce iterations=1000 compute=2
[MOTIF] Fini

#Congestor
[JOB_ID] 4
[NID_LIST] 17-20,84-89,98,123,124,141-144
[MOTIF] Init
[MOTIF] Alltoall iterations=1 bytes=<Congestor Size>
[MOTIF] Fini
```

Listing 6.1: Contended Network Utilisation SST Input Workload

At 662ms more of the downlinks to NICs are stalled waiting for the data to be sent for a 1K congestor compared to the 64K where these links are more idle and active. More of the orange downlinks are waiting for the transmitting and receiving data and not as idle as the NIC ports at 64K

Understanding the state of the ports due to the congestion provides insights that could allow for scheduling improves to applications, such as delaying the application communication to reduce the effects of contention. One approach could be to stop the NIC from communicating and buffer the data on the node rather than inside of the network, similar to the idea presented by Q Liu et al. [84].

The communications also suffered a slow down with the AllReduce slowing down by 6.6μs, Figure 6.5 shows how the time for an AllReduce slowdown as the congestor size increases. The slowdown in the AllReduce operation may not seem significant but given that to solve benchmark 5, a total of 75,635 iterations are required across the 20 time steps. This results in 151,270 AllReduce operations increasing the runtime by 0.85s on top of the best case time.

6.3 StressBench as simulation validation tool

Finely controlling jobs across a machine is complicated, because jobs may not start at the same time. The communications may also not start at the same time due to computation imbalance. The initialisation may take varying times so the comparison of the communication patterns might not be correct. StressBench overcomes some of these issues and may be a suitable tool to validate congestion simulations.

The full system workload presented in Chapter 4.3.1 has been used inside of the simulator to explore whether the simulator can provide analysis in to congested workloads of real systems. The applications made use of an AllToAll, I/O workloads, Sweep3D and TeaLeaf running concurrently across the system. The same application parameters were used between the measurements and the simulations.

The Tinis platform validated in Section 5.2 was used for the hardware platform and the endpoints were provided the same job placement scheme as in Section 4.3.1.

The timings were captured by enabling the motif log for each of the jobs and comparing the time they exited from the *fini* motif to calculate the congestion impact. The simulator built on top of the SST does not provide a mechanism to extract the finish times of individual jobs, therefore the time they exit the final motif must be used to work out when the job finishes. Table 6.7 shows how the congestion impacts differs for the same workload. The I/O motifs are not shown due to unable to simulate these in a reasonable time frame; this is down to the very large message sizes which impact the performance of the simulator.

Table 6.7: Comparison of CI for Incast Traffic for Tinis

Pattern	Measured	Simulated
Sweep 1	1.0068	1.009
Sweep 2	1.0010	1.006
TeaLeaf 1	1.0172	1.001
TeaLeaf 2	1.0707	1.001
All To All 2K	1.0157	4.889
All To All 4K	1.0587	4.133

The reason for the over prediction in the congestion impact is that the routing algorithm takes the minimal path for downlinks and uplinks on the Fat Tree topology. The core switches are modelled as independent switches rather than combining the switches to match the required switch radix for the simulations. In the case of Tinis there are 12 switches with 3 ports each in the simulation rather than one 36 port switch like the real system. This means that the traffic is being forced over the same uplinks rather than being spread evenly over all available uplinks. This is a failing in the minimal path routing algorithms as they choose first shortest path rather than the equivalent shortest path.

6.4 Summary

This chapter has explored using the simulator to perform some advanced analysis at network tapering and congested workloads. This work has looked at both effects on the application level and inside of the network switches. This work has also investigated whether StressBench can be used to validate contented workloads to understand the congestion impact of workloads in the simulations.

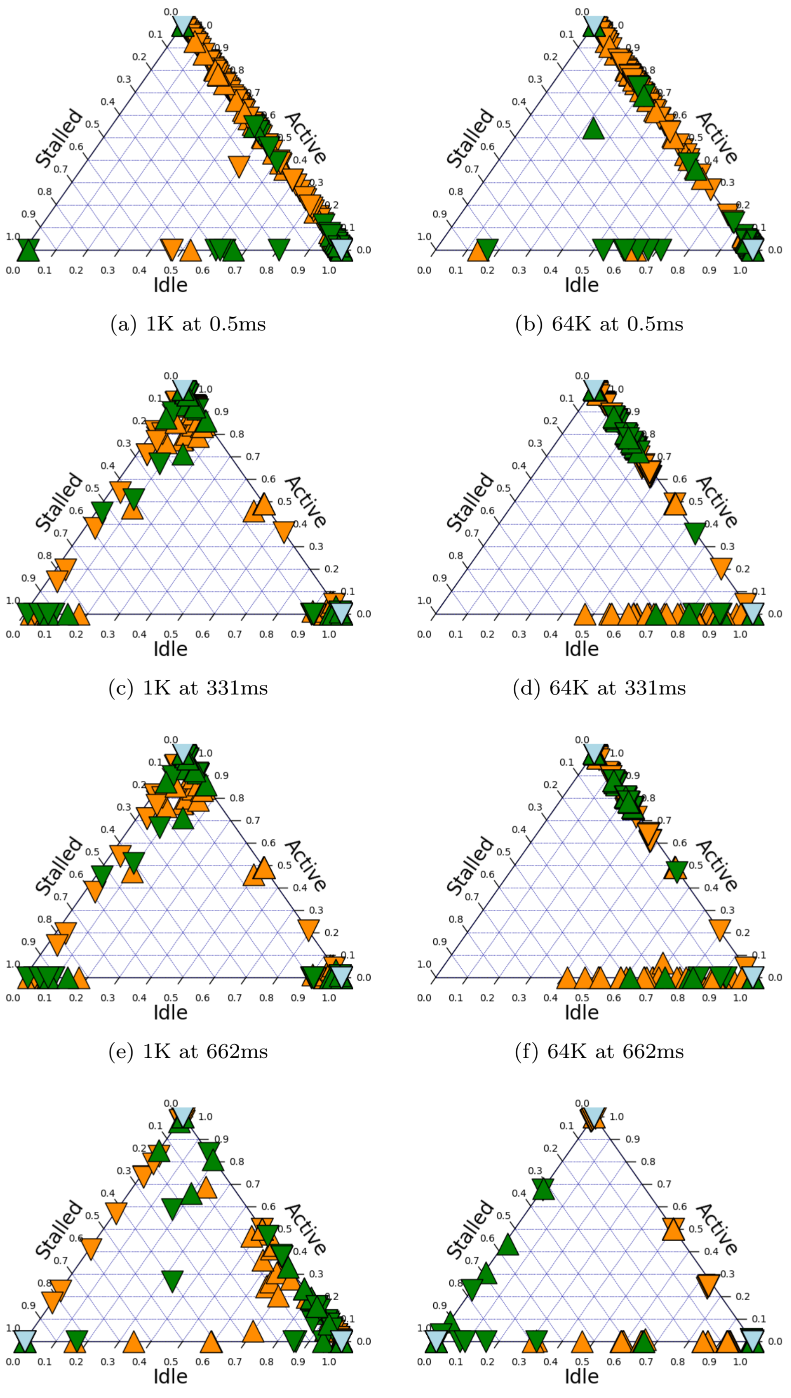
We have seen a 33.2% increase in the time to perform an AllToAll message when the tapered network is 8:1. When the system is considered as multi-user system the increase in runtime for an AllToAll can be as much as 11.4% when distributed randomly but for linear placement this is 7.4%. Contending

Sweep3D and TeaLeaf applications have a negligible effect due to the tapering because of the messages being sent are latency sensitive. This is of use when designing systems with multiple jobs as insight in to the type of applications bandwidth requirements can ensure the optimal network is designed to meet the requirements.

