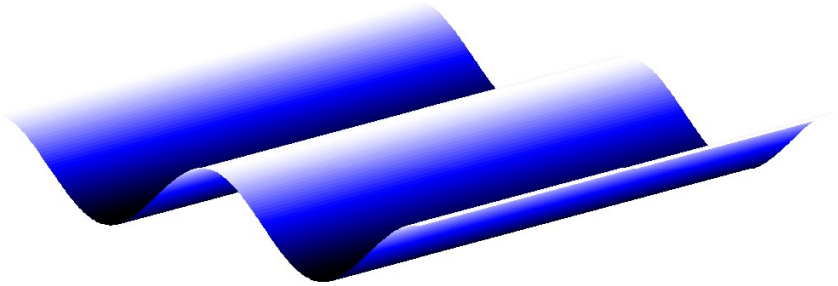
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Frithjof Brate

Masking Communication Overhead for a Finite Difference Shallow Water Equations Solver

Master's thesis in Computer Science
Supervisor: Jan Christian Meyer
June 2023



NTNU
Norwegian University of
Science and Technology

Frithjof Brate

Masking Communication Overhead for a Finite Difference Shallow Water Equations Solver

Master's thesis in Computer Science
Supervisor: Jan Christian Meyer
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Problem Description

This study aims to develop a performance model of a distributed memory finite difference solver for the Shallow Water Equations, quantify its potential for reducing communication overhead, and test the model experimentally.

Abstract

In this thesis, we focus on investigating the impact of varying border exchange thicknesses for a proxy application. The amount of data in the exchanges impact the computational load, and communication overhead. The proxy application is numerically solving the Shallow Water Equations, which are a set of partial differential equations that describe the flow of water in a pressure surface.

We create a performance model using Bulk Synchronous Parallel model to analyze the various costs attributed to the different states in the proxy application. We gather data for sub-domains sizes ranging from $50 \cdot 50$ to, $1000 \cdot 1000$ with increments of 50 for varying border exchange thicknesses of 1-20. The costs of the states are then used to predict the calculation and communication time of the sub-domain size and border exchange thickness configurations. The configurations that predict a performance benefit are experimentally tested to see if the performance model is able to characterize the proxy application.

The results show that there are configurations where a speedup can be achieved, but the performance model is lacking accuracy and needs further improvements to accurately model the proxy application.

Sammendrag

I denne avhandlingen fokuserer vi på å undersøke effekten av varierende tykkelser på kantutvekslinger for en proxy-applikasjon. Mengden data i utvekslingene påvirker kostnaden av beregningene og overheaden i forbindelse med kommunikasjon. Proxy-applikasjonen løser gruntvannlikningene, som er en serie av partielle differensiallikninger som beskriver vannstrømmen i en flate.

Vi lager en ytelsesmodell ved hjelp av Bulk Synchronous Parallel-modellen for å analysere de ulike kostnadene knyttet til de ulike tilstandene i proxy-applikasjonen. Vi samler inn data for sub-domener fra $50 \cdot 50$ til $1000 \cdot 1000$, med en økning på 50 for hver størrelse, og med varierende tykkelser på kantutvekslingen fra 1 til 20. Kostnadene for tilstandene brukes deretter til å forutsi beregnings- og kommunikasjonstiden for hver kombinasjon av sub-domenestørrelser og kanttykkelser. Konfigurasjonene som forutsier en ytelsesfordel, blir eksperimentelt testet for å se om ytelsesmodellen er i stand til å karakterisere proxy-applikasjonen.

Resultatene viser at det er konfigurasjoner der en hastighetsøkning kan oppnås, men ytelsesmodellen modellerer ikke proxy-applikasjonen helt nøyaktig, og trenger videre forbedringer.

Acknowledgements

I would like to thank my supervisor Jan Christian Meyer for guiding me through the field of high performance computing, performance modelling, valuable lessons regarding academic writing, partial differential equations, inspiration and motivation.

The computations were performed on resources provided by Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abbreviations

BSP	=	Bulk Synchronous Parallel
CPU	=	Central Processing Unit
FDM	=	Finite Difference Method
FTCS	=	Forward Time Central Space
HPC	=	High Performance Computing
LCM	=	Least Common Multiple
MIMD	=	Multiple Instruction (Stream) Multiple Data (Stream)
MPI	=	Message Passing Interface
NIC	=	Network Interface Card
Q1	=	25th percentile in a boxplot.
Q3	=	75th percentile in a boxplot.
RDMA	=	Remote Direct Memory Access
SWE	=	Shallow Water Equations

Chapter 1

Introduction

The Shallow Water Equations (SWE) are a set of partial differential equations used to model the flow of water, such as waves and tides[3]. The equations can be solved numerically by a proxy application using a finite difference method (FDM). FDM requires neighboring values to compute, so we perform border exchanges to send the necessary data between processes. It is interesting how the parallel computation of the sub-domains are affected by varying the frequency of communication between the processes. This reduces how often the application encounter the overhead of communication, but the trade-off is that it also increases the computational load.

A performance model gives us the ability to identify configurations where a speedup can be achieved. These configurations can then be tested by the proxy application.

The goal of the thesis is to create a proxy application solver for the SWE using a FDM, create a performance model and identify how the varying frequency of border exchanges affect the overall performance of the application.

1.1 Scope

In this thesis we develop a proxy application solving the SWE using a Lax-Friedrich Scheme. It is parallelized in a horizontal topology using Message Passing Interface(MPI). The performance model is created by evaluating the cost of each super-step in the proxy application, which is done by gathering data for various sub-domains and border exchange thicknesses. The gathered data provide estimates of the overall run-time and performance benefits for certain sub-domains sizes and border exchange thicknesses. We run the proxy application to verify if the performance model's prediction is achievable. The tests are run on the Idun cluster, provided by Norwegian University of Science and Technology (NTNU), and Fram and Betzy clusters, provided by UNINETT Sigma2.

1.2 Structure

The remainder of this thesis has the following structure:

- In Chapter 2 we present the background material for SWE, FTCS, Lax-Friedrichs Scheme, Message Passing Interface (MPI), Bulk-Synchronous Parallel(BSP), enabling overlap between communication and computation, and reducing the communication overhead.

- In Chapter 3 we explore the states of the proxy application, and how they handle the varying border exchange sizes.

- In Chapter 4 we explain the performance model, the various benefits and downside of varying the border exchange thickness, how we gather data for the performance model, and how we inspect the data.

- In Chapter 5 we introduce the HPC clusters and the experimental setup for the proxy application and performance model.

- In Chapter 6 we compare the results of the performance model to the proxy application tests.

- Finally, we summarize our conclusion and topics for future work in Chapter 7.

Chapter 2

Background

In this chapter the background material is presented. We present a numerical solution of the SWE using Lax-Friedrichs Scheme, introduce the Message Passing Interface, Bulk Synchronous Parallel model, LogGP model, explain how we enable overlap between computation and communication, and how we can reduce the communication overhead in the proxy application.

