



DEPARTMENT OF COMPUTER SCIENCE

HPC in the Cloud

A Performance Comparison Between Isambard ThunderX2 and Amazons Graviton3 for HPC Workloads



A dissertation submitted to the University of Bristol in accordance with the requirements of the degree
of Master of Science in the Faculty of Engineering.

Thursday 18th May, 2023


Abstract

The demand for High-Performance Computing (HPC) has seen a significant increase, with high demand from the scientific and machine learning sectors. This increasing demand has prompted cloud companies to offer HPC services hosted in data centers, along with the pay-for-what-you use model, providing an interesting alternative to the traditional supercomputer model and raising questions about performance differences between the two approaches. This project aimed to answer these questions, by evaluating the performance of Amazon's latest cloud High-Performance Computing (HPC) service, Graviton3, in comparison to Isambard, a traditional HPC system. The objective was to assess the viability of cloud-based HPC as an alternative to the traditional supercomputer model. The systems were evaluated using a range of synthetic and application-based benchmarks, including STREAM, Intel MPI benchmarks, and SPEChpc2021, giving an insight into performance characteristics of each system. As a result, I was able to conclude that Amazon's cloud HPC using Graviton, is currently not an alternative to Isambard due to inter-node communication bottlenecks.

- I ran a suite benchmark which totaled more than 750 compute hours of total run time
- Through the evaluation of the systems, I demonstrated that Graviton3 has relatively significant inter-node communication bottlenecks.
- I was able to conclude that the Graviton3 processor is a significant improvement over ThunderX2, however due to inter-node communication bottlenecks, the viability as an alternative to Isambard is not clear.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

 Thursday 18th May, 2023

Contents

1	Introduction	1
1.1	HPC	1
1.2	ARM	1
1.3	Cloud Computing	2
1.4	Benchmarking	3
1.5	Aims and objectives	4
2	Background	5
2.1	High-Performance Computing	5
2.2	Programming Models	6
2.3	Benchmarks	7
2.4	Program Profiling	11
2.5	Overview Of Systems	11
3	Results	13
3.1	STREAM	13
3.2	Intel MPI	15
3.3	SPEChpc2021	17
4	Conclusion	26
4.1	Contributions and Achievements	26
4.2	Project Status	26
4.3	Future Plans	27

List of Figures

3.1	STREAM Benchmark showing the relative measured memory bandwidth across all kernels for Graviton3 normalised to ThunderX2	14
3.2	STREAM Benchmark showing the peak measured memory bandwidth across all kernels for Graviton3 and ThunderX2 in GB/s	14
3.3	Intel MPI Benchmark showing the relative performance of MPI.Sendrecv on Graviton3, normalised to ThunderX2. 64 ranks per node	15
3.4	Intel MPI Benchmark showing the relative performance of MPI.Allreduce on Graviton3, normalised to ThunderX2. 64 ranks per node	16
3.5	Intel MPI Benchmark showing the relative performance of MPI.Barrier on Graviton3, normalised to ThunderX2. 64 ranks per node	17
3.6	MPI functions as a percentage of run time on Isambard for tiny suite. Captured on 4 nodes on MPI-only model with Cray Pat	18
3.7	MPI functions as a percentage of run time on Graviton3 for tiny suite. Captured on 4 nodes on MPI-only model with Tau	19
3.8	MPI functions as a percentage of run time on Isambard for small suite. Captured on 8 nodes on MPI-only model with Cray Pat	19
3.9	MPI functions as a percentage of run time on Graviton3 for small suite. Captured on 8 nodes on MPI-only model with Tau	20
3.10	Scaling performance of MPI-only on ThunderX2 for tiny suite	21
3.11	Scaling performance of MPI-only on ThunderX2 for small suite	21
3.12	Relative scaling performance of MPI-only on Graviton3 for tiny suite, normalised to ThunderX2	22
3.13	Relative scaling performance of MPI-only on Graviton3 for small suite, normalised to ThunderX2. 605.lbm.s is unable to run on 4 nodes on Graviton due to exceeded memory capacity	22
3.14	MPI functions as a percentage of run time on Isambard for small suite. Captured on 8 nodes on MPI+OpenMP with Cray Pat. 619.clvleaf.s is missing due to Cray Pat unable to run MPI+OpenMP model	23
3.15	MPI functions as a percentage of run time on ThunderX2 for small suite. Captured on 8 nodes on MPI+OpenMP with TAU.	23
3.16	Relative scaling performance of MPI+OpenMP on Graviton3 for tiny suite, normalised to ThunderX2	24
3.17	Relative scaling performance of MPI+OpenMP on Graviton3 for small suite, normalised to ThunderX2. 605.lbm.s is unable to run on 4 nodes on Graviton due to exceeded memory capacity	25

List of Tables

2.1	SPEChpc 2021 benchmarks application properties	10
2.2	SPEChpc 2021 System requirements	11
2.3	Overview of System Specifications	12

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Dr Tom Deakin

Supporting Technologies

- The STREAM benchmark was used to measure memory bandwidth <https://www.cs.virginia.edu/stream/>
- Intel MPI benchmarks were used to evaluate the performance of MPI on the systems <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-mpi-benchmarks.html>
- I used SPEChpc2021 as an application based benchmark to support my analysis <https://www.spec.org/hpc2021/>
- I used the Tuning and Analysis Utilities(TAU) profiler to performance analysis <https://www.cs.uoregon.edu/research/tau/home.php>
- Cray Performance Tool uwas used to gather profiling data on applications <https://www.spec.org/hpc2021/>

Notation and Acronyms

An optional section

Any well written document will introduce notation and acronyms before their use, *even if* they are standard in some way: this ensures any reader can understand the resulting self-contained content.

Said introduction can exist within the dissertation itself, wherever that is appropriate. For an acronym, this is typically achieved at the first point of use via “Advanced Encryption Standard (AES)” or similar, noting the capitalisation of relevant letters. However, it can be useful to include an additional, dedicated list at the start of the dissertation; the advantage of doing so is that you cannot mistakenly use an acronym before defining it. A limited example is as follows:

API	:	Application Programming Interface
AWS	:	Amazon Web Services
CISC	:	Complex Instruction Set Computer
CPU	:	Central Processing Unit
DDR	:	Double Data Rate
DIMM	:	Dual In-line Memory Module
GFLOPS	:	Giga Floating-Point Operations Per Second
GW4	:	Great Western 4 Is a consortium of four research intensive universities in South West England and Wales. Consisting of Bath, Bristol, Cardiff and Exeter universities
HPC	:	High Performance Computing
HPG	:	High Performance Group
IaaS	:	Infrastructure as a Service
MIMD	:	Multiple Instructions Multiple Data
MPI	:	Message Passing Interface
NUMA	:	Non-Uniform Memory Access
OMP	:	Open Multi-Processing
OpenAcc	:	Open Accelerators
OpenCL	:	Open Computing Language
PAPI	:	Performance Application Programming Interface
RAM	:	Random Access Memory
RISC	:	Reduced Instruction Set Computer
RS	:	Reference System
SIMD	:	Single Instruction Multiple Data
SPEC	:	Standard Performance Evaluation Corporation
SUT	:	System Under Test
TAU	:	Tuning and Analysis Utilities
UMA	:	Uniform Memory Access

Chapter 1

Introduction

1.1 HPC

High Performance Computing(HPC) has become essential for solving computationally intensive problems across numerous scientific and engineering fields, including climate modelling, particle simulation, molecular simulations, among others. In addition, the explosion in deep learning requires substantial computation power, memory, and storage capacity due to the requirements of the large models and dataset needed for optimal performance. All of these demands have led to rapid development within the HPC industry, causing substantial change and growth within the field, further pushing the boundaries of computing.

Traditionally, the industry relied upon Moore’s Law, which is the observation that the number of transistors roughly doubles approximately every two years, resulting in increased performance on a regular cadence [20], for performance improvements. However, with the increased challenges with lithograph and fabrication of modern processors[10], these regular increases in performance can no longer be relied upon[32]. Therefore, the industry has begun exploring other areas, including new hardware designs and software optimisations, to try and continue performance improvements. This has led to modern HPC systems to be built with multiple accelerators, new processor architectures, and novel programming models to support this hardware.

1.2 ARM

One of the new processor architectures being used within modern HPC systems is Arm. The swift development of Arm processors was enabled by Arm’s licensing model of its architecture, which allows vendors to use Arm’s architecture to build their own designs. This led to Arm being used heavily within the fast growing mobile space, with multiple vendors producing Arm processors for a range of devices. The greater choice of processor manufacturer, led to intense competition between the vendors, reducing cost and providing innovative designs. Consequently, the architecture and the supporting software ecosystem underwent rapid development. As the designs and software ecosystems became more mature, and the processors more powerful, vendors began testing Arm designs for server space, and by extension, the HPC space.

In 2018, Marvell/Cavium was among these vendors, releasing their ThunderX2 Arm processor. Marvell focused on optimising the processor for HPC workloads, which included each processor having access up to eight DDR4 memory channels[17]. This allowed a dual socket system to achieve class leading memory bandwidth in excess of 250GB/s[17], positioning the ThunderX2 arguably the first HPC optimised Arm processor.

The development of ThunderX2 led to the creation of the Isambard project, aiming to produce the world’s first production Arm-based HPC supercomputer, with the system being designated as a Tier 2 supercomputer within the UK national integrated HPC ecosystem ¹. The project was a collaboration between the Great Western 4 (GW4) ² alliance along with the UK’s Met Office, Cray, Arm, and Cavium, with funding from EPSRC. Isambard is a Cray XC50 ”Scout” system that was delivered in 2018, initially

¹<https://www.hpc-uk.ac.uk/docs/>

²<https://gw4.ac.uk/>

consisting of 160 nodes, and was expanded to 329 nodes in 2020[1]. Each node contains a dual 32 core variant of the ThunderX2, the CN9980.

1.3 Cloud Computing

However, traditional HPC resources are provided through dedicated supercomputers or clusters of high-end servers. This requires significant capital investment, operational expertise, and long lead times before becoming operational. Furthermore, once built, the ongoing maintenance from specialists incurring running costs including power, cooling and hardware replacement, regardless whether the system is utilised or not. These significant costs require operators of these supercomputers, such as universities, research institutes, or government agencies, to pool funding together to cover the cost. For example the cost for the Isambard project totalled £6.5 million[1]. These costs make it essential for the operators to estimate the capacity requirements of the system accurately, so current and future demands of the users are met, without excess or limited capacity. In addition, due to the rapid development of computer hardware, additional investments are often required if the operators want to keep the system up to date, further adding to the already high running costs. Some of the reasons operators may want to upgrade the system include:

1. **Increase performance:** New processors offer increased performance due to a combination of factors, including improved die manufacturing that results in a die shrink, increasing transistor per area count, increased cache size and prefetching, improved branch prediction, and many others. All of these improvements result in higher Instruction Per Clock(IPC), leading often to reduced run time for applications, and increased compute density³.
2. **Increase efficiency:** The factors from increased performance which lead to increased IPC, also lead to improvements in performance per watt. Performance per watt is a measure of the rate computation per watt, with Floating Point Operations Per Second(FLOPS) being a common measurement. Due to the energy cost running processing, as well the energy cost used for cooling, which often outweigh the cost of the processors itself, improved performance per watt results in reductions in operating costs.
3. **Solve larger problems:** Increase in performance, as well memory and storage capacities, thus making it possible to solve problems that were previously considered impossible or impractical, furthering the possibilities of research.
4. **Hardware lifespan:** Most hardware vendors only offer limited support for their products, both for updates as well as replacement for faulty hardware. This becomes even more problematic as the probability of hardware failure increases with the age of hardware, thus resulting in set lifetimes for many computers, including supercomputers.

Cloud computing companies offer an alternative model to access HPC resources to institutions and users of HPC workload. Using the large customer base, cloud companies spread the total cost of the system, encompassing the upfront, running and upgrade costs, across all customers through a pay for what you use model. This model charges users of their service a fee, often at an hourly rate, that is significantly less than total cost of the system. As a result, users, especially institutions operating traditional supercomputers, can access HPC resources at a fraction of the cost. The lower cost further increases the customer base, enabling users who previously were unable to afford or access a supercomputer, to use HPC resources at a more affordable cost. Moreover, unlike traditional supercomputers which allow resources to a limited number of users, cloud computing can leverage the economies of scale, where the increasing number of customers decreases the overall cost per customer, until the hourly rate is the operating cost. Furthermore, cloud offers flexibility in regards to capacity requirements, enabling users to scale to meet their compute demands, rather than forcing a fixed capacity with the associated cost with unused compute, as is often the case with traditional supercomputers. Additionally, the hourly rate includes the cost of future upgrades to the system, leading to frequent releases of newer hardware variants of the services. These updated services take advantage of the benefits from updated hardware, including increased performance density and performance per watt, thus often allowing cloud companies to offer newer hardware, at lower prices.