Available bandwidth on a 2:1 tapered Fat Tree can be reduced by as much as 2GB/s when in contention of large AllToAlls, which can slowdown large messages that are bandwidth bound. In the presented congested workload AllReduces are slowed down by 5.6μs which for entire applications can slow down the time to solution. The SAI metrics can be used to go further than just looking at the slowdown due to a congestor but for evaluation of routing algorithms. The ternary plots provide a good visual aid to see direction of the trend, an optimal system *may* spend the most time idle, or active and the shortest time stalled.

Workloads from StressBench can be translated to easily working with the simulator although care must be taken to overcome some of the limitations of the simulator, such as topology generation not equating to the same as the system. We have seen that the simulator returns pessimistic results for the congestion impact for communication patterns that require a lot of bandwidth, such as AllToAll but for structured applications the results are similar to the measured results.

This type of performance modelling is slow to simulate given that multiple endpoints inject the network with different traffic patterns which is causing contention in the network switches. Going forward simulations like this should typically focus on smaller instances of time rather than entire application or workflow to improve speed and reduce data output. Simulation of larger networks can make it difficult to see the observed effects given the vast amount of data generated.



(a) 1K at 0.5ms

(b) 64K at 0.5ms

(c) 1K at 331ms

(d) 64K at 331ms

(e) 1K at 662ms

(f) 64K at 662ms

(g) 1K at 993ms

(h) 64K at 993ms

Figure 6.4: Congestor Ternary Plot for 1K and 64K congestors, orange represent NIC ports, Green represent the level 1 ports, blue represents the core switch ports. The direction of the triangle shows whether it is an uplink or downlink

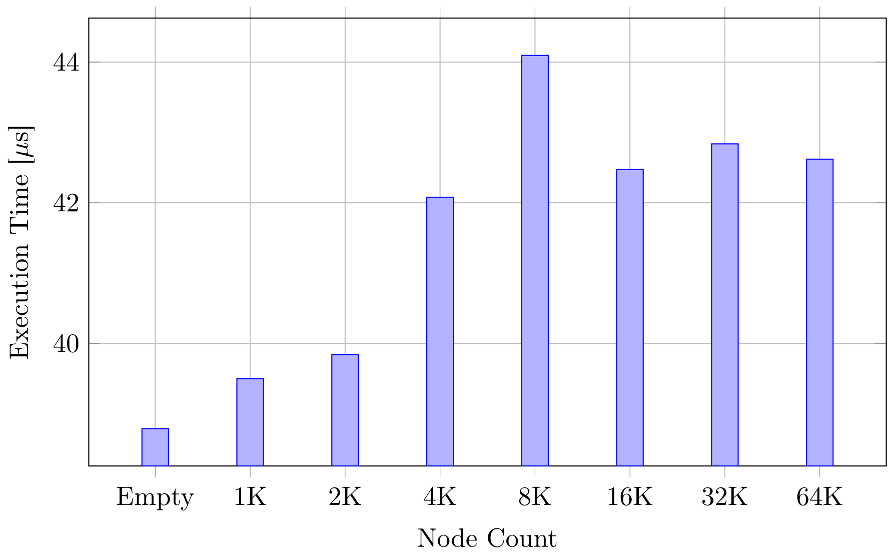


Figure 6.5: AllReduce Times against AllToAll Congestor Size

CHAPTER 7
Discussions and Conclusions

This thesis has looked at moving the state of the art network benchmarking forward, modelling applications and systems to prepare for the next era for low latency interconnection networks. Going forward it may be irrelevant to show the peak performance of a network given that multi-user system environments fail to offer this level of performance. Something which should be considered during the design and implementation phases of a new system. Simulators provide an invaluable tool to providing insight in to the goings on of a speculative design, with the improved performance of higher fidelity simulators the vast design space for a low latency interconnection network can be evaluated within reasonable timeframes. Simulation can add value to the response of an RFP as real world scenarios can be replicated and replayed to ensure that the interactions of applications is not overlooked and the performance is representative of what a user would expect from a multi-user shared environment.

One of the issues is that networks remain slow and have not advanced as quickly as processing power is memory latency, and so having to move the data to memory remains one primary reason for slow network performance. While advances in network design have improved data transfer rates and scalibility the endpoints are becoming the limiting factor, something which will get worse as networks grow in size.

Chapter 4 presents a new configurable network and I/O benchmark for stressing low latency interconnection networks, allowing domain specialisms to be abstracted to away from the network traffic and allows these to run concurrently across a system. This also shows the need to use real I/O traffic rather than incast traffic generators as they often fail to saturate the network sufficiently.

Chapter 5 has presented a validation methodology for benchmarking and modelling systems which has been proven for four systems. The first performance model of TeaLeaf has been presented and a performance model of Sweep3D has been shown built on top of the SST. Some of the Exascale low latency interconnection design space has been explored to evaluate whether higher radix switches become cost effective for Dragonfly and Fat Tree topologies.

Chapter 6 looks at pushing existing validated models to understand interactions between communication patterns through simulation, called contention aware performance modelling. This contention aware performance modelling has looked at the effects on an application's time to solution for tapered Fat Trees and the effects of contention within a network switch. Comparisons are drawn upon for using StressBench as a validation tool for the contention aware performance models.

The remainder of this chapter will focus on the limitations of the thesis (Section 7.1). In Section 7.2 discusses the future work.

7.1 Limitations

This work focuses on the time to solution (the execution time for an application) as a performance metric, while it shows any performance gains it overlooks other useful measurements such as power consumption and cost. Practical configurations are also overlooked such wiring larger switch radii leads to complications given that wires should be the same length to ensure that they have the same latency, longer cables can be come an issue to bundle in a space constrained server rack.