2.1 Shallow Water Equations

The Shallow Water Equations (SWE) (2.1) - (2.3) are a set of partial differential equations that describe the flow of water in a pressure surface. The SWE are characterized by that the vertical dimension is smaller than the horizontal dimension for the grid. They are often used to model oceans, and rivers to predict tides, and currents[3].

$$\frac{\partial(\rho\eta)}{\partial t} + \frac{\partial(\rho\eta u)}{\partial x} + \frac{\partial(\rho\eta v)}{\partial y} = 0 \tag{2.1}$$

$$\frac{\partial(\rho\eta u)}{\partial t} + \frac{\partial}{\partial x}(\rho\eta u^2 + \frac{1}{2}\rho g\eta^2) + \frac{\partial(\rho\eta uv)}{\partial y} = 0 \tag{2.2}$$

$$\frac{\partial(\rho\eta v)}{\partial t} + \frac{\partial(\rho\eta uv)}{\partial x} + \frac{\partial}{\partial y}(\rho\eta v^2 + \frac{1}{2}\rho g\eta^2) = 0 \tag{2.3}$$

where

- u is the velocity in the x direction.
- v is the velocity in the y direction.
- g is acceleration of gravity.
- ρ is the fluid viscosity.
- η is the height (z axis).

In this thesis the computations use a single fluid viscosity where $\rho = 1$. The equations can therefore be simplified with Equation (2.4).

$$H = \rho\eta \tag{2.4}$$

The three Equations (2.1) - (2.3) can be simplified using Equation (2.4), which gives us Equations (2.5) - (2.7):

$$\frac{\partial(H)}{\partial t} = -\frac{\partial(Hu)}{\partial x} - \frac{\partial(Hv)}{\partial y} \tag{2.5}$$

$$\frac{\partial(Hu)}{\partial t} = -\frac{\partial}{\partial x}(Hu^2 + \frac{1}{2}\,Hg\eta) - \frac{\partial(Huv)}{\partial y} \tag{2.6}$$

$$\frac{\partial(Hv)}{\partial t} = -\frac{\partial(Huv)}{\partial x} - \frac{\partial}{\partial y}(Hv^2 + \frac{1}{2}Hg\eta) \tag{2.7}$$

2.1.1 FTCS

Solving the Shallow Water Equations with the finite difference method converts the problem domain to a grid of uniformly distributed points. By using Forward Time Central Space[4], we can rewrite the Shallow Water Equations expressions so that they use the points of neighboring grid values to approximate next-step values. We change the left side of the Equation (2.5) - (2.7) to utilize Forward Euler[5], and the right side with a finite central difference stencil method. This gives the following Equations (2.8) - (2.10).

$$\frac{H^{t+1}_{x,y} - H^{t}_{x,y}}{\Delta t} = -\frac{(Hu)^{t}_{x+1,y} - (Hu)^{t}_{x-1,y}}{2\Delta x} - \frac{(Hv)^{t}_{x,y+1} - (Hv)^{t}_{x,y-1}}{2\Delta y} \tag{2.8}$$

$$\frac{(Hu)^{t+1}_{x,y} - (Hu)^{t}_{x,y}}{\Delta t} = -\frac{(Huv)^{t}_{x,y+1} - (Huv)^{t}_{x,y-1}}{2\Delta y} - \frac{(Hu^2)^{t}_{x+1,y} - (Hu^2)^{t}_{x-1,y} + \frac{1}{2}(Hg\eta)^{t}_{x+1,y} - \frac{1}{2}(Hg\eta)^{t}_{x-1,y}}{2\Delta x} \tag{2.9}$$

$$\frac{(Hv)^{t+1}_{x,y} - (Hv)^{t}_{x,y}}{\Delta t} = -\frac{(Huv)^{t}_{x+1,y} - (Huv)^{t}_{x-1,y}}{2\Delta x} - \frac{(Hv^2)^{t}_{x,y+1} - (Hv^2)^{t}_{x,y-1} + \frac{1}{2}(Hg\eta)^{t}_{x,y+1} - \frac{1}{2}(Hg\eta)^{t}_{x,y-1}}{2\Delta y} \tag{2.10}$$

where

- superscript $(^t)$ is for the current time-step (known value).
- superscript $(^{t+1})$ is for the next-step approximate value.
- subscript (x, y) is for the value at position (x, y) in the grid.

Finally, the Δt, $H^t_{x,y}$, $(Hu)^t_{x,y}$ and $(Hv)^t_{x,y}$ can be moved to the right side of the equation, creating Equation (2.11) - (2.13), which calculates the mass, velocity in x-direction (u) and velocity in y-direction (v), using values from the previous time-step. Because we are working with a uniformly distributed grid, we can substitute $\Delta x = \Delta y = h$. All the equations estimate next-step values in the grid by using the known values of the current time-step.

$$H^{t+1}_{x,y} = H^t_{x,y} - \Delta t[\frac{(Hu)^t_{x+1,y} - (Hu)^t_{x-1,y}}{2h} + \frac{(Hv)^t_{x,y+1} - (Hv)^t_{x,y-1}}{2h}] \tag{2.11}$$

$$(Hu)^{t+1}_{x,y} = (Hu)^t_{x,y} - \Delta t \cdot [\frac{(Huv)^t_{x,y+1} - (Huv)^t_{x,y-1}}{2h} + \frac{(Hu^2)^t_{x+1,y} - (Hu^2)^t_{x-1,y} + \frac{1}{2}(Hg\eta)^t_{x+1,y} - \frac{1}{2}(Hg\eta)^t_{x-1,y}}{2h}] \tag{2.12}$$

$$(Hv)^{t+1}_{x,y} = (Hv)^t_{x,y} - \Delta t \cdot [\frac{(Huv)^t_{x+1,y} - (Huv)^t_{x-1,y}}{2h} + \frac{(Hv^2)^t_{x,y+1} - (Hv^2)^t_{x,y-1} + \frac{1}{2}(Hg\eta)^t_{x,y+1} - \frac{1}{2}(Hg\eta)^t_{x,y-1}}{2h}] \tag{2.13}$$

2.1.2 Lax-Friedrichs Scheme

Forward Time Central Space is not numerically stable for the two-dimensional Shallow Water Equations. To ensure numerical stability, we can replace the $H^t_{x,y}$, $(Hu)^t_{x,y}$ and $(Hv)^t_{x,y}$ of the Forward Euler formulation in Equations (2.8) - (2.10) with the spatial average as shown in Equations (2.14) - (2.16), called the Lax-Friedrich Scheme.

$$H^t_{x,y} = \frac{H^t_{x+1,y} + H^t_{x-1,y} + H^t_{x,y+1} + H^t_{x,y-1}}{4} \tag{2.14}$$

$$(Hu)^t_{x,y} = \frac{(Hu)^t_{x+1,y} + (Hu)^t_{x-1,y}}{2} \tag{2.15}$$

$$(Hv)^t_{x,y} = \frac{(Hv)^t_{x,y+1} + (Hv)^t_{x,y-1}}{2} \tag{2.16}$$

By inserting the replacements of Equations (2.14) - (2.16) in Equations (2.11) - (2.13) we get Equations (2.17) - (2.19), which estimate the next time-step value for the mass, velocity in the x-direction and velocity in the y-direction for the SWE using a Lax-Friedrichs Scheme.