³Another operating cost of supercomputers is rent, increasing compute per metre, reducing the amount of area needed for the same compute, reduces costs

Amazon Web Services (AWS) is utilising the potential benefits provided by cloud HPC and the Arm Architecture by offering an Arm based Cloud HPC option in 2018[6]. The latest version of this option uses their own Arm processor called Graviton3, which was released in 2021[2]. The processor is a 64-bit modified Arm Neoverse-V1 core, with 64 cores and uses DDR5 memory[5]. As with many cloud options, there are several versions available ranging from 1 core with 2GiB of memory to the full 64 cores with 128 GiB of Memory costing \$2.3123 an hour[4]. Compared to Isambard’s upfront cost, you could get 449 days³ of non-stop compute on Graviton 3 for the same price. When considering the operating cost of Isambard, and the true utilisation instead of fixed scale costs, it would take many more days before Isambard would match the cost. Therefore, AWS’ cloud HPC service using Graviton 3 provides an interesting alternative to Isambard, both in performance and pricing model.

1.4 Benchmarking

The advantages of cloud computing, particularly the on-demand model, are obviously desirable to users, however this is in the context that the performance cloud HPC offers meets the demands of traditional supercomputers users. Therefore, evaluating and comparing the performance of both AWS’ Graviton3 and Isambard’s ThunderX2, will give an insight into the viability of cloud HPC.

To evaluate these systems, a selection of carefully designed benchmarks will be used. Benchmarks are an essential tool for measuring and evaluating the performance of systems. Utilising both synthetic and application benchmarks, including SPEChpc 2021, will enable the comparison between the two systems. Furthermore, benchmarks also aid in the demonstration of performance improvements, analysis of software ecosystems, by informing purchasing decisions and enabling regression testing, among other benefits.

Performance Analysis of Software: As the number and complexity of architectures increase, many of which have their own software ecosystems, benchmarks provide a way of evaluating the maturity and performance of these software stacks. These software stacks contain a wide range of items from compilers, to programming model API, communication interface, and others. Moreover, there are often several different options of software at each layer of the stack, increasing the possibility of incompatibility or bugs between software in the stack, as well as sub-optimal setup of the stack. Benchmarks suites like SPEChpc2021, that contain a range of applications that can test multiple layers of the stack, can allow operators and developers to identify any issues within the stack, as well as selecting the optimal stack for users.

Informing Purchasing Decisions: Selecting the right HPC system for users is crucial due to the high procurement costs involved. Benchmarks aid operators as they can look at a range of results across multiple systems and software setups. This not only allows operators to select the best hardware for the needs of the user, but also allows them to define the thresholds of the system to the users and funding organisations. However, hardware manufacturers know the value of these benchmarks and can tune their system to excel in synthetic benchmarks that may not represent real world application performance. Therefore, it is advantageous to use multiple benchmarks based on real world application, that test various aspects of a system to give a more accurate characteristic of system performance.

Regression Testing: Systems require regular maintenance, which includes software and security updates, as well as hardware maintenance. During these tasks, inadvertent performance regressions can occur. Using comprehensive benchmarks, including SPEChpc 2021, after maintenance of any kind, can reveal these regression. This, in combination with stored good results, can be used to automatically compare and flag any issues. Furthermore, running tests periodically, not just after maintenance, can reveal any additional issues, such as non-performing nodes. This technique was used with the SPEChpc 2021 benchmark to detect non-performant nodes in supercomputers at TU Dresden and RWTH Aachen [7].

³329 nodes * \$2.3123 = \$760.75
Using £1 = \$1.26
Isambard Cost £6,500,000 == \$8,199,717.50
\$8,199,717.50 / 760.75 = 10778.5 hours

1.5 Aims and objectives

The overall aim of this project is to provide a thorough analysis between AWS' Graviton3 and Isambard's ThunderX2 by utilising multiple benchmarks including SPEChpc 2021 and other tools to produce a complete comparison between the two systems. In addition, this project aims to contribute to the understanding and viability of cloud based HPC systems and their potential as an alternative to traditional supercomputers. This necessitates the following aims and objectives which are detailed below:

1. Use a suite of benchmarks to collect data about the characteristic of each system
2. Evaluate the scaling performance of Graviton3 compared to Isambard
3. Identify any bottlenecks or performance limitations with Graviton3
4. Determine whether Graviton3 is a viable alternative to Isambard

Chapter 2

Background

This section provides a technical background to the technologies and concepts used in this project. This chapter introduces the required concepts from computer architecture, to HPC system architecture and the corresponding programming models. In addition, it will introduce the benchmarks and tools used to evaluate the systems, as well as information on the systems being evaluated.

2.1 High-Performance Computing

This section provides an introduction to the field of HPC, with the main focus on the technical aspects of a modern supercomputer or clusters. It is divided into two subsections, system architecture, and programming models. System architecture provides a high-level overview of the designs and challenges faced by supercomputers and clusters, while programming models offer an explanation of how the hardware is programmed. This is covered to give a background on HPC systems, and the various aspects that need to be considered when evaluating the performance of them.

2.1.1 System Architecture

Modern HPC systems and supercomputers are a mixture of different designs, architectures, and hardware, with certain designs and architectures being more common than others. Specifically, these tend to be inherently parallel, heterogeneous, cluster systems with multi-core processors, often with multiple sockets per node, and with optional accelerators.

Node

The building block of a cluster is known as a “node”, which is essentially a single, or multi socket computer. These sockets in modern nodes will be multi-core processors, with some nodes containing accelerators, such as a Graphic Processing Unit (GPU). For multi-core processors, the cores share data via an on-chip interconnect, accessing data through a hierarchy of cache¹, then to main-memory. Data can be duplicated across all layers of cache, from per core cache to shared cache, as well as in main memory. Thus, a data write can cause a potential data race issue, where parts of the caches and memory are not consistent. Therefore, data writes are most often propagated to all layers, which is called a cache coherence protocol.

NUMA

In multi socket systems, shared main memory is normally divided between processors, with each processor having direct access to its own region of memory, and having to go through a distributed shared memory connection, which increases the time to access memory, for other regions. This is called Non-Uniform Access Memory (NUMA), opposed to the traditional Uniform Access Memory (UMA), where each processor shares memory uniformly. Processors in shared memory architecture usually follow the Multiple-Instruction-Multiple Data(MIMD) principle [9]. However, UMA suffers from bus arbitration, which is when both processors try to access the shared memory at the same time, and due to only one

¹often denoted by the capital L, which stands for level, with higher numbers being slower

processor being able to access memory at a time, the other processors have to wait, thus increasing memory access time. Despite the access time penalty for memory outside of a processor's NUMA region, the issues with bus arbitration often result in NUMA being the better solution for HPC workloads. This is because the data in a lot of HPC workloads can be split between processors, with minimum cross memory access, reducing the overall penalty of NUMA.

Accelerators

In addition, modern supercomputers have seen the rise in the use of hardware accelerators. These include Field Programmable Gate Array(FPGA), Application Specific Integrated Circuits, and General-Purpose computing on Graphic Processors(GPGPU), often referred to as Graphic Processing Unit (GPU). Which are often designed (especially with GPUs) to work using Single Instruction, Multiple Data (SIMD)[9], which is when the same instruction is applied on multiple data, at the same time. The preferred computational problem for accelerators is independent and embarrassingly parallel vector operations, which include stencil operations or convolution neural networks. This use case results in a relatively high core count compared to CPU, with core counts often in the range of hundreds to thousands of cores, with Nvidia's latest GPU having 18432 CUDA cores[21]. However, these cores are not comparable to CPU cores, as they are less versatile, and mainly deal with data streams. Furthermore, most accelerators contain their own pool of memory. This is because transferring data over the bus from the host processor's memory pool has a significant overhead. This overhead is from both the physical distance from the host processor's memory, and because of bus and memory bandwidth limitations. Moreover, some data transfers from the host processor's memory are transferred through the host processor, instead of directly to the accelerator, adding further overhead.

Scalability Issues

However, shared memory designs have inherent boundaries when it comes to scalability. In what is often known as vertical versus horizontal scaling, vertical scaling adds more compute and memory to a single system to increase performance, but has fundamental limitations. These include limits on increasing the processor die size before cost and manufacturing limitation prevent further scaling, physical limits on how much heat can be dissipated while increasing power, the signal delays within the processor, just to name a few. Thus, to increase performance once vertical scaling is unviable, the best solution is to scale horizontally, which is adding more systems together to increase performance, referred to as a cluster. This approach is used in modern HPC systems, and is achieved via connecting multiple systems, or nodes, together via an interconnect. Moreover, because each node has its own memory, this is referred to as a distributed memory system. Modern HPC systems are also becoming more heterogeneous, which is the mixture of nodes with different capabilities, which can include different processors, memory and optional accelerators.

2.2 Programming Models

As systems scale beyond a single core and include multiple cores, multiple sockets, and multiple nodes in a cluster, along with accelerators, writing applications that can effectively take advantage of the hardware while remaining portable across architectures has become increasingly challenging. As a result, multiple programming model standards have been developed.

2.2.1 OpenMP

One of the most common models for multiprocessing is Open Multi-Processing(OpenMP). OpenMP is a multi-platform Application Programming Interface (API) for multiprocessing on shared memory architectures written in C,C++ and Fortran[8]. By providing a set of compiler directives, library functions and environment variables, it enables developers to write parallel code with minimal changes to existing code. The parallel model used in OpenMP is the Fork-Join model. All programs start with a single main, or master thread, which executes the program serially until reaching a parallel region. Upon reaching a parallel region, the main thread creates a work group of threads, including itself, and divides the parallel region task among the group. When the task is completed, the join part of the process starts. As each worker thread will complete the task at different rates, there is an implicit barrier at the end of the parallel region, suspending the threads. Once all worker threads complete the task, only the main

thread resumes execution. This is only a high level overview, with OpenMP offering multiple options to customise this process, from changing the number of workers, to the splitting of the tasks, where to bind threads, and many more. Furthermore, OpenMP offers atomic variables, barriers and many other synchronisation clauses to aid in the creation of multi-threaded applications.

However, when programming for systems with multiple sockets, as mentioned at section 2.1.1, NUMA can become an issue. The simplest solution is by taking advantage of the first touch principle, which is the standard method most operating systems use to handle initialising data to memory regions for NUMA. This can be done in OpenMP by binding the threads to the sockets, and splitting the data between threads the same way, thus data is always located in the threads NUMA region.

OpenMP is a portable, scalable interface which allows developers to create applications that can run on single CPU systems to multiple sockets. However, due to OpenMP being a shared memory model, to scale past a single node, it must be used in combination with a standard to communicate to other nodes. The Message Passing Interface (MPI) is the de-facto standard for achieving this, with this combination being referred to as hybrid. This allows for scaling an application across a whole supercomputer.

2.2.2 Message Passing Interface

MPI is a dominant standard for portable communication between distributed memory systems and is written in C, C++ and Fortran[11]. Although, the standard did officially add support for shared memory programming in MPI-3, which is useful in NUMA systems when taking advantage of memory locality. The MPI standard defines a set of functions for the sending and receiving of messages between processes, as well as the synchronisation of execution of the various processes. Each process runs the same base code, with access to its own memory space, known as the Single Program, Multiple Data (SPMD) model[9].

MPI is designed to be highly scalable, allowing programs to scale efficiently on systems ranging from a single processor, to a cluster with thousands of nodes. As MPI is an open standard, there are multiple API implementations of it, each with slight variations in an attempt to improve performance, with vendors providing system specification optimisations.

In MPI, processes are split into groups, with each process being identified by its rank, which is an enumerated number from 0 to the size of the group. To handle this, objects called communicators connect groups of processes and handle communication between them. By default, all processes are within one group, which can be changed by `MP_COMM_WORLD`. The majority of MPI communication functions can be broadly classified into two categories, namely point-to-point and collective operations.

Firstly, point-to-point operations are functions with communication directly between specified processes, which include send and receive. Point-to-point operations are useful in scenarios with patterned or irregular communication, such as halo exchange. With there being blocking and non blocking functions, as well as ready send functions, where the data is sent when the receiver is ready.

Secondly, collective operations are functions with communication either between a programmed subset of ranks, or the whole pool. A good example is broadcast, which sends data from one process, to all other processes in the group. Another example is gather, which a process receives data from all the processes in the group. Finally, a third example is all-to-all, which is a combination of the gather and scatter functions, where n items of data are rearranged so that each n th process gets the n th item of data.

2.3 Benchmarks

In this section, an overview of benchmarking is given, along with a description of the benchmarks used within this project, and a detailed background of one of the benchmarks, SPEChpc2021. This is covered to give an understanding of the main method used in this project to analyse the two systems, as well as the significance the role benchmarks play within the field HPC.

Benchmarks are an essential method for measuring and evaluating the performance of systems. By using a range of benchmarks, the performance of the system can be characterised. There are multiple types of benchmarks, however the focus in this project will be on synthetic and application based benchmarks. Synthetic benchmarks are artificial programs designed often to test a specific aspect of a system performance, encapsulating the important performance characteristics, for example adding two 32 bit integers together, memory access, among others, without the complexity of real world applications. On the other hand, application based benchmarks are based on parts or whole real-world applications, which measure the performance of a system in a more representative workload.