7.1.1 Software

The majority of the applications used throughout this thesis are structured, this means that message sizes are typically the same size across the same dimension, this is is not the case for unstructured applications.

The work in this has primarily focused on MPI structured applications, such as Sweep3D and TeaLeaf which simulate physics applications. While they are relevant they do not reflect all types of applications that are typically run on HPC systems. The applications used throughout this thesis overlook some other disciplines such as Biology, Chemistry and Machine Learning applications that are now becoming widely used. The work still has relevance as some of the communication patterns used in this thesis are still common place in these domain areas. This work has overlooked other widely used parallel programming paradigms such as SHMEM and Partitioned Global Address Space (PGAS).

7.1.2 Hardware

The systems used during the course of this thesis vary in size, network technology and CPU, while the majority of analysis was performed on small systems (less than 512 nodes). This is because there are issues with running StressBench at scale, such as a large number of processes reading from a file can cause a file system to hang as the traffic acts like a denial of service attack. One approach to resolve this would look at starting applications with SPINDLE from Lawrence Livermore National Laboratory [58]. A large number of processes are slow to initialise and can take a significant time to start running, this is an active area of research as systems grow in size to reach exascale there will be more processes. Process Management Interface - Exascale (PMIx) is under active development looking at the effects of large process counts and how to speed up application start up for exascale [31].

7.2 Future Work

The future work highlights where benchmarking and modelling should go as such we discuss both independently.

7.2.1 Benchmarking

This thesis has presented a MPI and I/O benchmark called StressBench which can stress low latency interconnection networks. StressBench (in its current form) has severed its purpose to understand network interactions for this thesis and valuable studies to gain an understanding of communication traffic interactions with I/O traffic.

In Chapter 4 it is stated that applications may be less impacted by I/O traffic when placed interleaved on a Dragonfly rather than linearly, with the aforementioned enhancements to StressBench we can now test this hypothesis.

TeaLeaf and CloverLeaf were the patterns of interest rather than just the composite parts as in Section 4.3.2. The same file sizes were used on Isambard as the I/O study on the communication patterns (see Section 4.3.2). TeaLeaf benchmark 5 was used for this study, and CloverLeaf used benchmark 64.

Table 7.1 shows how the applications perform against different file sizes and job placements on Isambard. The trend matches that seen in Chapter 4, this shows that the patterns themselves are sufficient to perform a comparison. This shows that application performance can be improved by using interleaved job placement when contending perform significant I/O operations, such as large write operations.

Table 7.1: Comparison of Applications against Congestion Impact - Isambard

		CloverLeaf	TeaLeaf
Interleaved	1GB	1.036	1.041
	10GB	1.029	1.067
	100GB	1.037	1.036
	500GB	1.060	1.052
Linear	1GB	1.083	1.061
	10GB	1.090	1.083
	100GB	1.123	1.281
	500GB	1.156	1.135
Random	1GB	1.096	1.142
	10GB	1.085	1.021
	100GB	1.114	1.130
	500GB	1.265	1.213

Studies involving communication patterns lead to the same results, demonstrating that for benchmarking just communication patterns is a suitable approach compared to entire applications. There are several advantages to this approach, firstly the domain complexity can be removed as there is no need for the computation this reduces the time the benchmark takes to run as the entire application needs not be run. Secondly this allows for exploration of how different implementations of communication patterns affect the performance to see if any performance gain can be achieved by changing the implementation without rewriting the application.

In the current implementation it is severely limited in functionality due to design choices during the early development, these include a text based input that is manually parsed. One improvement may be to create an interface using Python so that some flexibility can be added to input decks for generating the more complicated workflows, similar to how inputs are created for *Ember*, *Firefly* and *Merlin*.

Another limitation of the current design is that for large *jobs*, buffers may consume large amounts of RAM as each *motif* has their own buffer. One improvement may be that a job takes a problem size parameter and defines job local arrays that are used by all motifs, Listing 7.1 demonstrates a small example. This would significantly reduce the memory footprint of the benchmark framework, although this *may* change the performance characteristics of the benchmark compared to the communication pattern that is trying to be replicated.