$$H_{x,y}^{t+1} = \frac{H_{x+1,y}^{t} + H_{x-1,y}^{t} + H_{x,y+1}^{t} + H_{x,y-1}^{t}}{4} - \Delta t[\frac{(Hu)_{x+1,y}^{t} - (Hu)_{x-1,y}^{t}}{2h} + \frac{(Hv)_{x,y+1}^{t} - (Hv)_{x,y-1}^{t}}{2h}] \tag{2.17}$$

$$(Hu)_{x,y}^{t+1} = \frac{(Hu)_{x+1,y}^{t} + (Hu)_{x-1,y}^{t}}{2} - \Delta t \cdot [\frac{(Huv)_{x,y+1}^{t} - (Huv)_{x,y-1}^{t}}{2h} + \frac{(Hu^2)_{x+1,y}^{t} - (Hu^2)_{x-1,y}^{t} + \frac{1}{2}(Hgn)_{x+1,y}^{t} - \frac{1}{2}(Hgn)_{x-1,y}^{t}}{2h}] \tag{2.18}$$

$$(Hv)_{x,y}^{t+1} = \frac{(Hv)_{x,y+1}^{t} + (Hv)_{x,y-1}^{t}}{2} - \Delta t \cdot [\frac{(Huv)_{x+1,y}^{t} - (Huv)_{x-1,y}^{t}}{2h} + \frac{(Hv^2)_{x,y+1}^{t} - (Hv^2)_{x,y-1}^{t} + \frac{1}{2}(Hgn)_{x,y+1}^{t} - \frac{1}{2}(Hgn)_{x,y-1}^{t}}{2h}] \tag{2.19}$$

2.2 Boundary conditions

When calculating within a grid, it is necessary to define how the system reacts when the edges of the grid are encountered. In this simulated environment, we use Neumann boundary conditions to preserve the energy of the simulation by reflecting the energy back into the domain[6].

2.3 Message Passing Interface

From Flynn's taxonomy, our paper's experiment is conducted using the MIMD category [7]. The processes are responsible for computing their sub-domain, performing communication, and verifying successful message transmissions. We use the Message Passing Interface (MPI) to facilitate data transfer and computation across multiple processors (MPI ranks).

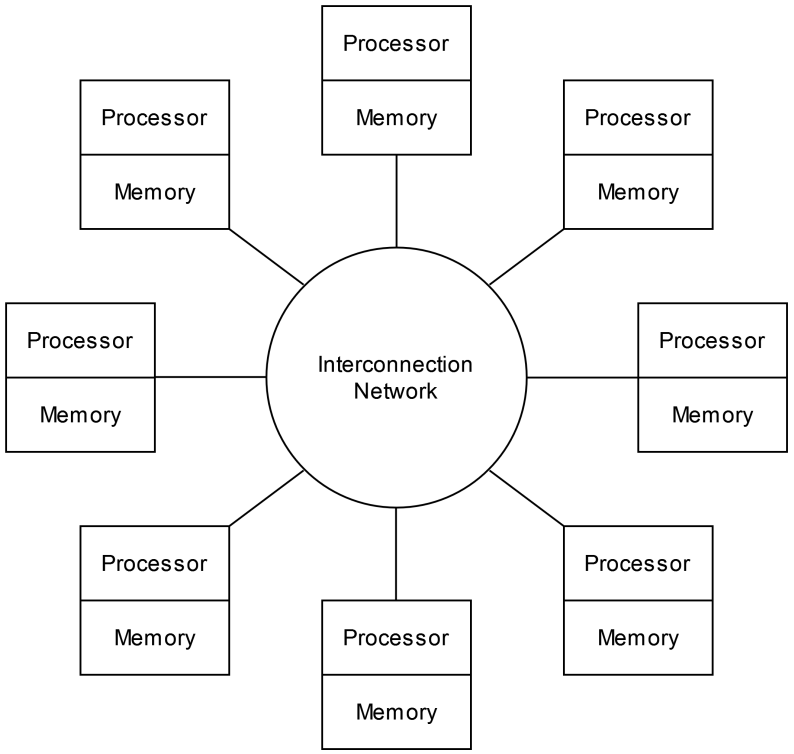


Figure 2.1: A representation of an interconnection network [1].

MPI [8] is an interface for sending messages between computing MPI ranks. This can be within a CPU or across clusters with several nodes. There were a lot of existing solutions during the 90s with different syntax and semantics, which made it so that experiments had to be re-implemented for different architectures[1]. The Message Passing Interface was created by the HPC community and vendors to make a standard for how programs interacted, which increased the portability of HPC applications across architectures. This interface is used for the proxy application in Chapter 3, and provides parallelism between the computer nodes[9].

2.3.1 Blocking MPI calls

MPI has the following blocking calls for sending and receiving data between compute nodes: MPI_Send([...]) and `MPI_Recv([...])`. These are blocking communication calls. When `MPI_Send` is executed, the program pauses until the corresponding `MPI_Recv` verifies that the source buffer safely can be modified.

2.3.2 Non-blocking MPI calls

The MPI calls: `MPI_Isend([...])` and `MPI_Irecv([...])` are non-blocking calls [8]. In contrast to the blocking MPI calls, these only initiate the message transmission, and do not guarantee that it is safe to modify the data. The program continues to run, and can potentially access unsafe buffers. One of the arguments of the `MPI_Isend` and `MPI_Irecv` is a `MPI_Request` handle, which can be used to verify that the message transmission is complete. This request handle is verified using a `MPI_Wait` or `MPI_Test` call [8].

2.4 Bulk Synchronous Parallel Programming Model

The BSP model[10] is a parallel computing model that aims to simplify the relation of hardware and software for parallel computing. For our purpose, the SWE domain is divided into sub-domains for each MPI rank. This model consists of three steps:

1. $Computing$ The data that are processed are located in the process' local memory and are computed without interference.

2. $Communicating$ the required data so that the next 1) step safely can compute. This step is usually performed by a router or network hardware, which offloads the communication so that the CPU can perform other tasks [11].

3. $Synchronize$ Ensures that all communication has been successfully transmitted and that all processes are ready to start again.

One cycle of these steps are called a super-step. The cost of each super-step is defined with Equation (2.20).

$$\max_{i=1}^{p}(w_i) + \max_{i=1}^{p}(h_i g) + l \tag{2.20}$$

where

- w_n = local computation cost of MPI_Rank.
- h_n = number of messages sent by the MPI_Rank.
- g = message delivery cost.
- l = synchronization cost.

2.5 LogGP Model

The LogGP model[12], an extension of the LogP model[13], is used to predict the performance of communication in parallel computing. Unlike the LogP model, which does not consider the gap time between longer messages (G), the LogGP model incorporates this factor. The five parameters are of the LogGP model are:

- L is the latency of the transmission. It represents the time it takes for a message to travel through the network. It is usually modeled after the worst-case scenario.
- o is the overhead. This represents the time the CPU is unavailable to perform other tasks. During this time, it is occupied with designating send and receive buffers and initializing communication. This overhead can be different for the sending and receiving process.
- g is the gap time between multiple messages sent in bursts. It represents the time between consecutive messages.
- G is the gap time between individual bytes in messages.
- P is the number of processes involved in the communication. For this paper, this is two, since we are using one-to-one communication for the border exchanges.