While synthetic benchmarks may not reflect real-world performance, they can still be valuable when used alongside application-based benchmarks. By using application-based benchmarks to measure the

performance of the entire system, and then performing a more detailed analysis of the system using synthetic benchmarks, a more comprehensive characterization of the system can be obtained. Good Benchmark Principles Creating high-quality benchmarks can be challenging, with many pitfalls that can cause the results of the benchmark to be less valuable, to being completely unreliable as a measurement tool. An example of a pitfall is not checking the correctness of the result of the benchmark, which would allow a run of zero computation to be valid. To try and avoid these issues themselves, SPEC created the following list of principles to follow when creating benchmark[24]:

- **Specifies a workload** - A strictly-defined set of operations to be performed.
- **Produces at least one metric** - A numeric representation of performance. Common metrics include:
 - **Time** - For example, seconds to complete the workload.
 - **Throughput** - Work completed per unit of time, for example, jobs per hour.
- **Is reproducible** - If repeated, will report similar metrics.
- **Is portable** - Can be run on a variety of interesting systems.
- **Is comparable** - If the metric is reported for multiple systems, the values are meaningful and useful.
- **Checks for correct operation** - Verify that meaningful output is generated and that the work is actually done.
- **Has run rules** - A clear definition of required and forbidden hardware, software, optimization, tuning, and procedures.

Following these principles will ensure that the benchmark will provide a set level effectiveness in the workload it is measuring, but these principles do not ensure that the workload being benchmark is useful.

2.3.1 Intel MPI Benchmarks

Intel's MPI benchmarks is a suite of synthetic benchmarks used to characterise a system MPI performance[14]. MPI is the most common standard for communicating between nodes, therefore, the ability to measure a system MPI performance is important.

The suite of synthetic benchmarks are designed to test various point-to-point and global communication operations, across a range of message sizes and ranks. These benchmarks allow for the following characterising of the system:

- **Performance of the System** - The maximum achievable performance of each MPI function.
- **Node Performance** - The maximum individual node performance, can identify slow nodes
- **Network Latency** - The latency between nodes during communication
- **Throughput** - The maximum throughput of the system
- **Efficiency of the MPI Implementation** - Whether the MPI implementation is performing as expected, useful when comparing MPI implementation and configurations.

2.3.2 STREAM Benchmark

STREAM is another useful synthetic benchmark. STREAM is the de-facto standard for measuring the achievable sustained memory bandwidth of a CPU in MB/s[18]. The benchmark consists of four kernels, three simple element-wise arithmetic operations, and one simple copy operation on vectors, with the kernels being:

- **Copy** - The transfer rate without any arithmetic : $a[i] = b[i]$
- **Scale** - Multiplication operation : $a[i] = constant * b[i]$
- **Sum** - Adding two vectors together : $a[i] = b[i] + c[i]$
- **Triad** - The combination of Sum and Scale : $a[i] = b[i] + constant * c[i]$

The standard rule for running STREAM is that each array is at least four times the size of the last level of cache, or a million elements, whichever is largest[18]. To calculate the achieved memory bandwidth, it is modelled as three times the length of the arrays, divided by the fastest run time for that kernel.

2.3.3 SPEChpc 2021

The Standard Performance Evaluation Corporation (SPEC) developed SPEChpc2021 as a suite of benchmarks based on real-world applications[31]. It consists of 9 applications and mini-applications, covering a wide range of scientific domains, programmed in multiple languages and programming models, with 4 different problem sizes. Moreover, the benchmark suite can scale across thousands of nodes, allowing for a comprehensive evaluation of modern HPC systems using real-world problem sets.

About SPEC

The Standard Performance Evaluation Corporation (SPEC), is a non-profit organisation founded in 1988[29]. SPEC members include hardware and software vendors, universities and researchers[29]. Its goal is to "establish, maintain and endorse standardised benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems"[25]. SPEC also reviews and publishes submitted results from the member organisations and other benchmark licensees [30]. SPEC High Performance Group (HPG) is responsible for designing and maintaining benchmarks for High Performance Computing (HPC)[27]. The group has released a series of benchmarks with their first being in 1996[27]. With them currently maintaining four benchmark suites:

- **SPEC OMP 2012** - Designed for measuring intra-node CPU parallelism performance using applications based on the OpenMP 3.1 [27];
- **SPEC MPI2007** - Designed for measuring inter-node communication performance of applications using the Message-Passing Interface (MPI)[27];
- **SPEC ACCEL** - Designed for measuring the performance of hardware accelerator of applications using the OpenCL, OpenACC, and OpenMP standards[27];
- **SPEChpc 2021** - Consists of 9 full and mini-applications across multiple languages including C/C++/Fortran. They are designed for measuring multiple programming models, including MPI, MPI+OpenACC, MPI+OpenMP, and MPI+OpenMP with target offload. For Both CPU-only and heterogeneous HPC systems. [27].

Design of SPEChpc2021

The development and creation of SPEChpc arose from the challenges associated with benchmarking modern HPC systems. As HPC architectures continue to evolve at a fast pace, becoming increasingly heterogeneous, incorporating numerous types of accelerators, and evaluating and characterising their real-world performance becomes more challenging. This difficulty is further compounded with the use of new parallel and accelerator programming models to support the hardware. Therefore, it would be highly advantageous to have a suite of portable application-based benchmarks, which evaluates the characteristics and performance of these systems, along with the maturity of the software ecosystems of different architectures.

SPEC HPG set out to address these challenges in 2017 when they initiated their search for benchmarks to address this problem[28]. They used their benchmark search program, which accepts external benchmarks based on real world applications, to develop their benchmarks[26]. They worked with the HPC community to gather benchmarks with the following characterised to create SPEChpc[24]:

- **Representativeness** - The benchmarks reflect real world applications.
- **Multiple Parallel Models** - Support for MPI only, MPI+OpenACC, MPI+OpenMP, MPI+OpenMP with target offload.
- **Multiple Programming Languages** - Consists of benchmarks programmed in Fortran, C and C++.
- **Predictable Code Paths** - Algorithms behave the same across different platforms.
- **Limited I/O** - Benchmark's should not be limited by I/O.
- **Strong Scaling** - Should scale within and across nodes, across multiple suite sizes.
- **Portability** - Ability to run on a variety of CPU architectures, including x86, Arm and Power ISA.
- **Valid Output** - Check for correctness of the result within a margin of error.

Benchmark Composition

Nine HPC applications and mini-applications were selected from a pool of solicited applications through the search program. SPEChpc 2021 was released on 28/10/2021[31] (version 1.0.3) which consisted of a suite of 9 benchmarks, that cover a range of application areas and multiple programming languages, which is summarised in Table 2.1. There are 4 different benchmark sizes, tiny, small, medium and large. However, not all benchmarks are included in all the sizes, namely medium and large. This is due to either the application’s failing to scale to the suite size, or the application’s realistically largest problem is too small for larger suite sizes.

The requirements for these benchmarks are listed in Table 2.2. The requirements for the benchmarks are defined to reflect typical HPC system size[7], with the MPI rank count range being defined to ensure strong scaling for the benchmarks. Although however, the requirements are given with the caveat that requirements may change depending on the individual system overheads[22]. Additionally, memory requirements change with high MPI rank count due to communication buffer sizes.

SPEC Harness

SPEChpc2021, like other SPEC benchmarks, contains the SPEC harness, which plays a critical role in all stages of running the benchmarks. The harness is responsible for various tasks from installing the benchmark, protecting the source code from tampering via a checksum, to handling benchmark execution, validates the output correctness and produces an output report. The output produced by the harness contains all the information needed to reproduce the results of benchmark, including run flags, compiler flags and environment flags.

Metrics

SPEC Score: SPEChpc2021 reports the SPEC score along with the execution time of each benchmark, which is based on execution time to solution, not the whole application run time. A ratio of the run time of the reference system, to the System Under Test (SUT), for each benchmark is calculated, with the SPEC score being the geometric mean of all the benchmark ratios[24]. So therefore, a higher SPEC score indicated a faster run time.

Reference System (RS) The reference system is from TU Dresden’s Taurus System using the Haswell CPU Islands. Each node consists of 2, 12 core Intel Xeon E5-2680 v3 CPUs running at 2.5GHz, with 64GB of DDR4 running at 2133MHz over a 56Gb/s Mellanox InfiniBand FDR interconnect[24]. Using only the MPI version of the benchmark, the tiny size reference time used 24 ranks on a single node, small used 10 nodes (totaling 240 ranks), medium used 85 nodes (for 2040 ranks), and large used 340 nodes (for a total of 8160 ranks). [24]

Base and Peak Two options are available when running benchmarks, the base score and the optional peak score. The base score metric requires that all benchmarks use the same compiler flags, in the same order, and the same node-level parallel model, with the same number of ranks and the same number of host threads per rank. In contrast, the optional peak score allows for greater flexibility by allowing for more options to optimise for each individual benchmark. This includes allowing different compiler options, node-level parallel models, and the numbers of ranks and threads per host rank. Limited source code modification is also allowed to tune directive models. As a result, the base metric provides a more basic level of tuned performance, while the peak metric provides a heavily tuned level of performance. [24]

Table 2.1: SPEChpc 2021 benchmarks application properties

Name	Application Area	Suite	Language	Approx. #LOC	#MPI Calls	# OMP dir.
LBM D2Q37 (x05)	Computational Fluid Dynamics	T/S/M/L	C	9000	118	50
SOMA (x13)	Physics / Polymeric Systems	T/S	C	9500	90	192
Tealeaf (x18)	Physics / High Energy Physics	T/S/M/L	C	5400	22	86
Cloverleaf (x19)	Physics / High Energy Physics	T/S/M/L	Fortran	12500	23	827
Minisweep (x21)	Nuclear Engineering - Radiation Transport	T/S	C	17500	41	39
POT3D (x28)	Solar Physics	T/S/M/L	Fortran	495000 (incl. HDF5)	88	124
SPH-EXA (x32)	Astrophysics and Cosmology	T/S	C++	3400	82	36
HPGMG-FV (x34)	Cosmology, Astrophysics, Combustion	T/S/M/L	C	16700	53	206
miniWeather (x35)	Weather	T/S/M/L	Fortran	1100	11	36

Table 2.2: SPEChpc 2021 System requirements

Suite	Number of Ranks	Memory
Tiny (t) (5xx)	1-256	60 GB
Small (s) (6xx)	64-1024	480 GB
Medium (m) (7xx)	256-4096	4 TB
Large (l) (8xx)	2024-32768	14.5 TB

2.4 Program Profiling

Program profiling is an extremely useful dynamic analysis tool used to analyse programs and aid in improving performance of applications. Profiling works by using a tool called a profiler on either the source code or the program’s executable. A profiler uses multiple techniques to collect data, including operating system hooks, performance counters, hardware interrupts and code instrumentation. This data allows the profiler to measure multiple characteristics of the program, including the memory usage of the program, the frequency and duration of function calls, to usage of certain instructions, and cache utilisation, among other data.

Although in this project’s context, profiling will not be used to optimise any applications, the analysis provided on the benchmark by profiling will provide useful insight into the systems as a whole, as well as the software ecosystems.

2.5 Overview Of Systems

This section covers an overview of both systems used in the comparison for this project. The overview will contain a small background on the system, as well as a detailed description of the specification of the processors, ending with software used on the systems. A summary of the technical specification is shown in Figure 2.3.

2.5.1 Isambard ThunderX2

The systems that will be used as the reference system in this investigation is the Marvell’s ThunderX2 hosted in GW4 Isambard supercomputer. GW4 is a consortium of four research intensive universities in South West England and Wales, Bath, Bristol, Cardiff and Exeter universities. The GW4 Alliance², together with the UK’s Met Office, Cray, Arm, and Cavium, with funding from EPSRC, delivered Isambard for £3 million in 2018[19]. Isambard consists of multiple system architectures, but the focus will be on the Cray XC50 ‘Scout’ with Marvell’s ThunderX2 CPUs. The system is the world’s first production Arm-based supercomputer, with it initially consisting of 168 nodes, which was then expanded to 329 nodes in 2020[1], resulting in a total of 21056 high performance, Armv8 cores. The XC50 Scout system consists of 42 blades in a cabinet, with each blade containing four, dual socket nodes, connected via a Cray Aries interconnect in a dragonfly topology operating at 14Gbps [33].

Marvell’s ThunderX2 is a 64 bit ARM processor based on the Armv8-A architecture, and is their second generation of Arm-based server Processors targeted for HPC [17]. The model of ThunderX2 used in Isambard is the 32 core CN9980 and is permanently set to turbo at 2.5Ghz. Each processor can execute 1, 2 and 4 way hardware multi-threading, however for benchmarking, 1 way multithreading was used. Each processor has access to 8 DDR4 channels, of which 160 of the nodes contain 256GB of memory, with the other 169 nodes containing 512GB of memory. They are arranged in 32GB and 64GB per DIMMs respectively, with them both operating at 2666MT/s. Thus yielding a STREAM triad result of over 250 GB/s per node [19]. Each core has 32kB of level 1 data and instruction caches, and 256kB 8-way set associative level 2 cache. There is 1MB of distributed level 3 cache per core, resulting in a total of 32MB of cache. Shared access to level 3 cache is provided across an inter-core ring network-on-chip[12]. Furthermore, the processor supports 128-bit vector Arm Advanced SIMD instruction, which is

²<https://gw4.ac.uk/>

Table 2.3: Overview of System Specifications

Processor	ThunderX2	Graviton3
MicroArch	Vulcan	Neoverse-V1
Frequency	2500MHz	2600MHz
ISA	ARMv8.1	ARMv8.4-a
SIMD	128-bit SIMD “NEON”	4x Neon 128bit Vectors/ x2SVE 256bit
Cores	2x32	64
L1 (Per Core)	32KB Inst/Data	64KB Inst/Data
L2 (Per Core)	256KB	1MB
L3 (Shared)	32MB	32MB
Memory	8x DDR4	8x DDR5
Memory Per Node	256GB/512GB	128GB
Peak Memory Bandwidth	320 GB/s	Over 300GB/s

also referred to as “NEON”.