```
[JOB_NAME] TeaLeaf_CG_Iteration
[PARAMS] -x 4000 -y 4000 -iterations 10
[NID_LIST] 0,1,2,3
[MOTIF] AllReduce
[MOTIF] AllReduce
[MOTIF] 2DHalo
```

Listing 7.1: Proposed Parameter Extension to StressBench

The current implementation of StressBench does not support hybrid programming models (such as MPI + OpenMP) something which applications have shifted towards in an effort to improve performance of applications. Another hybrid model is MPI on GPU which can support multiple GPUs across multiple nodes with a technology called GPUDirect [102]. Given this rise of popularity of programming models being able to use these as part of the benchmark suite will provide further realistic insight in to network usage, such as network contention, and CPU-NIC bus utilisation.

One of the things that may cause StressBench to fail as a benchmark is its adoption in the benchmark field. This is considered a limitation because it provides a benchmark framework which may be too customisable compare to something like GPCNeT. If consumers contribute communication patterns and encourage results during a RFP they will be able to make more informed decisions to their procurement process. If the ideas from StressBench were to merge with another MPI benchmark suite then the framework may be formally adopted by the community.

7.2.2 Modelling

The work presented in this thesis has validated the four elements for SST. This approach provides an accurate approximation for modelling a network and the associated software stack but lacks some features of modern HPC systems.

Firstly, the endpoints modelling are primarily CPU based, which provides useful understanding for current and previous systems. The heterogeneity of systems is increasing as we approach the exascale era, therefore considering how MPI on GPU performance affects the software and hardware interactions will become key when modelling future applications. Another issue is that I/O endpoints are not currently modelled; the placement of these I/O nodes can have effects on the performance of network traffic (see Section 4.3.2). Allowing the option to situate the I/O nodes could allow the simulator to become a vital tool during procurement for HPC systems.

The simulator provides capability to support both hardware and software co-design. This functionality can aid the design of future algorithms to understand how they will perform of different topologies. This should not just be limited to algorithms and communication patterns but applied to paradigms as well to understand if there is a better approach, this is of significant interest as system architecture changes to reach exascale level of performance.

This work has primarily used the latency or measured bandwidth to assess there correctness of results, these measurements are taken at the application layer, this means that errors cannot be broken down on a per layer basis. Potential advances in telemetry inside of the network and lower software layers may enable a finer granularity to capture and assess the error at different layers in the networking stack.

The use of contention aware performance modelling demonstrated in this thesis just scratches the surface of the area. The complexity introduced could begin to look at what improvements to flow control algorithms could be developed to improve performance of link contention. Another consideration is to rapidly evaluate routing algorithms to look at congestion avoidance during traffic movement, this may be a semi-adapative routing solution.

Bibliography

[1] Infiniband architecture specification volume 1 - general specifications. *InfiniBand Architecture Specification*, 1:1–4379, 2015.

[2] Openmp application programming interface. *OpenMP 4.5 Complete Specifications)*, 2015.

[3] Ieee standard for ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, 2018.

[4] Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, 2021.

[5] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, and A. Sussman. Tuning the performance of i/o-intensive parallel applications. In *Proceedings of the fourth workshop on I/O in parallel and distributed systems: part of the federated computing research conference*, pages 15–27, 1996.

[6] V. S. Adve. *Analyzing the Behavior and Performance of Parallel Programs.* PhD thesis, University of Wisconsin - Madison, October 1993.

[7] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. Hyperx: Topology, routing, and packaging of efficient large-scale networks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. Association for Computing Machinery.

[8] S. R. Alam, J. A. Kuehn, R. F. Barrett, J. M. Larkin, M. R. Fahey, R. Sankaran, and P. H. Worley. Cray xt4: an early evaluation for petascale scientific simulation. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–12, 2007.

[9] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. Loggp: Incorporating long messages into the logp model for parallel computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.

[10] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967,*

Spring Joint Computer Conference, AFIPS '67 (Spring), page 483–485, New York, NY, USA, 1967. Association for Computing Machinery.

[11] H. Anzt, E. Boman, R. Falgout, P. Ghysels, M. Heroux, X. Li, L. Curfman McInnes, R. Tran Mills, S. Rajamanickam, K. Rupp, B. Smith, I. Yamazaki, and U. Meier Yang. Preparing sparse solvers for exascale computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190053, 2020.

[12] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al. The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.

[13] B. W. Barrett and K. S. Hemmert. An application based mpi message throughput benchmark. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–8, 2009.

[14] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, M. Snir, et al. Taming parallel i/o complexity with auto-tuning. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2013.

[15] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2013.

[16] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P.-T. Bremer, M. Schulz, and L. V. Kale. Identifying the culprits behind network congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 113–122. IEEE, 2015.