2.6 Border exchange

The finite difference solution of the Shallow Water Equations requires neighboring cells to compute the value for the next time-step. The proxy application we develop in this thesis use MPI to split the problem domain into a sub-domain for each MPI rank. The Figure 2.2 shows how each MPI rank receives data from the neighboring sub-domain though border exchange[14]. It illustrates how the MPI ranks send their colored edge, and it is received by the neighboring MPI rank. The MPI ranks send the data to the neighbors every Hth iteration, where H represents the thickness of each border in the halo. The example in Figure 2.2 has a thickness of one, so only a single row is sent for each communication.

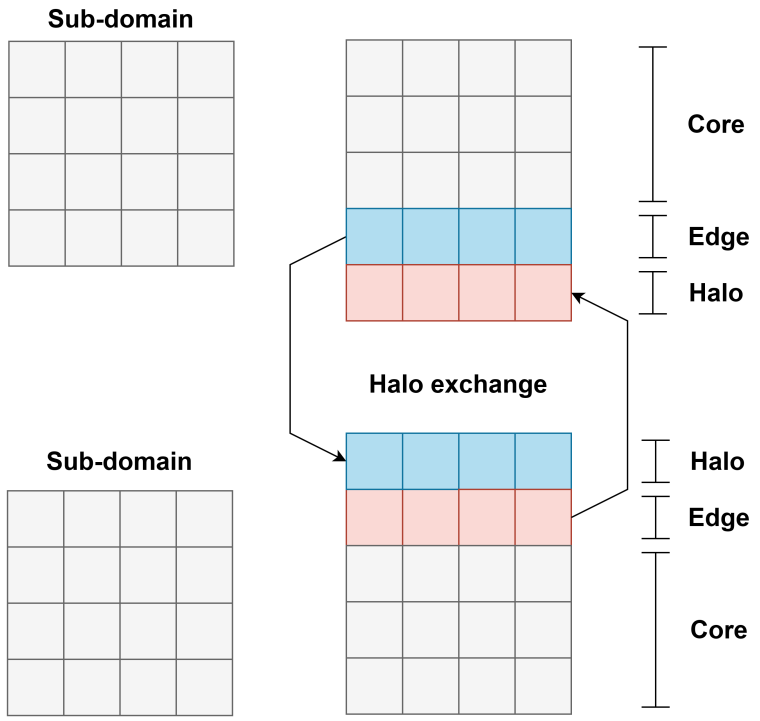


Figure 2.2: Two MPI ranks sending and receiving the necessary halo data to calculate their edges.

2.7 Overlap between communication and computation

By using non-blocking communication, mentioned in Section 2.3.2, it is efficient that the MPI rank can calculate unrelated data as the network hardware is completing the message transmission. Figure 2.3 shows how the CPU is only busy for a certain amount of time (overhead) before it is able to continue unrelated computation. By enabling the proxy application to have enough unrelated data to calculate, we can improve performance of the proxy application, and reduce its bandwidth requirements[15].

The CPU is then only busy with setting up the message transmissions before it is offloaded to the NIC, which used RDMA to store the data. This is necessary to achieve a benefit from the overlap and can be regarded as hardware parallelism. [11].

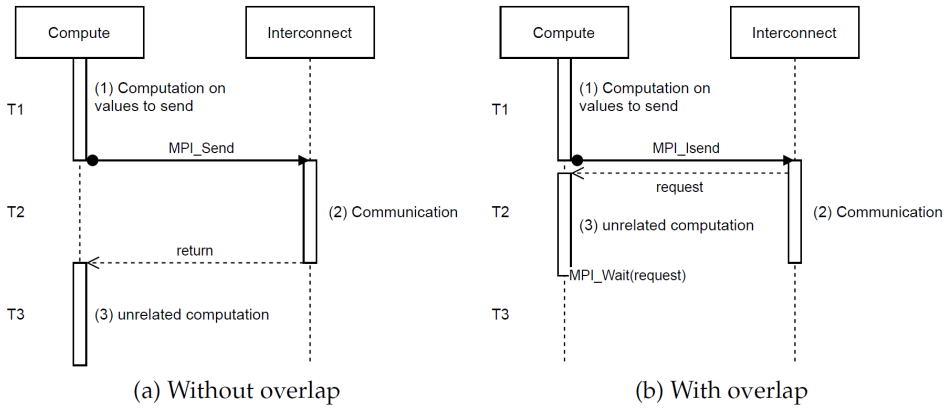


Figure 2.3: Communication scenario with and without overlap [2].

2.8 Reduce communication overhead

From the LogGP model[12] we know that the time it takes for messages of size k to travel from one processor to another is defined by Equation (2.21).

$$o_{send} + (k-1)G + L + o_{receive} \tag{2.21}$$

We also know that the processors are only busy during the overhead if they are able to overlap computation and communication. So if overlap is achieved, the time a processor is busy when sending or receiving a message is o_{send} or $o_{receive}$.

By reducing how often we initiate border exchanges, we can:

- Reduce the amount of times the overhead (o) is encountered for the receiving and sending processor.
- Reduce the delay of the latency (L) if we are not able to overlap communication and computations.
- Keep the same amount of gap time between bytes (G) in total, as we are sending the same amount of data in total.
- Reduce the gap time between messages (g), because we are sending fewer messages in total.

The trade-off of is that each MPI rank has to store more border exchange data, as seen in Figure 2.6, and these additional data also have to be computed. To have an overall performance benefit, it is necessary that the reduction of overhead(o) and gap time between messages (g) is bigger than the increase in computational load.

Chapter 3

Proxy Application

In this thesis we develop a proxy application solving the SWE equations using a FDM with Lax-Friedrichs Scheme. It starts out with an initial wave and because of the artificial viscosity of the Lax-Friedrichs Scheme the energy in the grid is reduced as the iterations accumulate. Along the edges of the domain, the energy is reflected back into the model using Neumann boundary conditions [6].

In this chapter we illustrate how the proxy application works. First we describe the initial condition, followed by different states of the application, and how they handle the varying border exchange thickness. The states have a different amount of workload for each super-step depending on the border exchange thickness.

As seen in Figure 2.2, the following parts of a sub-domain refers to:

- *Border* is the outer edge of the grid where data are received from neighboring MPI ranks. These are the data we receive from the border exchange, also referred to as halo data.
- *Edges* are the data which will be sent to neighboring MPI ranks. These are the data we are sending using border exchanges. The amount of data sent varies depending on the thickness of the border.
- *Core* refers to the data between the edges. The calculation of these data can be overlapped with the border exchange.

The application is designed to calculate a sub-domain of the Shallow Water Equations for a specified number of iterations. The flowchart in Figure 3.1 provides an overview of the states of the application.