At the moment of data collection, Isambard was running the Cray Linux Environment (version 7.5), Suse SLES 15. Along with Cray Programming Environment (CPE), most importantly the Cray compiler (Cray clang and Fortran version 11.0.4), cray mpich (version 7.7.17), Cray netcdf(version 4.7.4.4). Also, the scheduling system used was the Portable Batch System (PBS) (version 19.2.4). For compiler flags, -mcpu=thunderx2t99 and -mtune=thunderx2t99 were used, as recommended by Marvell and Arm [16]

2.5.2 Amazon Graviton 3

The cloud system used to investigate the viability of cloud HPC is Amazon’s latest c7g instance, which uses an AWS Graviton3 processor (Graviton3 will be used as an analogous to c7g for the rest of the paper). The cluster used for the evaluation consists of 8 Graviton3 nodes, using the “c7g.metal” instance size. The “c7g.metal” consists of the full 64 cores of Graviton3, 128GiB of DDR5 memory, 30Gbps of network bandwidth, and costs \$2.3123 an hour [4][3].

Graviton3 is Amazon’s in-house 64 bit Arm processor based on a modified Arm Neoverse-V1 core using the ARMv8.4-a architecture [5]. Graviton3 is a single socket system, thus has only one NUMA region compared to Isambard’s two regions. Each core has access to 64KB of L1 data and instruction cache, 1MB of L2 cache and 32MB of shared L3 Cache. Furthermore, the processor supports 4x NEON 128bit vectors and 2x SVE 256 bit SIMD instructions. Graviton3 also provides high memory bandwidth with a total of eight memory channels with a peak theoretical memory bandwidth of over 300GB/s[23] This is achieved by Graviton3 using the latest DDR5 memory, which has some significant changes over DDR4 which is what ThunderX2 uses. The first difference is the reduction in channel width to 32 bits for data, down from 64 bits for DDR4. However, each DIMM has 2 memory channels, resulting in the same total width per channel. By having two, smaller independent channels, the efficiency of memory access is improved, which is on top of the benefit that increased memory speed provides.

At the moment of data collection, Graviton was running on version 5.10 of the Linux kernel. The MPI implementation used was MPICH (version 4.0.2), with the gcc (11.3.0) used as compiler, along with libfabric(version 1.16.1). The scheduling system was slurm(version 22.05.8). For compiler flags, was -mcpu=neoverse-512tvb as recommended by Amazon [5].

Chapter 3

Results

This chapter focuses on presenting the results of the conducted benchmarks and evaluating the data produce. The evaluation of the result will allow for the systems to be compared, and the objectives listed in 1.5 to be fulfilled.

3.1 STREAM

High memory bandwidth is one of the most important characteristics for a HPC system to have, given that a majority of HPC applications or kernels are predominantly bound by memory bandwidth. ThunderX2 and Graviton3 both have extremely competitive theoretical peak memory bandwidth compared to other vendors, such as Intel’s Broadwell and Skylake processors, which have 154GB/s[19] and 256GB/s[19] peak theoretical memory bandwidth respectively[19]. ThunderX2 is able to utilise up to eight memory channels of DDR4 per socket, compared to six channels for skylake and four channels of DDR3 for Haswell, to produce a theoretical peak of 320GB/s[19]. Graviton3 with four memory channels of DDR5 DIMMs, resulting in a total of eight memory channels, for a claimed peak bandwidth of over 300GB/s[23], however due to the memory speed of the memory not being disclosed, the exact peak bandwidth cannot be calculated¹. Despite this, both ThunderX2 and Graviton3 have attractive peak theoretical memory bandwidth. Although, S.D. Hammond Et. al. (2019)[13] raises an important point of the usefulness of peak memory bandwidth, as recent processor designs have seen a trend where the peak memory bandwidth has surpassed the on-die transport capabilities of the network-on-chip, allowing for improved responsiveness as memory DIMMs can operate at a lower-load level [13].

Theoretical peak memory bandwidth, as the term suggests, is a theoretical value, actual bandwidth depends on multiple factors, such as the memory controller on the processor. The STREAM benchmark measures the sustained memory bandwidth of the processor across four different kernels vectors to calculate bandwidth. The STREAM benchmarks were configured to use arrays of 2^{25} double-precision elements.

Figure 3.2 shows Graviton3 attaining higher memory bandwidth across all four kernels, which is to be expected as Graviton3 has the same number of total memory channels, whilst having newer, higher clocked DDR5 memory. Resulting in the single socket Graviton3 achieving a 1.09x improvement in memory bandwidth over the dual socket ThunderX2, in the important triad kernel.

Graviton3 also achieves the best peak memory efficiency in the triad kernel between the processors, at 88.4%², compared to the 76% of ThunderX2. In a study conducted by S. McIntosh-Smith et al. (2018)[19], it was demonstrated that ThunderX2 on Isambard achieves a peak bandwidth of 253.4GB/s in the triad kernel, which results in a better efficiency of 79.2%. The difference between the ThunderX2 memory bandwidth results will be due to differences in the compiler used, the flags used for running STREAM, and slight hardware variation³.

Figure 3.1 shows the scaling of measured memory bandwidth across all STREAM kernels for Graviton3 normalised to ThunderX2. OpenMP was configured to spread threads across the socket evenly for ThunderX2 to take advantage of the multiple sockets. The first notable observation is the ability for

¹Amazon has not disclosed the memory speed, and also hides the speed in Linux hardware info. However, using amazons claimed peak bandwidth of 300GB/s, the memory speed is at least 4800MH/z

²Using 300GB/s as peak. The bandwidth is stated as over 300GB/s, thus the true efficiency will be less

³In the paper, S. McIntosh-Smith et al. (2018)[19] the benchmark was ran on a pre-production versions of ThunderX2, as well as on Isambard early access nodes, with processors running at a lower clock speed of 2.2GHz.

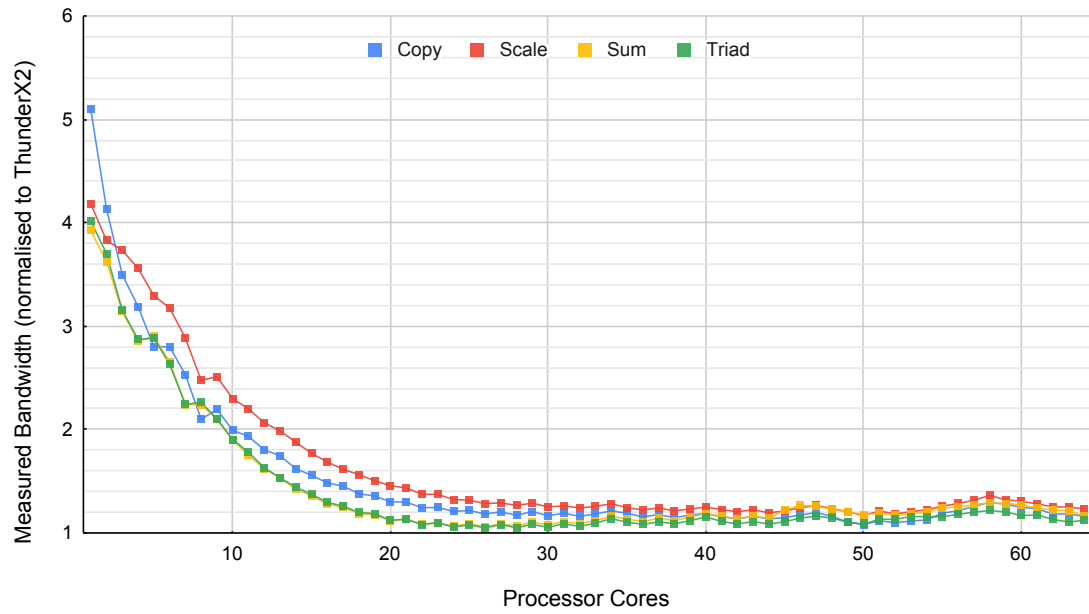


Figure 3.1: STREAM Benchmark showing the relative measured memory bandwidth across all kernels for Graviton3 normalised to ThunderX2

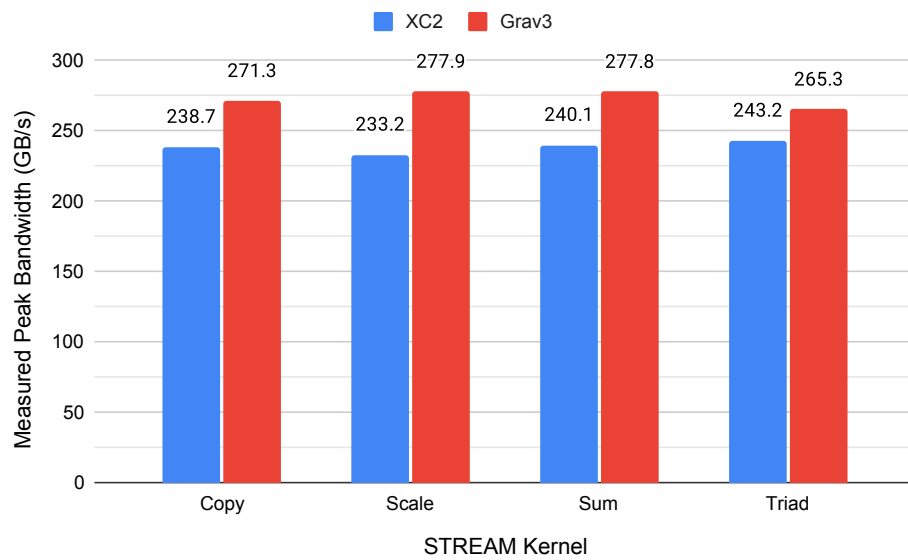


Figure 3.2: STREAM Benchmark showing the peak measured memory bandwidth across all kernels for Graviton3 and ThunderX2 in GB/s

Graviton3 to achieve higher relative memory bandwidth on lower core counts relative to ThunderX2, with increased bandwidth of 3.41x on the triad kernel for a single core. This is primarily due to two factors, the individual faster cores of Graviton3 with a higher clock speed, and the single socket design compared to ThunderX2 dual socket design. The relative memory bandwidth decreases as the core count increases until 16 cores, where it plateaus to the values in Figure 3.2 as the bottleneck becomes the memory bandwidth rather than the core speed, micro-architecture and bus width. Therefore, serial and poor scaling applications or kernels that are memory bandwidth bound could gain a significant performance increase on Graviton3.

3.2 Intel MPI

The Intel MPI benchmark suite was used to Investigate the performance of Graviton3, and compare it to Isambard's ThunderX2. This will enable an insight into any potential issues using Graviton3 for HPC applications. The benchmarks used are Sendrecv, Allreduce, and Barrier, with the off_cache flag set to avoid cache re-use to simulate more realistic throughput results. These MPI functions were chosen to be benchmarked because of their common use and to measure point to pointer and collective MPI operations.

3.2.1 SendRecv

The SendRecv benchmark is based on the point to point function MPI_Sendrecv, a commonly used function within MPI, and is measured in the maximum throughput in MB/s. The benchmark creates a periodic communication chain, where each process sends a message to the right neighbour, and receives a message from the left process, with the message data type being a MPI_BYTE[15].