[17] R. F. Bird, S. A. Wright, D. A. Beckingsale, and S. A. Jarvis. Performance modelling of magnetohydrodynamics codes. In M. Tribastone and S. Gilmore, editors, *Computer Performance Engineering*, pages 197–209, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak. Intel® omni-path architecture: Enabling scalable, high performance fabrics. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 1–9, 2015.

[19] A. S. Bland, J. C. Wells, O. E. Messer, O. R. Hernandez, and J. H. Rogers. Titan: Early experience with the cray xk6 at oak ridge national

laboratory. In *Proceedings of cray user group conference (CUG 2012)*, pages 3–4. Cray User Group Stuttgart, Germany, 2012.

[20] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.-K. Su. Myrinet: a gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[21] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz. Caliper: performance introspection for hpc software stacks. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 550–560. IEEE, 2016.

[22] D. Böhme, T. Gamblin, P.-T. Bremer, O. Pearce, and M. Schulz. Caliper: Composite performance data collection in HPC codes, 2015.

[23] K. Boland and A. Dollas. Predicting and precluding problems with memory latency. *IEEE Micro*, 14(4):59–67, 1994.

[24] E. G. Boman, Ü. V. Çatalyürek, C. Chevalier, and K. D. Devine. The zoltan and isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[25] P. J. Braam and P. Schwan. Lustre: The intergalactic file system. In *Ottawa Linux Symposium*, volume 8, pages 3429–3441, 2002.

[26] R. Brightwell, K. Pedretti, K. Underwood, and T. Hudson. Seastar interconnect: Balanced bandwidth for scalable performance. *IEEE Micro*, 26(3):41–57, 2006.

[27] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. Storage*, 7(3), Oct. 2011.

[28] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale i/o workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, 2009.

[29] C. D. Carothers, D. Bauer, and S. Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.

[30] E. Carson, N. Knight, and J. Demmel. An efficient deflation technique for the communication-avoiding conjugate gradient method. *Electronic Transactions on Numerical Analysis*, 43(125141):09, 2014.

[31] R. H. Castain, J. Hursey, A. Bouteiller, and D. Solt. Pmix: Process management for exascale environments. *Parallel Computing*, 79:9–29, 2018.

[32] D. G. Chester, T. L. Groves, S. D. Hammond, T. R. Law, S. A. Wright, R. P. Smedley-Stevenson, S. A. Fahmy, G. R. Mudalige, and S. A. Jarvis. StressBench: A Configurable Full System Network and I/O Benchmark Framework. In *Proceedings of ISC HIGH PERFORMANCE 2021*, 2021.

[33] D. G. Chester, T. L. Groves, S. D. Hammond, T. R. Law, S. A. Wright, R. P. Smedley-Stevenson, S. A. Fahmy, G. R. Mudalige, and S. A. Jarvis. StressBench: A Configurable Full System Network and I/O Benchmark Framework. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021.

[34] D. G. Chester, S. A. Wright, S. D. Hammond, T. R. Law, R. P. Smedley-Stevenson, S. Maheswaran, and S. A. Jarvis. Full-system modeling and simulation : contributions towards coupling, contention, and I/O. In *MODSIM 2019*, 2019.

[35] D. G. Chester, S. A. Wright, and S. A. Jarvis. Understanding communication patterns in HPCG. *Electronic Notes in Theoretical Computer Science*, 340:55–65, 2018.

[36] S. Chunduri, T. Groves, P. Mendygral, B. Austin, J. Balma, K. Kandalla, K. Kumaran, G. Lockwood, S. Parker, S. Warren, et al. Gpcnet: designing a benchmark suite for inducing and measuring contention in hpc networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–33, 2019.

[37] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran. Run-to-run variability on xeon phi based xc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2017.

[38] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross. Codes: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies*. ACM, 2011.

[39] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '93, page 1–12, New York, NY, USA, 1993. Association for Computing Machinery.

[40] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, July 1993.

[41] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.

[42] W. J. Dally. Virtual-channel flow control. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ISCA '90, page 60–68, New York, NY, USA, 1990. Association for Computing Machinery.

[43] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[44] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[45] F. Darema, D. George, V. Norton, and G. Pfister. A single-program-multiple-data computational model for epex/fortran. *Parallel Computing*, 7(1):11–24, 1988.

[46] P. De, V. Mann, and U. Mittaly. Handling os jitter on multicore multithreaded systems. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–12. IEEE, 2009.

[47] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

[48] T. M. Declerck et al. Using robinhood to purge data from lustre file systems. *Proceedings of the 2014 User Group, Lugano*, 2014.

[49] F. Dekking, C. Kraaikamp, H. Lopuhaä, and L. Meester. *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Springer Texts in Statistics. Springer London, 2006.

[50] J. Dickson, S. A. Wright, D. Harris, S. Maheswaran, J. Herdman, M. C. Miller, and S. A. Jarvis. Enabling portable i/o analysis of commercially sensitive hpc applications through workload replication. *Cray User Group*, pages 1–14, 2017.

[51] D. W. Doerfler. Trinity: Next-generation supercomputer for the asc program. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2014.

[52] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK users' guide*. SIAM, 1979.

[53] M. R. Dorr and C. H. Still. Concurrent source iteration in the solution of three-dimensional, multigroup discrete ordinates neutron transport equations. *Nuclear Science and Engineering*, 122(3):287–308, 1996.

[54] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard. Cray cascade: A scalable hpc system based on a dragonfly network. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–9, 2012.

[55] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[56] S. Fogerty, M. Martineau, R. Garimella, and R. Robey. A comparative study of multi-material data structures for computational physics applications. *Computers & Mathematics with Applications*, 78(2):565–581, 2019. Proceedings of the Eight International Conference on Numerical Methods for Multi-Material Fluid Flows (MULTIMAT 2017).

[57] M. Forum. MPI: A Message-Passing Interface Standard Version 3.1. Technical report, MPI Forum, June 2015.