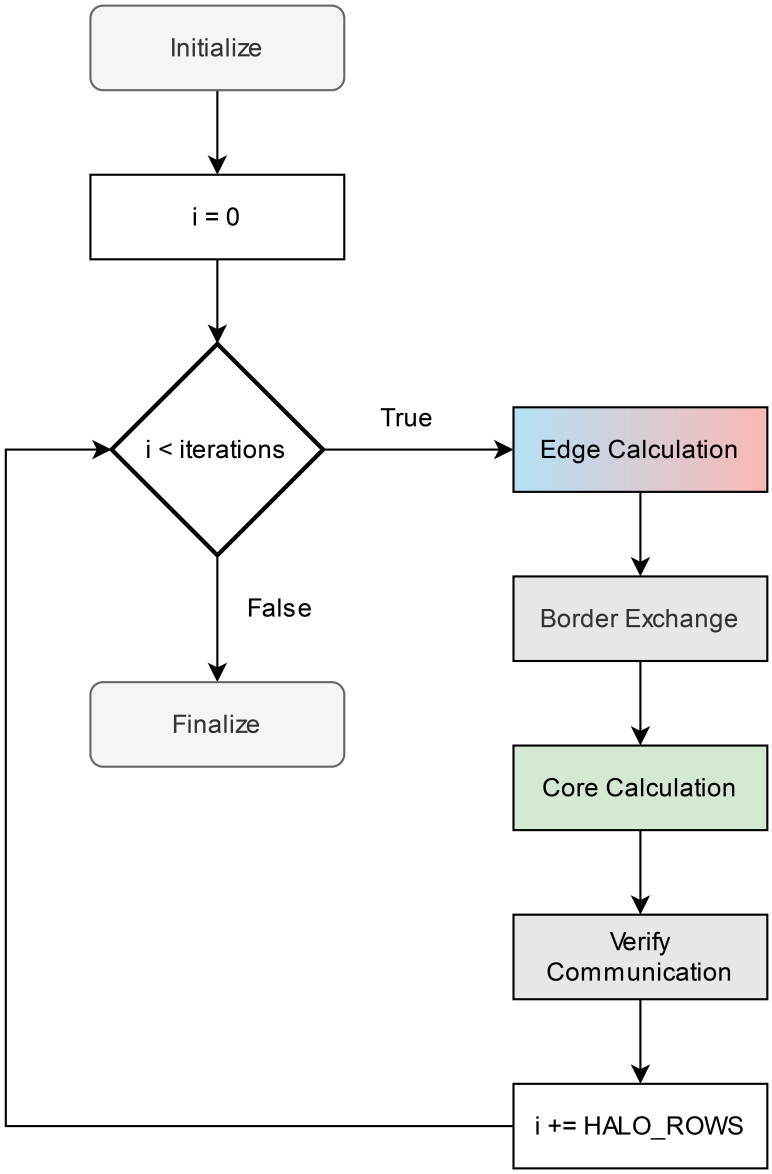


Figure 3.1: Flowchart representation of the proxy application.

3.1 Initial condition

The initial SWE domain is the cosine function in Equation (3.1). The frequency varies by the domain width. Figure 3.2 and Figure 3.3 is a visual verification that the border exchanges and stencil computations are correct.

$$\sum_{j=1}^{width} 10 + 3 \cdot cos(j \cdot \frac{\pi}{width/4}) + 1 \tag{3.1}$$

where

- 10 is the sea-floor.
- 3 is the amplitude.

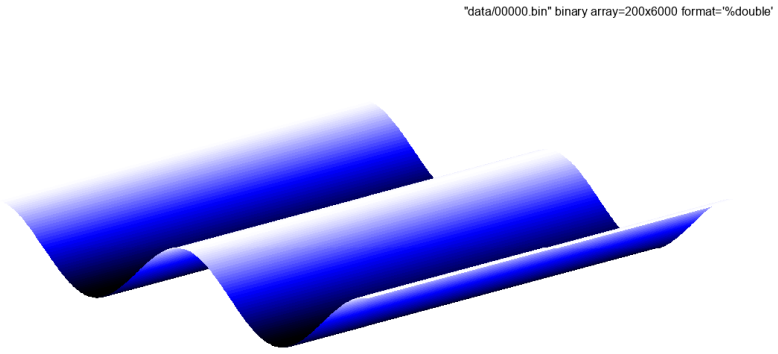
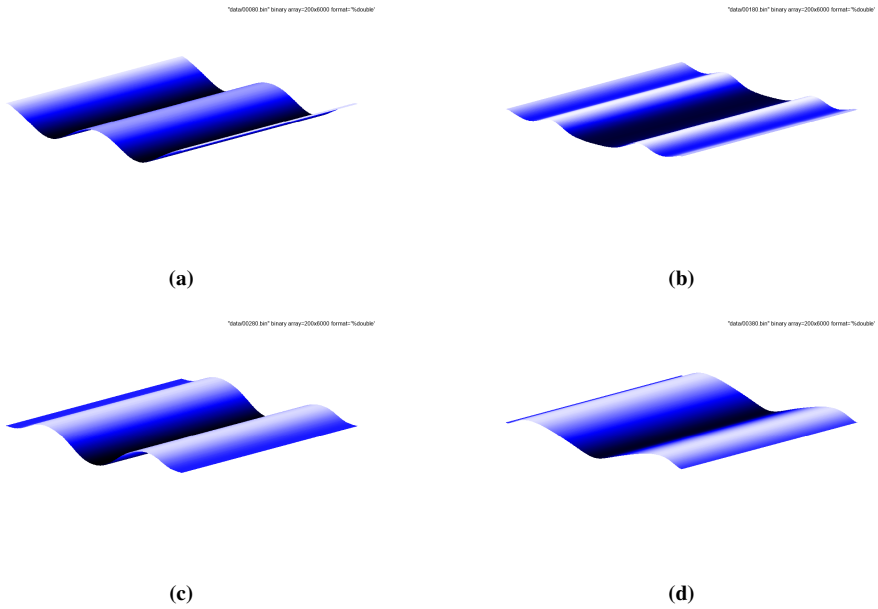


Figure 3.2: SWE wave starting position.



(a) **(b)**

(c) **(d)**

Figure 3.3: SWE wave after 80(a), 180(b), 280(c), and 380(d) iterations using 20 rows for the border exchange.

3.2 Application states

The proxy application completes several states for each loop, as seen in Figure 3.1. The following variables are used in the thesis:

- *Iterations* are the number of times the domain is calculated.
- *Time-step* is a reference to certain iteration within a super-step as seen in Figure 3.5, and Figure 3.6.
- *H* is the border exchange thickness, and can vary to reduce the frequency of border exchanges.
- *Super-steps* are the number of times we perform each state. This ratio is $\frac{iterations}{H}$.

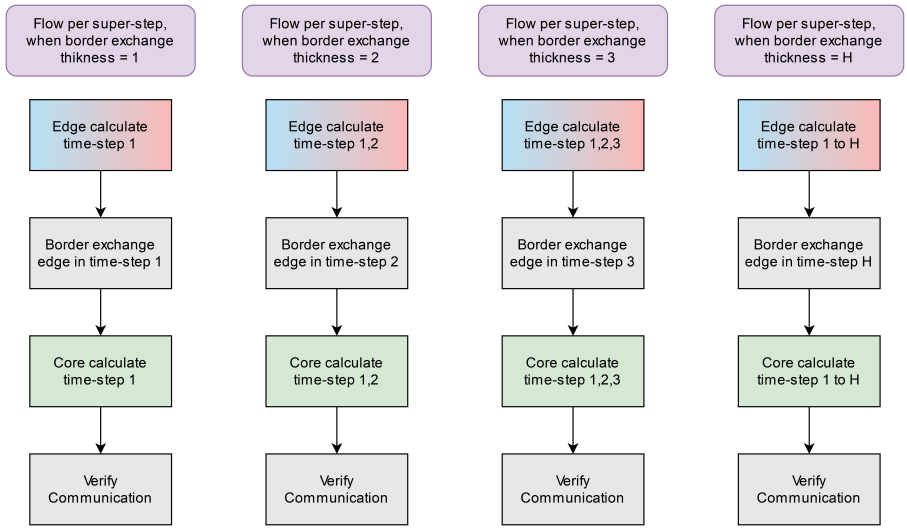


Figure 3.4: Varying workload for each application state depending on the border exchange thickness (H).