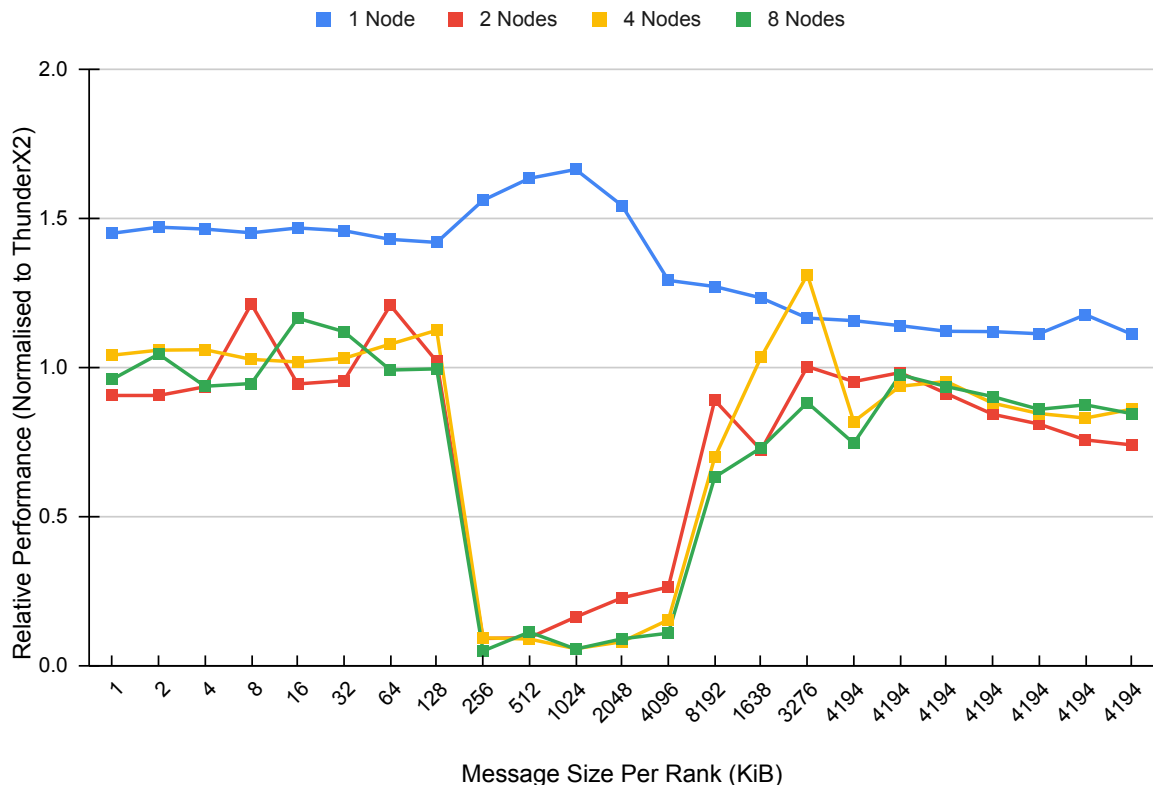


Figure 3.3: Intel MPI Benchmark showing the relative performance of MPI_Sendrecv on Graviton3, normalised to ThunderX2. 64 ranks per node

Figure 3.3 shows the throughput of Graviton3 normalised to ThunderX2. The first interesting observation is the dramatically worse inter-node communication compared to intra-node relative to ThunderX2.

Intra-node communication maintains higher throughput throughout, thanks to the higher core performance of Graviton3. At 256KiB intra-node performance increases to a peak of 1.66x, this is due to the increased L2 cache of Graviton3. Relative performance then decreases as message size increases until plateauing, becoming memory bandwidth bound. However, inter-node communication suffers from a significant throughput drop between 256KiB and 4096KiB, before returning back to near equal throughput with ThunderX2. The drop in relative performance when communicating between nodes suggests Amazon’s interconnect for Graviton3 is a bottleneck for MPI performance of the system, thus reducing the potential performance of Graviton3. Furthermore, the significant drop in performance relative to ThunderX2 between 256KiB and 4096KiB suggests a mis-configuration issue, even as the message falls out of L2 cache for ThunderX2, while still being in L2 cache for Graviton, which should result in an improvement, as shown with intra-node. Moreover, this issue would result in any workload passing message sizes in that range to have significantly degraded performance.

3.2.2 Allreduce

The Allreduce benchmark is based on the collective function MPI_Allreduce, a commonly used function within MPI, and is measured in the average time to complete. The MPI data type is MPI_FLOAT and the MPI operation is MPI_SUM [15].

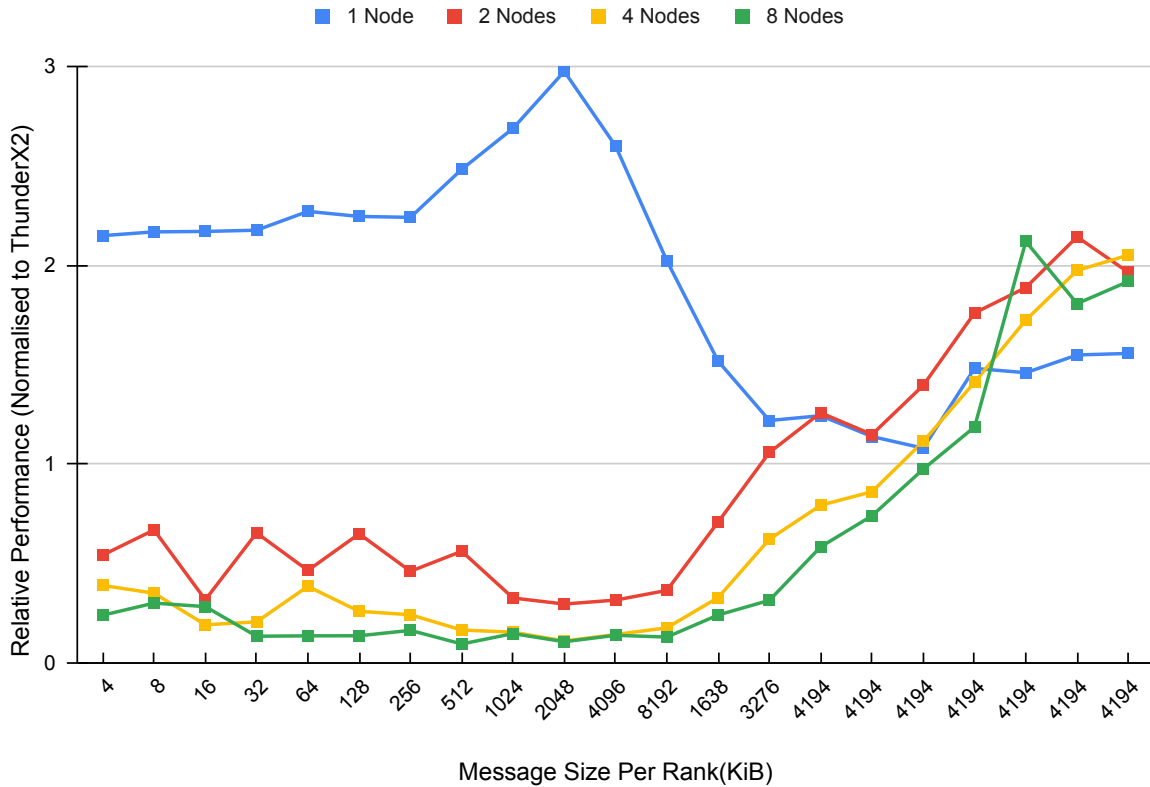


Figure 3.4: Intel MPI Benchmark showing the relative performance of MPI_Allreduce on Graviton3, normalised to ThunderX2. 64 ranks per node

The data presented in Figure 3.4 further supports the existence of substantial communication overhead between nodes. Once again, intra-node communication is significantly higher than ThunderX2, with message size within L1 cache reducing an averaging 2.24x faster than ThunderX2. Relative performance only increases as message size falls out of ThunderX2’s cache at 256KiB, until reaching a peak delta of 2.97x faster to reduce. As the message size falls out of Graviton3’s L2 cache, relative performance decreases, reaching a low of only 1.08X faster than ThunderX2, when outside of Graviton3’s L3 cache. Inter-node communication is significantly slower than ThunderX2, with performance staying plateaued until the message size of 8192KiB due to communication overhead, with zero relative increase in performance when message sizes transition outside of ThunderX2 L1 cache. However, as message size increases, the

primary bottleneck factor shifts from the interconnect to other factors, such as processing speed, cache and memory bandwidth.

3.2.3 Barrier

The Barrier benchmark is based on the MPI_barrier function, which blocks the caller until all processes in the communicator have called it [15]. It is interesting as it is often used for synchronisation between ranks, and can be used to detect latency between ranks.

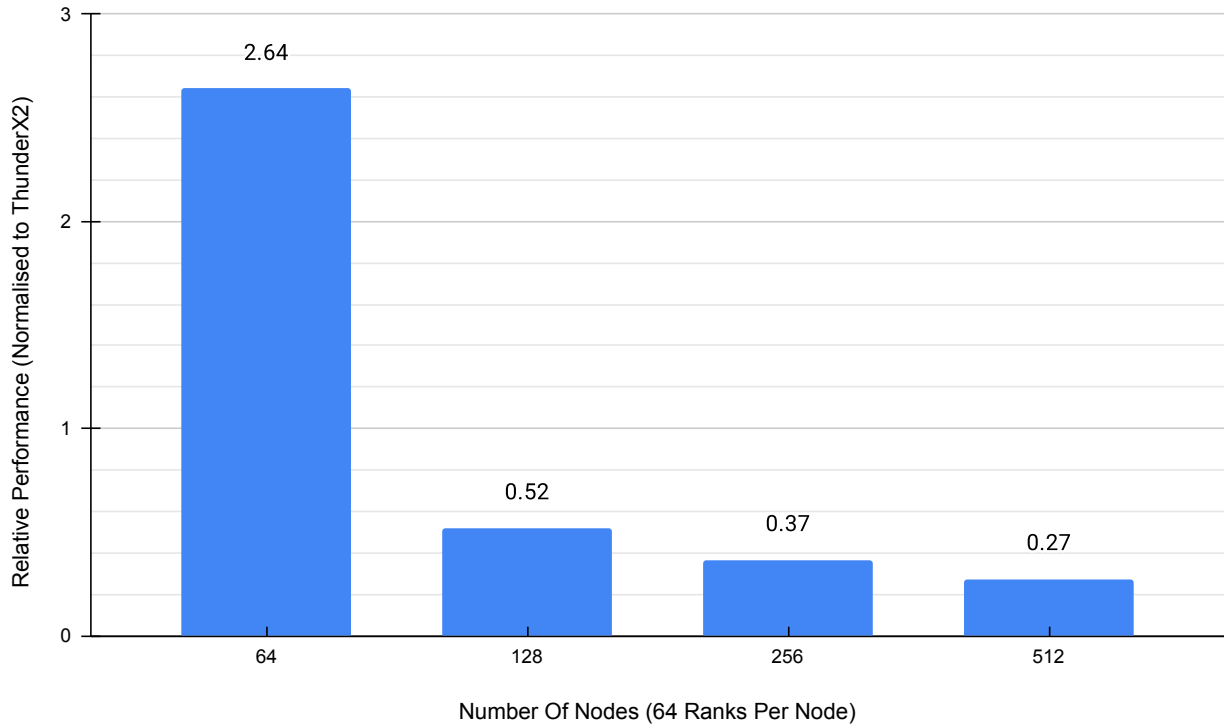


Figure 3.5: Intel MPI Benchmark showing the relative performance of MPI_Barrier on Graviton3, normalised to ThunderX2. 64 ranks per node

Figure 3.5 shows that Graviton3 has significant latency between nodes relative to ThunderX2. Intra-node communication is 2.64x faster than ThunderX2, whereas inter-node latency only increases as more nodes are added, relative to ThunderX2. As the layout of the Graviton3 cluster is not public, the exact cause of the latency is unknown, however causes could include, the physical distance between nodes, the switching delay between nodes, or configurations of packet prioritisation.

Inter-node communication for Graviton3 has significant performance issues as demonstrated by these results from the Intel MPI benchmark suite. Inter-node MPI communication performance is crucial for any HPC systems, with these results doubts over the suitability of Amazon’s cloud-base HPC offering, Graviton3, as a variable alternative to traditional supercomputers like Isambard.

3.3 SPEChpc2021

In this section, the systems will be evaluated using results from the tiny and small suites of SPEChpc2021, which are scaled up to eight nodes. Furthermore, the result from the synthetic benchmarks (3.1 and 3.2), will be used to augment the evaluation. When comparing systems, the scaling results are normalised to ThunderX2, showing the difference in performance of the two systems.

MPI and MPI+OMP programming models were both tested. Allowing for the analyse of different programming models between the two systems. For MPI+OMP, 1 MPI rank per node was used for Graviton3 due to the single NUMA region, and 2 MPI ranks per node were used for ThunderX2, with each MPI rank being placed on separate NUMA domains to reduce cross-NUMA memory traffic. Small

size LBM failed to run on Graviton3 at 4 nodes due to insignificant memory, and is therefore omitted in the graphs for that size.

Code for both were compiled using the compiler and environment listed in section 2.5. ThunderX2 used the compiler flags of `-Ofast -mcpu=thunderx2t99 -mtune=thunderx2t99 -fopenmp`, with Graviton3 using `-Ofast -mcpu=neoverse-512tvb -lm -ftree-vectorize` for C/C++, and `-O3 -mcpu=neoverse-512tvb` for Fortran. Changing the compiler flag and environment variables to tweak MPI and OpenMP, could result in improved performance. However, this project is focused on the comparison between systems, not micro optimisations of code, thus a good base level of optimisation was used.

Profiling data on ThunderX2 was collected using the Cray Performance Analysis Tool (known as CrayPat), with Graviton3 profiling data being collected with the Tuning and Analysis Utilities (TAU) profiler. Graviton3 however has no profiling data for codes using Fortran, this is due to a compiler error when using the TAU compiler wrapper. MiniSweep is also missing profiler data due to a segmentation fault when running. Moreover, profiling data is limited on Graviton due to no Performance Application Programming Interface (PAPI) events being available, thus only function run time data was collected. The absence of available PAPI event flags for Graviton3 presents a notable drawback of the processor for use in HPC. Profiling plays a crucial role in identifying bottleneck within codes, enable them to be fixed to improve code performance. Therefore, the limited profiling capabilities of Graviton3 result a significant drawback compared to ThunderX2.