[58] W. Frings, D. H. Ahn, M. LeGendre, T. Gamblin, B. R. de Supinski, and F. Wolf. Massively parallel loading. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, page 389–398, New York, NY, USA, 2013. Association for Computing Machinery.

[59] Glenn Lockwood. Inode sizes on NERSC's production file systems. `https://zenodo.org/record/2530940` (accessed December 17, 2020), 2019.

[60] O. I. W. Group. ofiwg/libfabric: Open Fabric Interfaces. `https://github.com/ofiwg/libfabric` (Accessed 27th September 2021), 2021.

[61] T. Groves, R. E. Grant, S. Hemmer, S. Hammond, M. Levenhagen, and D. C. Arnold. (sai) stalled, active and idle: Characterizing power and performance of large-scale dragonfly networks. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 50–59. IEEE, 2016.

[62] J. L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5):532–533, May 1988.

[63] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann. Hacc: Extreme scaling and performance across diverse architectures. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10. IEEE, 2013.

[64] M. Haller and T. Worsch. Skampi—including more complex communication patterns. In *High Performance Computing in Science and Engineering'03*, pages 455–466. Springer, 2003.

[65] S. D. Hammond, G. R. Mudalige, J. Smith, J. Davis, S. A. Jarvis, J. Holt, I. Miller, J. Herdman, and A. Vadgama. To upgrade or not to upgrade? catamount vs. cray linux environment. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8. IEEE, 2010.

[66] S. D. Hammond, G. R. Mudalige, J. Smith, S. A. Jarvis, J. Herdman, and A. Vadgama. Warpp: a toolkit for simulating high-performance parallel scientific codes. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, page 19, 2009.

[67] M. A. Heroux and J. Dongarra. Toward a new metric for ranking high performance computing systems. *SAND2013-4744*, 2013.

[68] A. S. C. Initiative et al. The ASCI SWEEP3D benchmark code, 1995.

[69] Intel. Intel MPI Benchmarks. `https://software.intel.com/en-us/imb-user-guide` (accessed September 20, 2020), 2020.

[70] Intel. Intel Trace Analyzer and Collector. `https://software.intel.com/en-us/intel-trace-analyzer` (accessed September 20, 2021), 2021.

[71] N. Jain, A. Bhatele, S. White, T. Gamblin, and L. V. Kale. Evaluating hpc networks via simulation of parallel workloads. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165. IEEE, 2016.

[72] R. Jain and R. JAIN. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley professional computing. Wiley, 1991.

[73] H.-W. Jin, S. Sur, L. Chai, and D. K. Panda. Lightweight kernel-level primitives for high-performance mpi intra-node communication over multi-core systems. In *2007 IEEE International Conference on Cluster Computing*, pages 446–451, 2007.

[74] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. Still. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)*, Boston, USA, May 2013.

[75] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks (1976)*, 3(4):267–286, 1979.

[76] J. Kim, W. Dally, S. Scott, and D. Abts. Cost-efficient dragonfly topology for large-scale systems. *IEEE Micro*, 29(1):33–40, 2009.

[77] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77–88. IEEE, 2008.

[78] B. Klenk and H. Fröning. An overview of mpi characteristics of exascale proxy applications. In *International Supercomputing Conference*, pages 217–236. Springer, 2017.

[79] S. Knight, J. P. Kenny, and J. J. Wilke. Supercomputer in a laptop: Distributed application and runtime development via architecture simulation. In *International Conference on High Performance Computing*, pages 347–359. Springer, 2018.

[80] K. R. Koch, R. S. Baker, and R. E. Alcouffe. Solution of the first-order form of the 3-d discrete ordinates equation on a massively parallel processor. *Transactions of the American Nuclear Society*, 65(108):198–199, 1992.

[81] P. Kogge. The tops in flops. *IEEE Spectrum*, 48(2):48–54, 2011.

[82] S. H. Lavington. *A history of Manchester computers.* NCC Publications, 1975.

[83] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[84] Q. Liu and R. D. Russell. Rgbcc: A new congestion control mechanism for infiniband. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 91–100, 2016.

[85] G. K. Lockwood, K. Lozinskiy, L. Gerhardt, R. Cheema, D. Hazen, and N. J. Wright. A quantitative approach to architecting all-flash lustre file systems. In *International Conference on High Performance Computing*, pages 183–197. Springer, 2019.

[86] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran. Benchmarking machine learning methods for performance modeling of scientific applications. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 33–44, 2018.

[87] A. Mallinson, D. A. Beckingsale, W. Gaudin, J. Herdman, J. Levesque, and S. A. Jarvis. Cloverleaf: Preparing hydrodynamics codes for exascale. *The User Group*, 2013, 2013.

[88] P. Marendić, J. Lemeire, T. Haber, D. Vučinić, and P. Schelkens. An investigation into the performance of reduction algorithms under load imbalance. In *European Conference on Parallel Processing*, pages 439–450. Springer, 2012.

[89] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ISCA '97, page 85–97, New York, NY, USA, 1997. Association for Computing Machinery.

[90] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. *SIGARCH Comput. Archit. News*, 25(2):85–97, May 1997.

[91] M. Martinasso and J.-F. Méhaut. A contention-aware performance model for hpc-based networks: A case study of the infiniband network. In *European Conference on Parallel Processing*, pages 91–102. Springer, 2011.

[92] J. D. McCalpin. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what*, 22:7, 1995.

[93] J. D. McCalpin. Memory bandwidth and system balance in hpc systems. *Invited talk, Supercomputing*, 2016.

[94] S. McIntosh-Smith, M. Martineau, T. Deakin, G. Pawelczak, W. Gaudin, P. Garrett, W. Liu, R. Smedley-Stevenson, and D. Beckingsale. Tealeaf: a mini-application to enable design-space explorations for iterative sparse linear solvers. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 842–849. IEEE, 2017.

[95] S. McIntosh-Smith, M. Martineau, T. Deakin, G. Pawelczak, W. Gaudin, P. Garrett, W. Liu, R. Smedley-Stevenson, and D. Beckingsale. Tealeaf: A mini-application to enable design-space explorations for iterative sparse linear solvers. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 842–849, Sep. 2017.