The workload of each of states in each super-step varies by the border exchange thickness. As seen in Figure 3.4 the edge calculation state is calculating H time-steps before the border exchange is initiated. Then the core calculation will finish the sub-domains that the edge calculation started on, before the communication finally is verified. Because one time-step corresponds to one iteration, we see that in flows where the super-step is H, we calculate H iterations of the sub-domain by only initiating one border exchange sequence. By only sending this one border exchange for larger H's we intend to reduce the overall communication overhead.

3.2.1 Edge calculation

The edge calculation state is responsible for computing the halo data which are received, and the edges so that it can be sent to neighboring MPI ranks. The halo data are used to calculate the edges for a specific number of iterations ahead, where the number of iterations ahead is equal to the thickness of the border rows. Figure 3.5 shows an example with a border thickness of 3.

Compared to a super-step that communicates every iteration, there will be an additional cost for some of the received data. Figure 3.5 visualizes how we calculate sub-domain, but the first and second time-step add 2 and 1 extra rows respectively for each edge. This is the additional cost of this implementation.

All the data are stored in an array for their time-step. Therefore, the application requires that there is enough memory to store the float values for all the time-step arrays. It is always $H + 1$ arrays, one for the initial data, and one for each time-step, as visualized in Figure 3.5, and Figure 3.6.

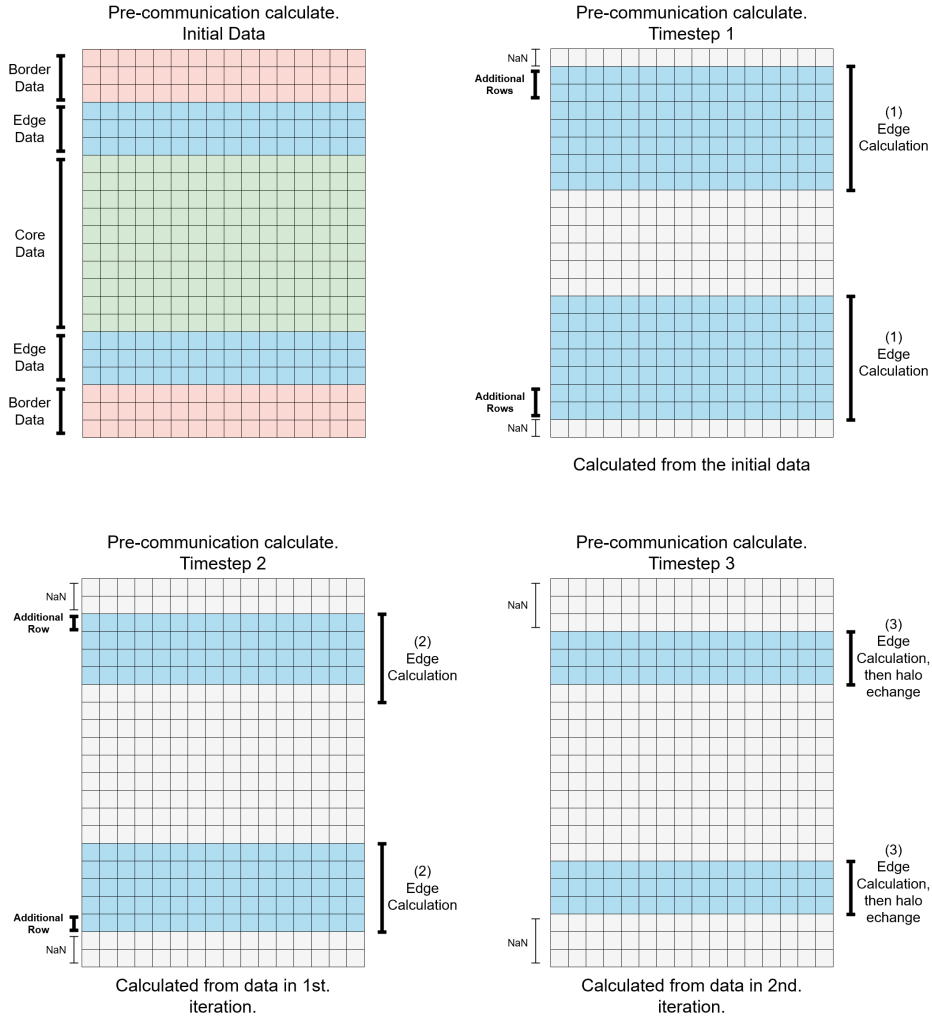


Figure 3.5: Visualizes how the edges are calculated 3 time-steps ahead.

3.2.2 Border exchange

After the two edges are calculated, they can be sent to the neighboring MPI ranks. As described in Section 2.7, the CPU will be busy with for a certain amount of time (overhead) before the transmission is offloaded to the network hardware. By varying the border exchange thickness, the frequency and amount of data we send change. In Figure 3.5 and Figure 3.6 we send three rows, but only every 3rd iteration.

3.2.3 Core calculation

While the halo rows are being calculated in Figure 3.5, the MPI ranks can continue calculating the core data while the borders exchanges are completed by the network hardware. Figure 3.6 shows how the green core data fields are calculated. They have all the necessary data from the previous time-step toestimate the grid cells of the next one. Combined with the blue fields which were calculated in the edge calculation state, we are left with a complete sub-domain for each time-step grid, seen in Figure 3.6.