3.3.1 MPI

For the analysis of the MPI programming model of SPEChpc 2021, profilers were used to collect the run time percentage of MPI calls. Profilers were run on both systems for the tiny and small suites at 4 and 8 ranks respectively. Figure 3.6 and Figure 3.8 show the data collected from ThunderX2, and Figure 3.7 and Figure 3.9 show the data for Graviton. On the same systems, the percentage of run time for each type of MPI call is similar between the suites, with some differences in a couple of benchmarks as shown. MPI calls are a large factor in many of the benchmark total run time, with some benchmarks having a significant percentage of their run time spent doing MPI calls, namely small suite MiniSweep at 68.7% on ThunderX2, while others like LBM only have 4.5%.

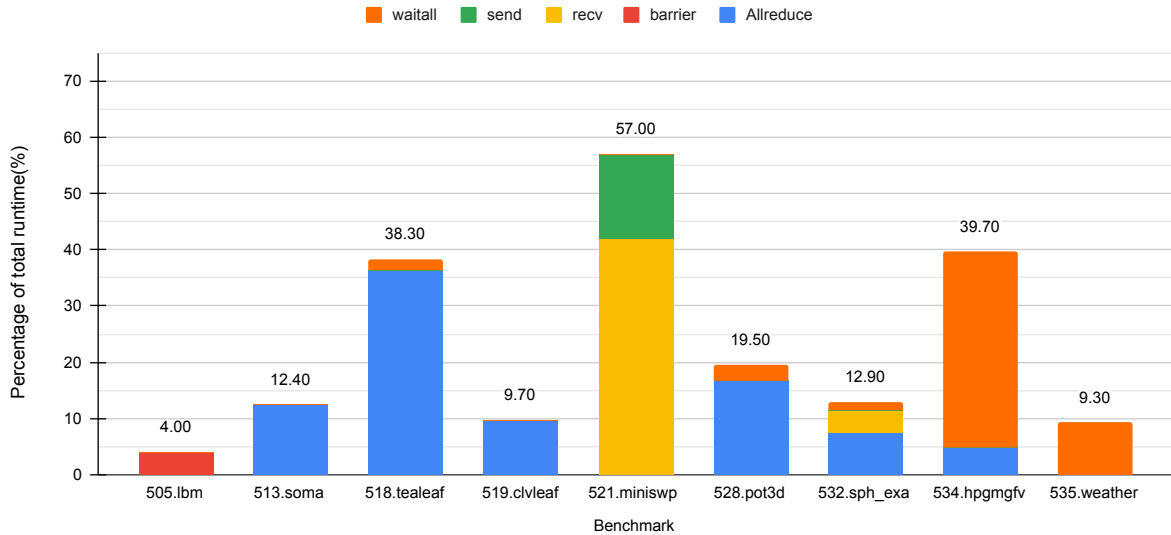


Figure 3.6: MPI functions as a percentage of run time on Isambard for tiny suite. Captured on 4 nodes on MPI-only model with Cray Pat

Application scaling: Most applications roughly scale linearly up to the maximum of 8 nodes in both suites, other than SOMA and Minisweep, which scale relatively poorly. In both suites, Minisweep scales the worst of all the benchmarks, with efficiency dropping below 60% in the tiny suite after 4 nodes, and only achieving 66% efficiency in the small suite at 8 nodes. This is due to Minisweep being bottlenecked by large amounts of MPI communication, totaling 68% of the run time for the small suite, and 57% of the run time for the tiny suite. Weather and LBM scale the best across both suites, with LBM achieving

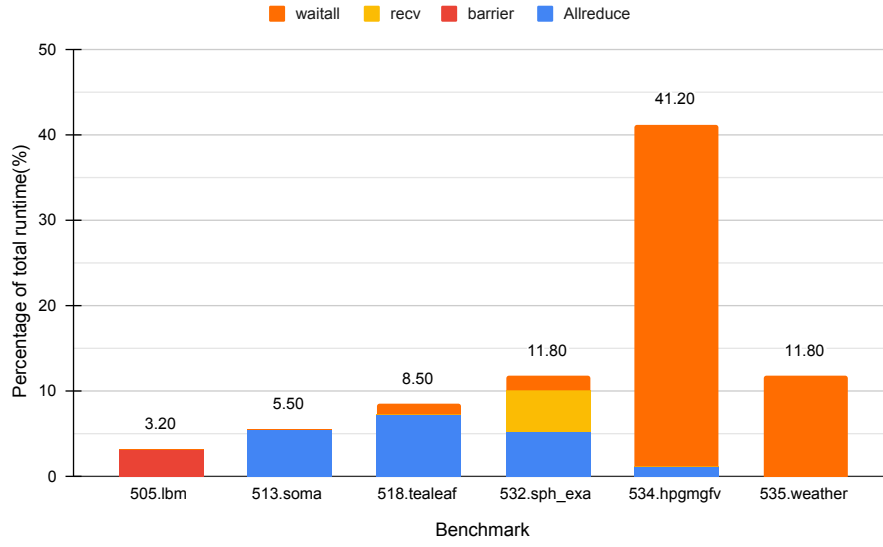


Figure 3.7: MPI functions as a percentage of run time on Graviton3 for tiny suite. Captured on 4 nodes on MPI-only model with Tau

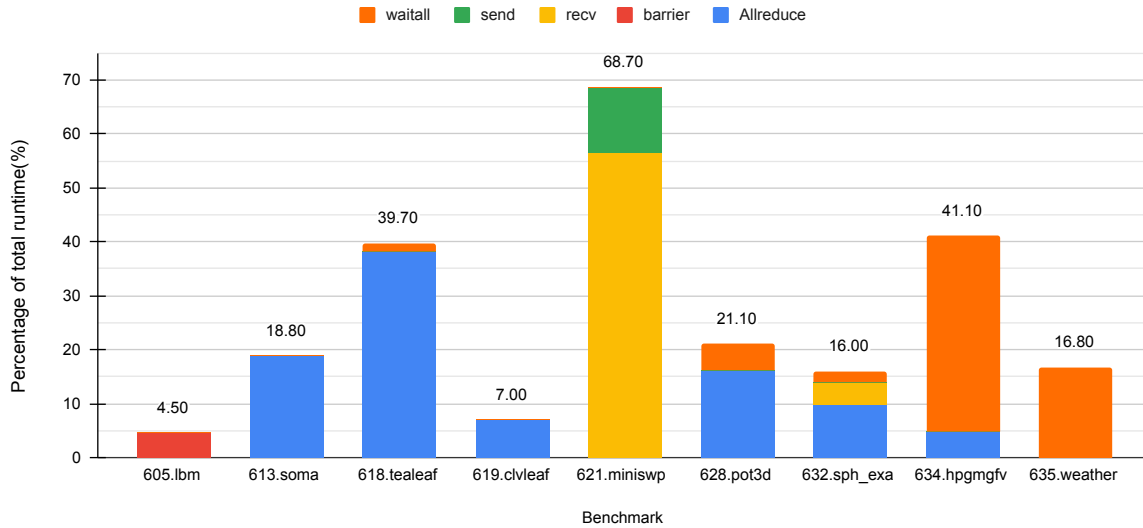


Figure 3.8: MPI functions as a percentage of run time on Isambard for small suite. Captured on 8 nodes on MPI-only model with Cray Pat

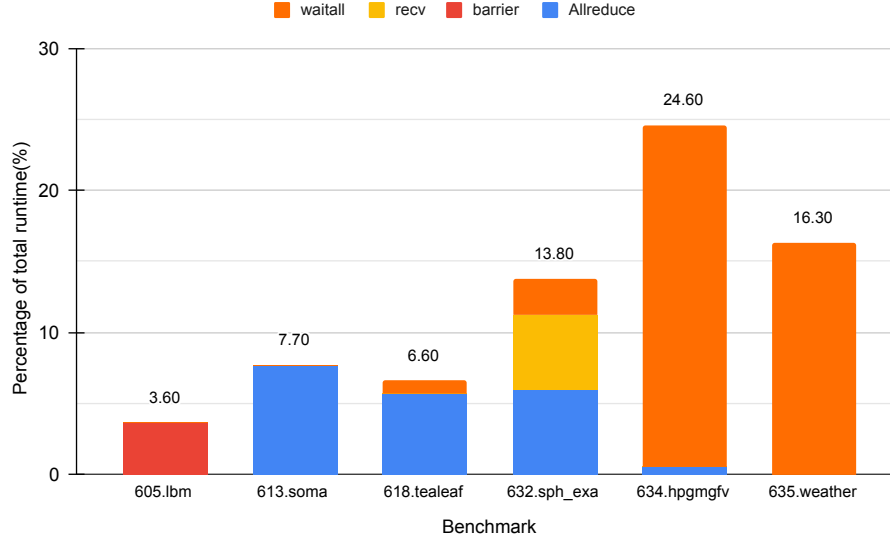


Figure 3.9: MPI functions as a percentage of run time on Graviton3 for small suite. Captured on 8 nodes on MPI-only model with Tau

a peak efficiency of 97% in the tiny suite at 4 nodes, and 99% in the small suite at 7 nodes. This is due to LBM minimal MPI communication, taking a total of 4.5% of the run time for the small suite, and 4% for tiny the tiny suite. Regarding Weather, it achieves 1.02% efficiency in both suites at 8 nodes, with efficiency only increase with node count.

Figure 3.12 and Figure 3.13 show the scaling performance of Graviton3 normalised to ThunderX2 for both suites. The first notable observation is the negative scaling for many of the benchmarks on node counts that are not a power of two, of which Pot3D and Tealeaf have the highest levels of regression. For Pot3D, there is a significant increase in the time to complete the CGDOT function, which is a function that gets the dot product of two vectors, then performs a MPI all_reduce sum on the result of the vector dot product. The function takes 7.86x longer on 7 nodes compared to 8 nodes, an increase to 36.8% of total run time, up from just 8.0% of total run time. Tealeaf, which suffers from a similar significant decrease in performance, also sees its MPI allreduce function calls jump from 5.7% to 43.1% of total run time, a 7.56x increase. This is notable as seen in Figure 3.4, Graviton3 suffers from extremely poor MPI allreduce performance when communicating between nodes, at messages of certain sizes, which suggests that this performance scaling issue is due to MPI issues with Graviton3.

The scaling issues due to MPI are unfortunate, because ignoring the performance regressions, Graviton3 shows strong performance improvements over ThunderX2, with LBM staying consistently above 3x the performance of ThunderX2, and peaking at 3.46x faster on 1 node for the tiny suite. This is because of a couple of factors, firstly LBM has the smallest percentage of MPI run time at 3.6% for the small suite, reducing the penalty from Graviton3 MPI bottleneck. Secondly, LBM benefits from Graviton3 additional vector units, along with the increased L2 cache size.

3.3.2 MPI+X

Profiling results of the benchmarks in the hybrid MPI+OpenMP model on ThunderX2 is shown in Figure 3.14. The results were gathered at 8 nodes, however Cloverleaf is omitted due to profiling errors when collecting data in the hybrid model. Graviton profiling results are shown in Figure 3.15, containing only Tealeaf, SPH, HPG and MiniWeather. This is because compiling errors with the Fortran codes, with the other benchmark failing to complete. Most of the applications see a reduction in MPI traffic as a percentage run time, with Soma and Pot3D not seeing a reduction.