[96] L. W. McVoy, C. Staelin, et al. lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pages 279–294. San Diego, CA, USA, 1996.

[97] J. Morel. Deterministic transport methods and codes at los alamos. Technical report, Los Alamos National Lab., NM (US), 1999.

[98] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns. Using massively parallel simulation for MPI collective communication modeling in extreme-scale networks. In *Proceedings of the Winter Simulation Conference 2014*, pages 3107–3118. IEEE, 2014.

[99] M. Mubarak, N. Jain, J. Domke, N. Wolfe, C. Ross, K. Li, A. Bhatele, C. D. Carothers, K.-L. Ma, and R. B. Ross. Toward reliable validation of hpc network simulation models. In *2017 Winter Simulation Conference (WSC)*, pages 659–674, 2017.

[100] G. R. Mudalige, M. K. Vernon, and S. A. Jarvis. A plug-and-play model for evaluating wavefront computations on parallel architectures. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–14. IEEE, 2008.

[101] T. Nesson and S. L. Johnsson. Romm routing on mesh and torus networks. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, page 275–287, New York, NY, USA, 1995. Association for Computing Machinery.

[102] NVIDIA. GPUDirect | NVIDIA Developer. `https://developer.nvidia.com/gpudirect` (accessed September 15, 2021), 2021.

[103] Oak Ridge National Laboratory. Summit – Oak Ridge Leadership Computing Facility. `https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/` (accessed September 15, 2021), 2021.

[104] Ohio State University. OSU Micro-Benchmarks. `http://mvapich.cse.ohio-state.edu/benchmarks/` (accessed September 20, 2020), 2020.

[105] A. M. B. Owenson, S. A. Wright, R. A. Bunt, Y. K. Ho, M. J. Street, and S. A. Jarvis. An unstructured cfd mini-application for the performance prediction of a production cfd code. *Concurrency and Computation: Practice and Experience*, 32(10):e5443, 2020. e5443 cpe.5443.

[106] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design The Hardware/Software Interface.* Morgan Kaufmann, fourth edition, 2012.

[107] H. Pritchard, I. Gorodetsky, and D. Buntinas. A ugni-based mpich2 nemesis network module for the xe. In *European MPI Users' Group Meeting*, pages 110–119. Springer, 2011.

[108] M. S. Rahman, S. Bhowmik, Y. Ryasnianskiy, X. Yuan, and M. Lang. Topology-custom ugal routing on dragonfly. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2019.

[109] J. Reinders. Intel AVX-512 Instructions. `https://software.intel.com/content/www/us/en/develop/articles/intel-avx-512-instructions.html` (Accessed 27th September 2021), 2013.

[110] R. Reussner, P. Sanders, L. Prechelt, and M. Müller. Skampi: A detailed, accurate mpi benchmark. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 52–59. Springer, 1998.

[111] R. M. Russell. The cray-1 computer system. *Commun. ACM*, 21(1):63–72, Jan. 1978.

[112] Y. Saad. *Iterative methods for sparse linear systems.* SIAM, second edition, 2003.

[113] Sandia National Laboratories. Ember Communication Pattern Library. `https://github.com/sstsimulator/ember` (accessed December 17, 2020), 2018.

[114] F. B. Schmuck and R. L. Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, 2002.

[115] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, et al. Ucx: an open source framework for hpc network apis and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE, 2015.

[116] A. Shpiner, Z. Haramaty, S. Eliad, V. Zdornov, B. Gafni, and E. Zahavi. Dragonfly+: Low cost topology for scaling datacenters. In *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pages 1–8, 2017.

[117] E. Slaughter, W. Wu, Y. Fu, L. Brandenburg, N. Garcia, W. Kautz, E. Marx, K. Morris, Q. Cao, G. Bosilca, S. Mirchandaney, W. Lee, S. Treichler, and U. Patrick McCormick pat@lanl.gov Los Alamos National Laboratory. Task bench: A parameterized benchmark for evaluating parallel runtime performance. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 864–878, Los Alamitos, CA, USA, nov 2020. IEEE Computer Society.

[118] A. Spector and D. Gifford. The space shuttle primary computer system. *Commun. ACM*, 27(9):872–900, Sept. 1984.

[119] H. Sullivan and T. R. Bashkow. A large scale, homogeneous, fully distributed parallel machine, i. In *Proceedings of the 4th Annual Symposium on Computer Architecture*, ISCA '77, page 105–117, New York, NY, USA, 1977. Association for Computing Machinery.

[120] H. Sullivan and T. R. Bashkow. A large scale, homogeneous, fully distributed parallel machine, i. *SIGARCH Comput. Archit. News*, 5(7):105–117, Mar. 1977.

[121] TOP500.org. June 2014 - TOP500 Supercomputers. `https://www.top500.org/lists/top500/2014/06/` (accessed September 15, 2021), 2014.

[122] TOP500.org. November 2016 - TOP500 Supercomputers. `https://www.top500.org/lists/top500/2016/11/` (accessed December 8, 2021), 2016.

[123] TOP500.org. June 2021 - TOP500 Supercomputers. `https://www.top500.org/lists/top500/2021/06/` (accessed September 15, 2021), 2021.

[124] K. Underwood and R. Brightwell. The impact of mpi queue usage on message latency. In *International Conference on Parallel Processing, 2004. ICPP 2004.*, pages 152–160 vol.1, 2004.

[125] L. G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.

[126] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Aug. 1990.

[127] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, page 263–277, New York, NY, USA, 1981. Association for Computing Machinery.

[128] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. Berendsen. Gromacs: fast, flexible, and free. *Journal of computational chemistry*, 26(16):1701–1718, 2005.

[129] S. A. Wright and S. A. Jarvis. Quantifying the effects of contention on parallel file systems. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 932–940, 2015.

[130] X. Wu, V. Deshpande, and F. Mueller. Scalabenchgen: Auto-generation of communication benchmarks traces. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1250–1260. IEEE, 2012.

[131] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan. Watch out for the bully! job interference study on dragonfly network. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 750–760, 2016.

[132] W. Yu, J. S. Vetter, and H. S. Oral. Performance characterization and optimization of parallel i/o on the xt. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–11. IEEE, 2008.

APPENDIX A
Applications inside of StressBench

A.1 TeaLeaf Motif Implementation