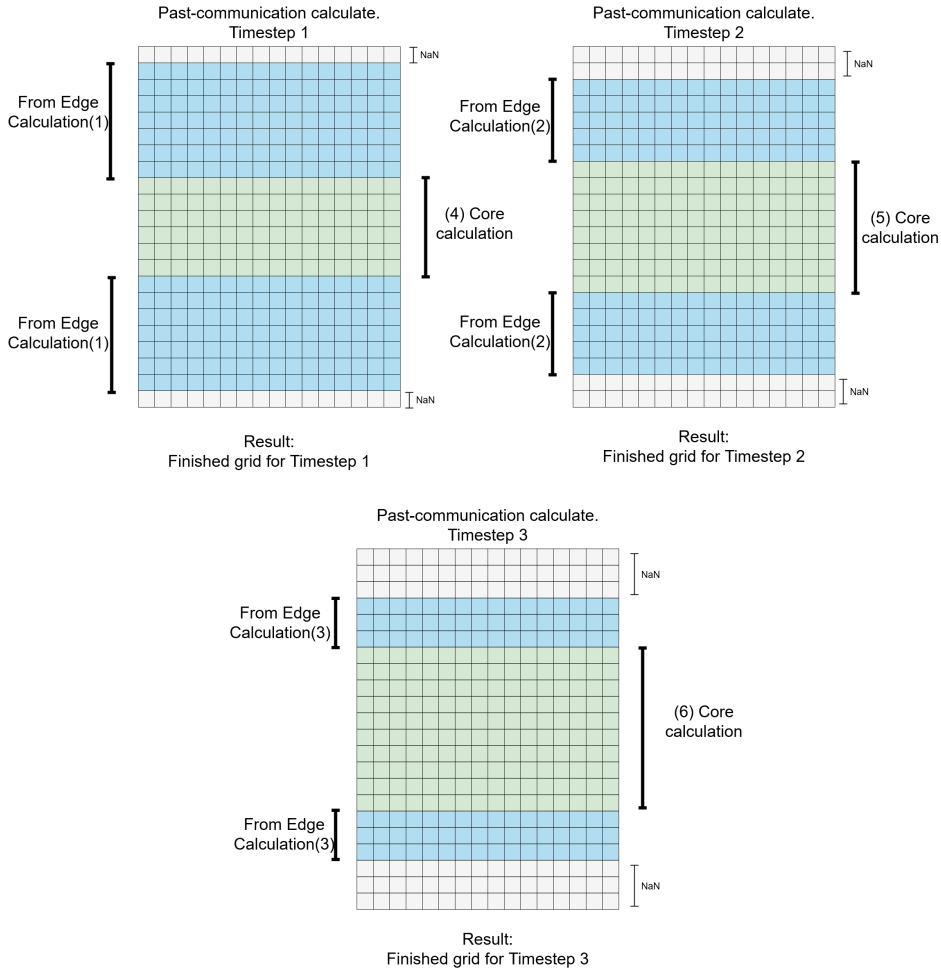


Figure 3.6: Demonstrates how the core calculation state fills up the time-step arrays until they contain a complete sub-domain.

3.2.4 Border exchange verification

After the last iteration of the core calculations state is complete, it is necessary to verify that the halo data from the border exchanges are safe to modify. When verified, the entire cycle repeats until the desired number of iterations are reached, as visualized in Figure 3.1. By verifying all the border exchanges, the application is inderectly synchronized, and all MPI ranks will start the next super-step at approximately the same time.

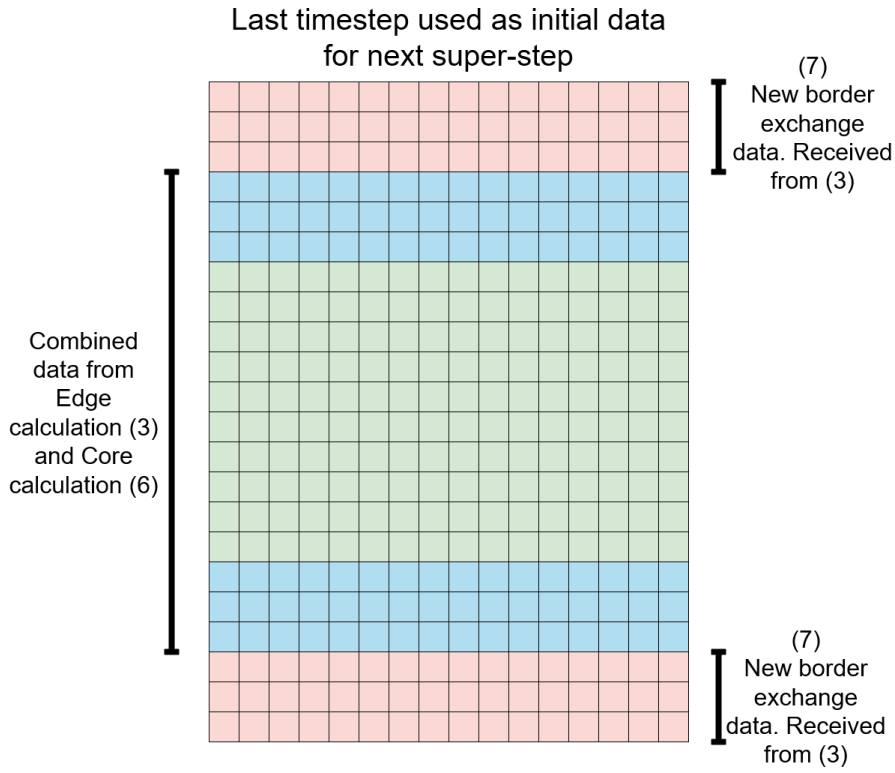


Figure 3.7: Shows how the data sent in Figure 3.5 and data combined in Figure 3.6 makes up the new initial data set for the next super-step.

In this chapter we have explained how our proxy application calculates the data for border exchanges H time-steps ahead before it initiates border exchanges, how those data are stored in arrays for their respective time-steps, and how the core calculation state calculates the remaining data for each time-step.

Chapter 4

Performance Model

In this chapter we look at the theory of the various computational and communicative costs of the application, how we gain a speedup of the proxy application, how we gather the data for the model, and how the data is inspected.

The proxy application is modeled using the costs of the various application states, which is done by gathering data for various sub-domain sizes and border exchange thickness combinations. Then we analyze how the cost of the various states affect the overall performance of the application.

4.1 Variable definitions

Table 4.1 describes the variables and what they represent throughout this thesis. All timings are recorded in units of seconds.

Variable	Description
s	Super-step as defined in Section 2.4.
R	The run-time for a desired number of iterations.
S_{cost}	The total cost of a super-step. As defined in the BSP model.
H	The number of border exchange rows sent and received for each super-step.
w	The width of sub-domain.
h	The height of the sub-domain.
E_p	The number of stencil points in the edge calculation state per super-step. Visualized in Figure 3.5.
E_t	The time it takes for the CPU to do the edge calculation state for each super-step.
E_a	The time used to calculate the additional stencil points of our implementation for each super-step.
C_p	The number of stencil points in the core calculation state per super-step. Shown in Figure 3.6.
C_t	The time it takes for the CPU to do the core calculation state for each super-step.
c_o	The overhead when calculating stencil points for the CPU. This is influenced by memory, caching and context-switching.
$c_{stencil}$	The speed of which the CPU is able to calculate stencil points per second.
B_s	This represents the time during which the CPU is busy initiating communication and creating buffers for the border exchange during each super-step.
B_r	The time the CPU is busy setting up the receiving end of the non-blocking communication for the border exchange during for each super-step.
B_v	The time the CPU is busy ensuring that the sent and received data can be accessed safely. This can be higher if overlap between communication and computations is not achieved.
B	$B_s + B_r + B_v$
B_b	The blocking communication time for the border exchanges.

Table 4.1: Overview of performance model terms.

4.2 Theory

From the LogGP model[12], there is an overhead cost associated with transferring data between MPI ranks. To reduce this overhead, we can reduce the frequency of communication. This is achieved by sending a larger amount of data, allowing each process to complete several iterations of the sub-domain for each border exchange. To accomplish this, we increase the number of halo rows exchanged during each border exchange. This comes with additional rows added to the sub-domain size, as explained in Section 4.2.1.3.