The negative scaling observed in the MPI only model on Graviton3 is present again for the hybrid MPI+OpenMP model for node counts that are not a power of two as depicted in Figure 3.16 and Figure 3.17. Once again, Pot3D saw a significant increase in its MPI allreduce “CGDOT” functions, jumping from 1.55% of run time on 8 nodes, to 29.5% of run time on 7 nodes, a 19x increase in run time in the small suite. Minisweep however, is affected by odd number node counts in the tiny suite, resulting in performance being 0.84x of ThunderX2, down from 1.84x on 6 nodes. LBM on Graviton3 once again

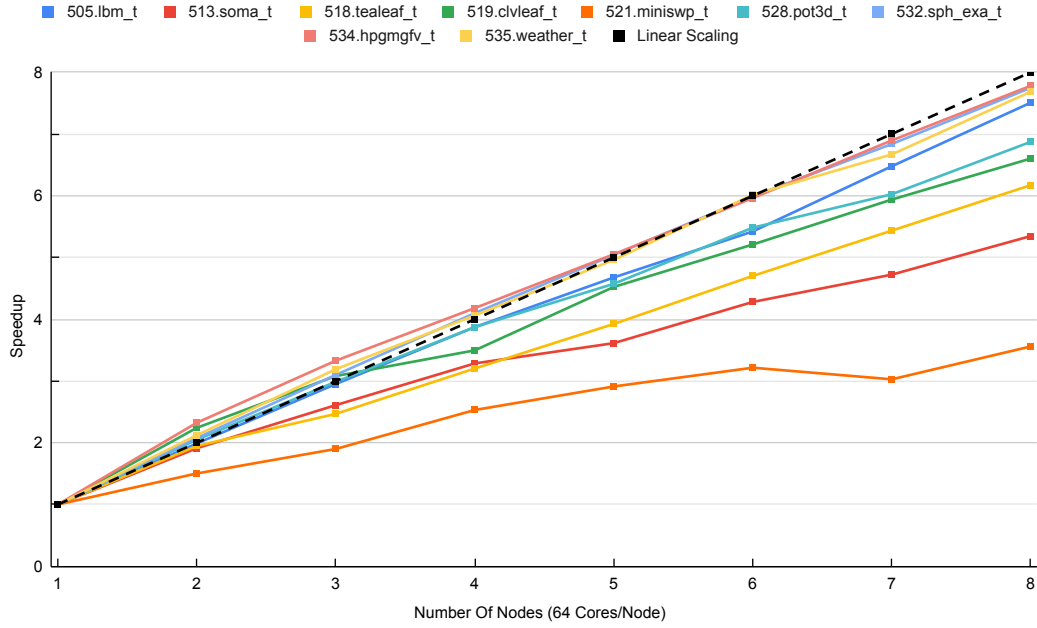


Figure 3.10: Scaling performance of MPI-only on ThunderX2 for tiny suite

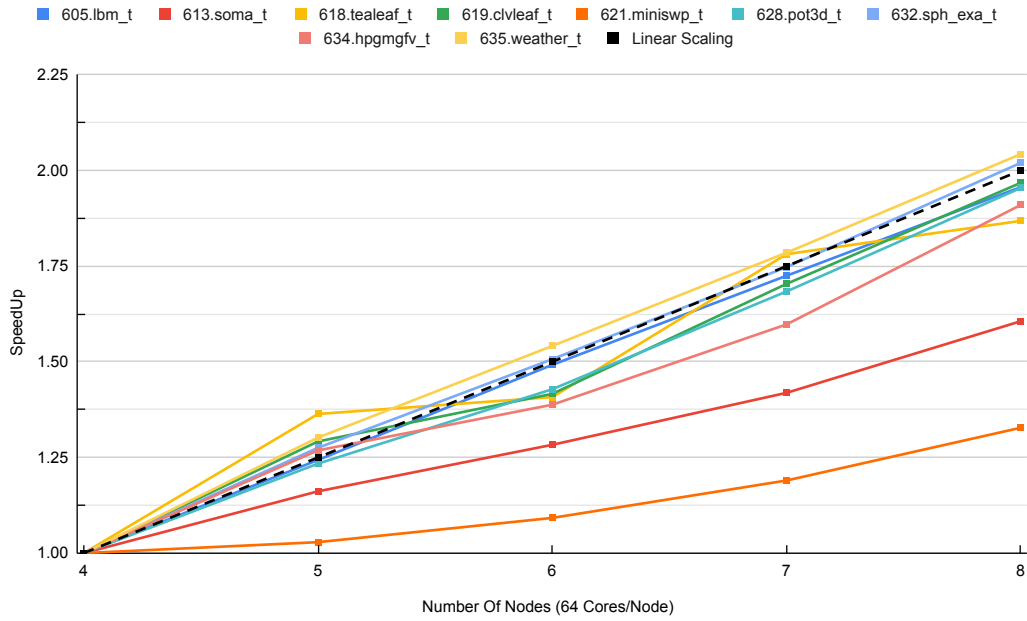


Figure 3.11: Scaling performance of MPI-only on ThunderX2 for small suite

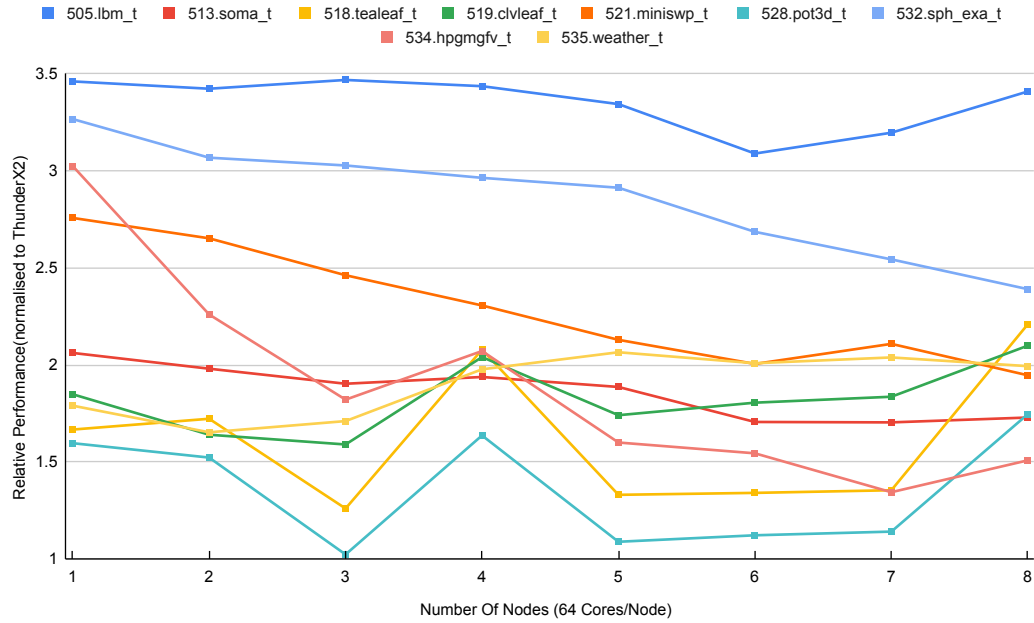


Figure 3.12: Relative scaling performance of MPI-only on Graviton3 for tiny suite, normalised to ThunderX2

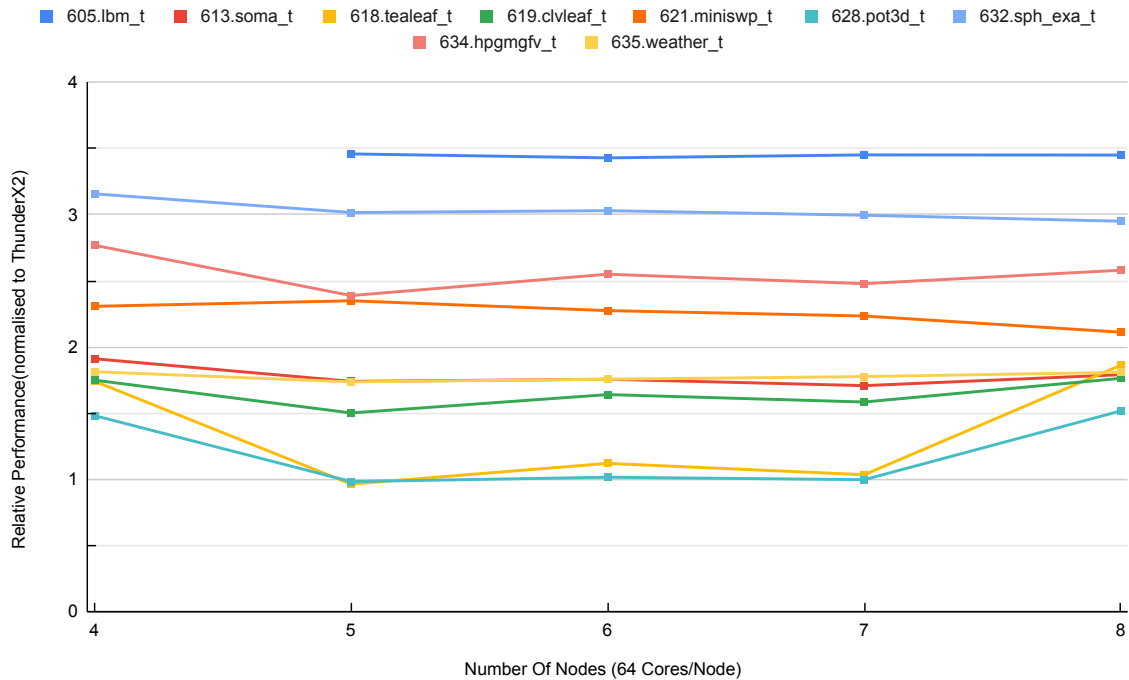


Figure 3.13: Relative scaling performance of MPI-only on Graviton3 for small suite, normalised to ThunderX2. 605.lbm_s is unable to run on 4 nodes on Graviton due to exceeded memory capacity

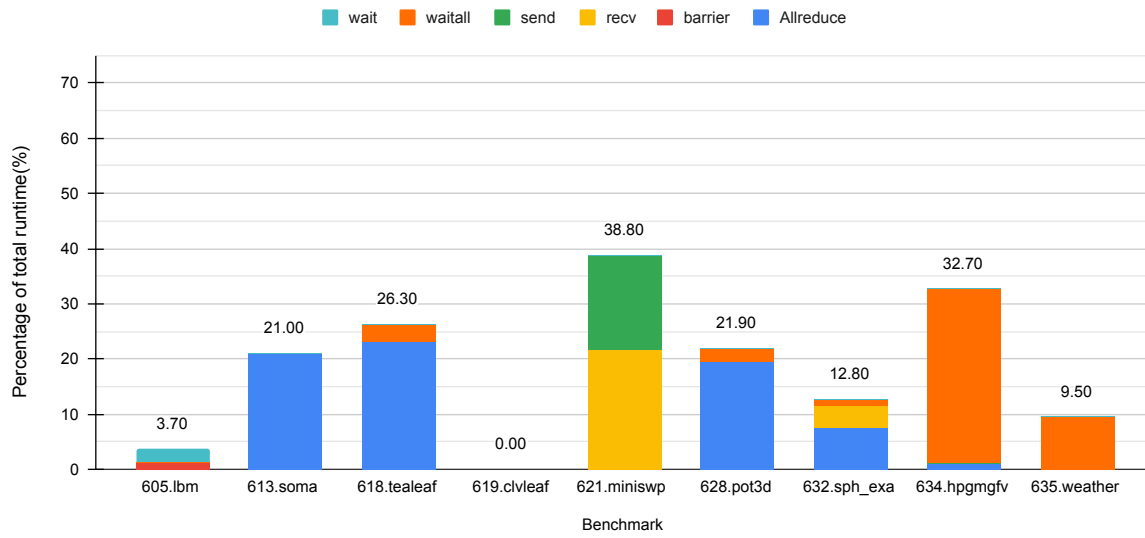


Figure 3.14: MPI functions as a percentage of run time on Isambard for small suite. Captured on 8 nodes on MPI+OpenMP with Cray Pat. 619.clvleaf_s is missing due to Cray Pat unable to run MPI+OpenMP model

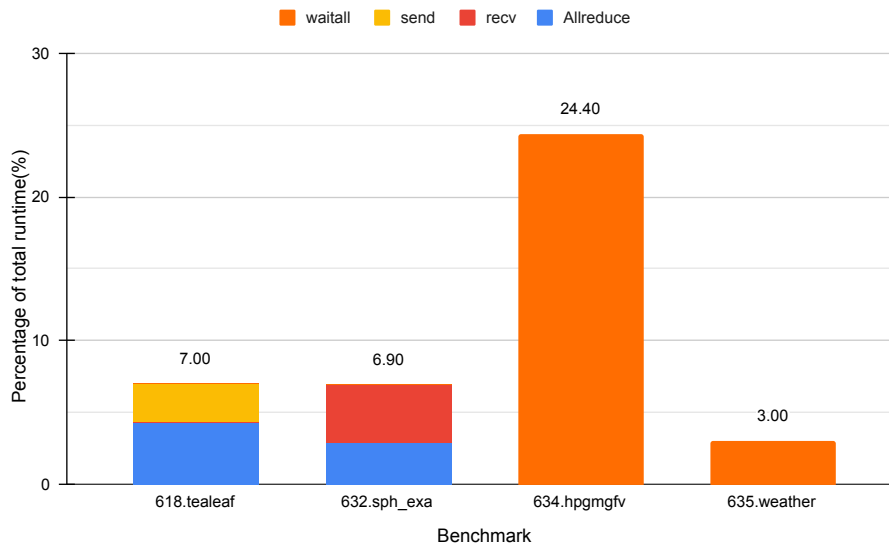


Figure 3.15: MPI functions as a percentage of run time on ThunderX2 for small suite. Captured on 8 nodes on MPI+OpenMP with TAU.

maintains a consistent 3-3.5x improvement over ThunderX2 across both suite sizes, with the minimal MPI traffic allowing LBM to not be as affected by the inter-node communication issues of Graviton3.