```
int TL_Decompose(MPI_Comm communicator, void** data, char** params,
     int numberOfParams) {
  int opt;
  char* inputFile = 0;
  char* outputFile = 0;
  TeaLeafData* in = malloc(sizeof(TeaLeafData));
  while((opt = getopt(numberOfParams, params, "i:o:")) != -1) {
    switch(opt) {
      case 'i':
        inputFile = malloc((strlen(optarg)+1)*sizeof(char));
        strcpy(inputFile, optarg);
        break;
      case 'o':
        outputFile = malloc((strlen(optarg)+1)*sizeof(char));
        strcpy(outputFile, optarg);
        break;
    }
  }

  // Create the settings wrapper
  Settings* settings = (Settings*)malloc(sizeof(Settings));
  set_default_settings(settings);
  if(inputFile != NULL) {
    settings->tea_in_filename = inputFile;
  }

  if(outputFile != NULL) {
    settings->tea_out_filename = outputFile;
  }
  // Fill in rank information
  initialise_ranks(settings, communicator);

  // Perform initialisation steps
  Chunk* chunks;
  initialise_application(&chunks, settings);
  in->chunks = chunks;
  in->settings = settings;
  *data = in;
  return 0;
}

int TL_Perform(MPI_Comm communicator, void* data) {
  TeaLeafData* in = (TeaLeafData*)data;
  // Perform the solve using default or overloaded diffuse
#ifndef DIFFUSE_OVERLOAD
  diffuse(in->chunks, in->settings);
#else
  diffuse_overload(in->chunks, in->settings);
#endif
}

int TL_Destruct(MPI_Comm communicator, void* data) {
  TeaLeafData* in = (TeaLeafData*)data;
  // Print the kernel-level profiling results
  if(in->settings->rank == MASTER)
  {
    PRINT_PROFILING_RESULTS(in->settings->kernel_profile);
  }

  // Finalise the kernel
  kernel_finalise_driver(in->chunks, in->settings);

  // Finalise each individual chunk
  for(int cc = 0; cc < in->settings->num_chunks_per_rank; ++cc)
  {
    finalise_chunk(&(in->chunks[cc]));
    free(&(in->chunks[cc]));
  }

  // Finalise the application
  free(in->settings);
}
```

Listing A.1: TeaLeaf Motif

A.2 TeaLeaf Settings Structure

```c
// The main settings structure
typedef struct Settings
{
    // Set of system-wide profiles
    struct Profile* kernel_profile;
    struct Profile* application_profile;
    struct Profile* wallclock_profile;

    // Log files
    FILE* tea_out_fp;

    // Solve-wide constants
    int rank;
    int end_step;
    int presteps;
    int max_iters;
    int coefficient;
    int ppcg_inner_steps;
    int summary_frequency;
    int halo_depth;
    int num_states;
    int num_chunks;
    int num_chunks_per_rank;
    int num_ranks;
    bool* fields_to_exchange;

    bool is_offload;

    bool error_switch;
    bool check_result;
    bool preconditioner;

    double eps;
    double dt_init;
    double end_time;
    double eps_lim;

    // Input-Output files
    char* tea_in_filename;
    char* tea_out_filename;
    char* test_problem_filename;

    Solver solver;
    char* solver_name;

    Kernel_Language kernel_language;

    // Field dimensions
    int grid_x_cells;
    int grid_y_cells;

    double grid_x_min;
    double grid_y_min;
    double grid_x_max;
    double grid_y_max;

    double dx;
    double dy;
    MPI_Comm communicator;

} Settings;
```

Listing A.2: Modified Settings Structure