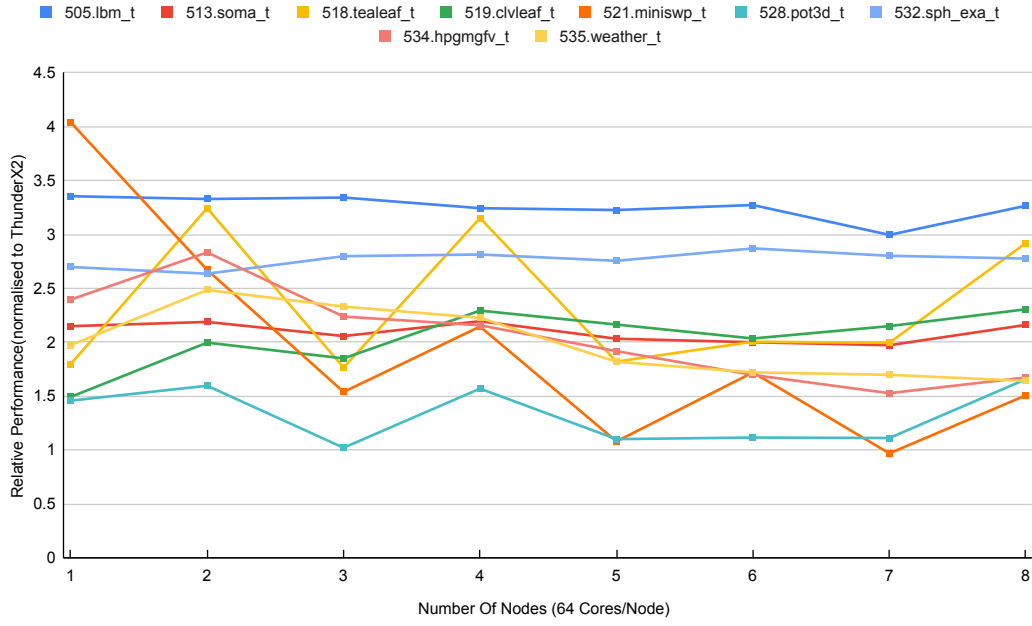


Figure 3.16: Relative scaling performance of MPI+OpenMP on Graviton3 for tiny suite, normalised to ThunderX2

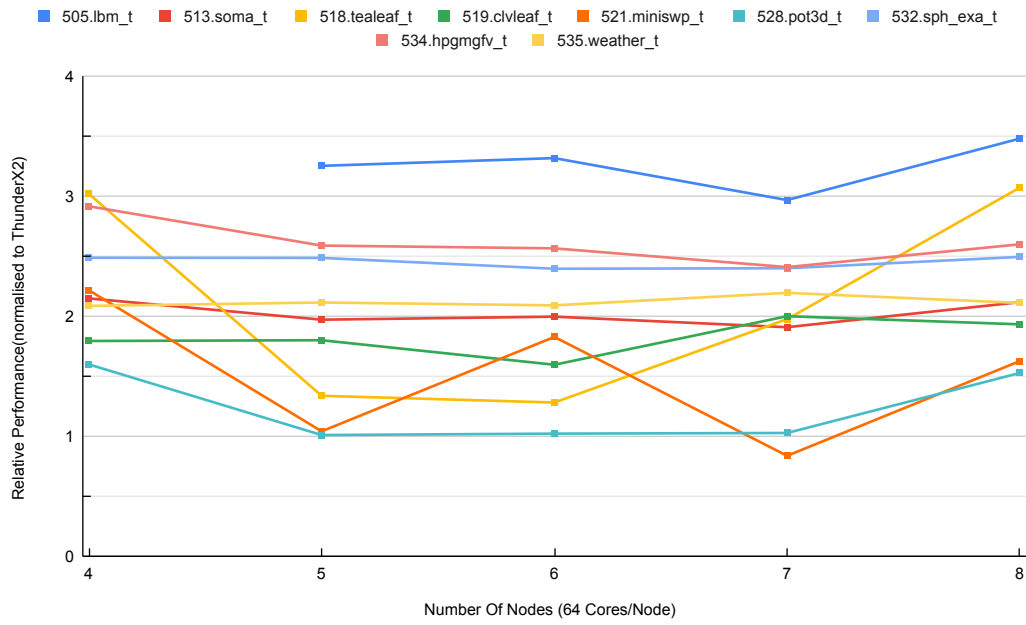


Figure 3.17: Relative scaling performance of MPI-OpenMP on Graviton3 for small suite, normalised to ThunderX2. 605.lbm_s is unable to run on 4 nodes on Graviton due to exceeded memory capacity

Chapter 4

Conclusion

4.1 Contributions and Achievements

In this project, the performance of Amazon’s latest cloud HPC offering, Graviton3, was compared to Isambard, a traditional HPC resource, for the evaluation of cloud HPC as an alternative to the traditional supercomputer model. Furthermore, the results in this paper are another demonstration of a performance competitive Arm-based processor for HPC. A range of synthetic and application-based benchmarks results were presented, namely STREAM, Intel MPI benchmarks, and SPEChpc2021, totaling more than 750 compute hours of total run time. Strong-scaling results were presented for SPEChpc2021 across tiny and small suite sizes, up to the maximum number of available Graviton3 nodes of eight. The benchmarks of SPEChpc2021 were then profiled on both systems, to collect MPI run time percentage, adding to the evaluation of inter-node communication of Graviton3. The results demonstrated that Amazon’s Graviton3 processor is an impressive performance improvement over Marvel ThunderX2, with an up to 3.48x increase shown for LBM code(Figure 3.13. The project also highlighted issues with Graviton3 inter-node communication, resulting in significant performance regression when scaling, which was shown in the results for the SPEChpc2021 benchmarks.

Thus, this project highlighted the potential benefits of cloud HPC as a service in regards to upfront costs and scalability, and showed Amazon’s Graviton3 having significant performance improvement over Marvel ThunderX2. However, it also showed that Amazon’s cloud HPC service comes with inter-node communication issues with high-latency, which will need to be addressed if Amazon wants their cloud HPC service to be a true alternative to traditional HPC systems like Isambard.

4.2 Project Status

Use a suite of benchmarks to collect data about the characteristic of each system

In Section 3, I have presented a wide range of carefully chosen benchmarks that demonstrate important characteristics to HPC application. Using the benchmarks: STREAM, Intel MPI benchmark, and SPEChpc 2021, a comprehensive data obtained, which allowed for the evaluation of the two systems.

Evaluate the scaling performance of Graviton3 compared to Isambard

In Section 3, SPEChpc 2021 tiny and small suites were run on both systems, scaling from one to eight nodes, and four to eight nodes respectively. Normalising the results to ThunderX2, I was able to demonstrate Graviton3’s scaling performance, in comparison to ThunderX2.

Identify any bottlenecks or performance limitations with Graviton3

In Section 3, I identified inter-node communication issues with Graviton3, reducing the performance of the system when communicating between nodes.

Determine whether Graviton3 is a viable alternative to Isambard

As discussed in Section 3, the processor performance of Graviton3 would make Graviton3 not only a viable alternative, but an improvement over Isambard. However, due to the inter-node communication

bottlenecks, the viable of Graviton3 as a alternative to Isambard is greatly diminished .

4.3 Future Plans

4.3.1 Evaluate Scaling on a Larger Cluster

The evaluation of the scaling performance of Graviton3 was limited by the available cluster, which consists of only eight nodes. Eight nodes is a small cluster in HPC system terms, with Isambard containing 329 nodes [1]. To provide a more accurate evaluation of Graviton3 as an alternative to traditional supercomputers, it is imperative to conduct scaling experiments on larger clusters. Furthermore, scaling to a larger number of nodes requires good inter-node communication performance, which Graviton3 struggles with, therefore allowing for the better characterisation of the communication issues.

4.3.2 Investigate Inter-node Communication Issues

Gaining a clear understanding of the underlying causes behind the communication issues between nodes in Graviton3 is crucial for identifying effective solutions or mitigation's to them. By addressing these issues, performance improvements can be achieved across various workloads, including the performance regression seen in section 3.3. Resolving the Inter-node Communication Issues would result in Graviton3 being a true alternative to traditional supercomputers.

4.3.3 Cost Analysis Investigation

One of the major benefits of Graviton3 is its cost model, which is a pay-as-you-go model, typically based on an hourly rate. This stands in contrast to the substantial upfront costs associated with traditional supercomputers, which also entail ongoing expenses even when not being utilised. Therefore, conducting an extensive cost analysis between Isambard and Graviton3 across a wide range of HPC applications and cluster scales would be of significant interest to supercomputer funders. If a more cost-effective alternative with comparable performance exists, it could potentially offer a compelling proposition, leading to potential cost savings for organisations investing in super-computing capabilities.

Bibliography

- [1] South west to host europe's largest arm supercomputer, 2020. URL: <https://www.bristol.ac.uk/news/2020/february/gw4isambard-.html>.
- [2] Announcing new amazon ec2 c7g instances powered by aws graviton3 processors, 2021. URL: <https://aws.amazon.com/about-aws/whats-new/2021/11/amazon-ec2-c7g-instances-aws-graviton3-processors/>.
- [3] Amazon. Amazon c7g instance information, 2023. URL: <https://aws.amazon.com/ec2/instance-types/c7g/>.
- [4] Amazon. Amazon ec2 on-demand pricing, 2023. URL: <https://aws.amazon.com/ec2/pricing/on-demand/>.
- [5] AWS. Aws graviton getting started. URL: <https://github.com/aws/aws-graviton-getting-started>.
- [6] J. Barr. Ec2 instances (a1) powered by arm-based aws graviton processors, 2018. URL: <https://aws.amazon.com/blogs/aws/new-ec2-instances-a1-powered-by-arm-based-aws-graviton-processors/>.
- [7] Holger Brunst, Sunita Chandrasekaran, Florina M. Ciorba, Nick Hagerty, Robert Henschel, Guido Juckeland, Junjie Li, Verónica G. Melesse Vergara, Sandra Wienke, and Miguel Zavala. First experiences in performance benchmarking with the new spechpc 2021 suites. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 675–684, 2022. doi:10.1109/CCGrid54584.2022.00077.
- [8] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. doi:10.1109/99.660313.
- [9] M.J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966. doi:10.1109/PROC.1966.5273.
- [10] Wei Gao, Yunbo Zhang, Devarajan Ramanujan, Karthik Ramani, Yong Chen, Christopher B. Williams, Charlie C.L. Wang, Yung C. Shin, Song Zhang, and Pablo D. Zavattieri. The status, challenges, and future of additive manufacturing in engineering. *Computer-Aided Design*, 69:65–89, 2015. URL: <https://www.sciencedirect.com/science/article/pii/S0010448515000469>, doi:<https://doi.org/10.1016/j.cad.2015.04.001>.
- [11] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996. URL: <https://www.sciencedirect.com/science/article/pii/0167819196000245>, doi: [https://doi.org/10.1016/0167-8191\(96\)00024-5](https://doi.org/10.1016/0167-8191(96)00024-5).
- [12] S.D. Hammond, C. Hughes, M.J. Levenhagen, C.T. Vaughan, A.J. Younge, B. Schwaller, M.J. Aguilar, K.T. Pedretti, and J.H. Laros. Evaluating the marvell thunderx2 server processor for hpc workloads. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, pages 416–423, 2019.
- [13] S.D. Hammond, C. Hughes, M.J. Levenhagen, C.T. Vaughan, A.J. Younge, B. Schwaller, M.J. Aguilar, K.T. Pedretti, and J.H. Laros. Evaluating the marvell thunderx2 server processor for hpc workloads. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, pages 416–423, 2019. doi:10.1109/HPCS48598.2019.9188171.

- [14] Intel. Introducing intel mpi benchmarks, 2018. URL: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-mpi-benchmarks.html>.
- [15] Intel. Intel mpi benchmarks user guide, 2021. URL: <https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-2/overview.html>.
- [16] J. Linford. Compiler flags across architectures: -march, -mtune, and -mcpu, 2019. URL: <https://community.arm.com/developer/tools-software/tools/b/tools-software-ides-blog/posts/compiler-flags-across-architectures-march-mtune-and-mcpu>.
- [17] Marvell. Manufacturing applications on marvell thunderx2. 2019.
- [18] John D McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, 1995. URL: <https://www.sciencedirect.com/science/article/pii/0167819196000245>.
- [19] Simon McIntosh-Smith, James Price, Tom Deakin, and Andrei Poenaru. A performance analysis of the first generation of hpc-optimized arm processors. *Concurrency and Computation: Practice and Experience*, 31(16):e5110, 2019. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5110>, [arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5110](https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5110), doi:<https://doi.org/10.1002/cpe.5110>.
- [20] G. E. Moore. Cramming more components onto integrated circuits. 1965.
- [21] Nvidia. Nvidia ada gpu architecture. page 7.
- [22] SPEChpc2021 System Requirements. Spec. URL: <https://www.spec.org/hpg/hpc2021/Docs/system-requirements.html>.
- [23] Amazon Web Services. Re: Invent conference, 2021.
- [24] SPEC. Documentation overview. URL: <http://spec.org/hpc2021/Docs/overview.html>.
- [25] SPEC. Spec. URL: <https://www.spec.org/>.
- [26] SPEC. Spec benchmark search program. URL: <https://www.spec.org/hpg/search/>.
- [27] SPEC. Spec hpg: High performance group. URL: <https://www.spec.org/hpg/>.
- [28] SPEC. Spec news. URL: <https://www.spec.org/news>.
- [29] SPEC. Spec organizational information. URL: <https://www.spec.org/spec/>.
- [30] SPEC. Spechpc2021 results repository. URL: <https://www.spec.org/results.html>.
- [31] SPEC. Spechpc 2021 press release, 2021. URL: <https://www.spec.org/hpc2021/press/release.html>.
- [32] T. N. Theis and H.-S. P. Wong. The end of moore’s law: A new beginning for information technology. 19:41–50, 2017.
- [33] GW4 User Guide XCI Marvell Thunder X2. Gw4. URL: <https://gw4-isambard.github.io/docs/user-guide/XCI.html>.