By utilizing the BSP model described in Section 2.4 as a starting point, our focus is on evaluating how different border exchange thicknesses affect the cost of each super-step. The total number of super-steps is determined by Equation (4.1).

$$s(H) = \frac{iterations}{H} \tag{4.1}$$

If we aim to complete 2520 iterations of our SWE domain, the corresponding number of super-steps (s) for each border exchange thickness is presented in Table 4.2. This table contains 10 super-steps, and the selected iteration count 2520 is the LCM of 1 to 10. In this thesis we are using up to 20 border exchange rows, however the LCM for that number approximately $20 \cdot 10^6$, so we have accepted inaccuracy on thicknesses greater than 10 instead.

Border Thickness, H	1	2	3	4	5	6	7	8	9	10
s(H)	2520	1260	840	630	504	420	360	315	280	252

Table 4.2: The ratio between iterations, border exchange thicknesses and super-steps.

With the BSP Equation (2.20) in mind, we can rewrite the following sections to account for varying border exchange thicknesses (H). It is important to note that the variables in the list below do not take the domain proportions into account.

- $w_n => E_t + C_t$
- $h_n => 3 \cdot 2$ Three messages for each SWE Equation and two for each edge.
- $g => B_s + B_r$
- $l => B_v$ Verifying border exchanges will indirectly synchronize all the MPI ranks.

`MPI_Wait` ensures that buffers can be accessed safely, as explained in Section 2.3.2. This is only possible when every process has completed its entire super-step. The formula for cost of a super-step with halo thickness(H) is given by Equation 4.2.

$$S_{cost}(H) = E_t(H) + C_t(H) + B(H) \tag{4.2}$$

To obtain the total run-time (R) of a desired number of iterations, the formula should be multiplied by the number of super-steps.

$$R = S_{cost}(H) \cdot s(H) \tag{4.3}$$

4.2.1 Costs

4.2.1.1 Edge calculation cost

The edge calculation step (E_t) in Section 3 varies by the width and border thickness. The grid height of the edge calculations will increase by 2 with each iteration time-step. This leads to Equation 4.4 for the edge calculation stencil points for each super-step. It is important to note that the product 3 and 2 represents three Shallow Water Equations and the number of edges for each process, respectively.

$$E_p(w, H) = 3 \sum_{k=1}^{H} w \cdot 2[H + 2(H - k)] \tag{4.4}$$

The time it takes to calculate the stencil points depends on CPU speed, the overhead of task switching which involves saving registers and memory maps and arithmetic operations involved in the varying loop sizes represented by the parameter k in Equation 4.4. This gives us Equation (4.5).

$$E_t(w, H) = \frac{E_p(w, H)}{c_{stencil} + c_o} \tag{4.5}$$

4.2.1.2 Core calculation cost

Core calculation speed is impacted by the width, height and border exchange thickness for the super-step. The various time-steps within the super-steps will have various grid sizes, as visualized in Section 3.6. The amount of core calculation stencil points for each super-step is shown in Equation 4.6. The factor 2 represents the 2 edges for each process in the horizontal topology.

$$C_p(w, h, H) = \sum_{k=1}^{H} w \cdot [h - 2H - 2(H - k)] \tag{4.6}$$

By factoring in the CPU overhead and computational speed, we get Equation (4.7)

$$C_t(w, h, H) = \frac{C_p(w, h, H)}{c_{stencil} + c_o} \tag{4.7}$$

4.2.1.3 Additional Stencil Points

When $H > 1$ there are additional rows added to the edge calculations on top and bottom of the sub-domain, these are the additional costs of this proxy application implementation, as seen in Figure 3.5. The additional cost for pre-calculating the data for a super-steps with border thickness H, is seen in Equation (4.8). It is worth mentioning that the expression $2(H - k)$ is the same that is subtracted from the core calculation in Equation (4.6).

$$E_a(w, H) = 3 \sum_{k=1}^{H} \frac{w \cdot 2(H - k)}{c_{stencil} + c_o} \tag{4.8}$$

4.2.1.4 Border exchange costs

From Section 2.8 we know that sending and receiving a message when overlap is achieved only makes the CPU busy during the overhead. Therefore, we can set the cost of sending the border exchanges as seen in Equation (4.9), the cost of receiving the border exchanges as seen in Equation (4.10), and the cost of verifying that the border exchange is complete, defined as B_v.

$$B_s(H) = 6 * o_{send}(H) \tag{4.9}$$

$$B_r(H) = 6 * o_{receive}(H) \tag{4.10}$$

The total cost for the border exchange per super-step is then defined by Equation (4.11).

$$B(H) = B_s(G) + B_r(H) + B_v(H) \tag{4.11}$$

By conducting tests for each possible width, and border thickness, we are able to include the variations of the overhead, and capture parameters which might not overlap during the border exchange.

4.2.1.5 Blocking communication cost

If the communication finished concurrently, `MPI_Wait` should not have to wait for longer than it takes to verify that the buffers can be accessed safely. To ensure that it does not wait for communication to finish, the MPI ranks require enough unrelated work to do in the core calculation state. Therefore, it is interesting to time how long a message uses from process A to B. To accomplish this, we have timed the blocking communication pattern for each width and border thickness combination (B_b).

4.2.2 Achieving speedup

To achieve an overall speedup of the application, the additional costs of the edge calculation, Equation (4.8), has to be smaller than the reduced cost of the border exchange seen in Equation (4.12). The cost of the border exchanges and additional stencil points is compared to the original border exchange cost when $H = 1$.

$$OverallCostChange(H) = s(H) * [B(H) + E_a(w, H)] - B(1) * s(1) \tag{4.12}$$

4.3 Collecting data

To collect data, tests are conducted on the clusters to extract the calculation, and communication time for each of the involved MPI ranks. Table 4.3 shows the values used in the

Variable	Value
WIDTHS	50-1000
HEIGHTS	50-1000
HALO_ROWS	1-20
INTERVAL	50

Table 4.3: List of the defined variables used in the data gathering.

data gathering. The *INTERVAL* is the increment of the *WIDTHS* and *HEIGHTS* between tests.

The variables $c_{stencil} + c_o$ from Equation (4.5) and Equation (4.7) are not directly collected in the data gathering process. We are gathering data for each combination of width, height and border thickness, so the CPU overhead and computational speed are included in the measurements.

4.3.1 Border exchanges

The border exchanges are performed by sending each of the SWE equation to the neighboring processors' subdomain, as seen in Algorithm 1.

Function `BorderExchangeSend`$(width, H)$
  $MPI\_Isend(SWE\_mass, width * H)$
  $MPI\_Isend(SWE\_velocity\_X, width * H)$
  $MPI\_Isend(SWE\_velocity\_Y, width * H)$
End Function
Function `BorderExchangeRecv`$(width, H)$
  $MPI\_Irecv(SWE\_mass, width * H)$
  $MPI\_Irecv(SWE\_velocity\_X, width * H)$
  $MPI\_Irecv(SWE\_velocity\_Y, width * H)$
End Function

Algorithm 1: Pseudo-code for border exchanges.

4.3.2 Blocking data gathering

Algorithm 2 illustrates how we collect the total blocking communication time (B_b), mentioned in Section 4.2.1.5, for each of the domain and border thickness combinations. The code block is timing the entire communication pattern with the neighboring MPI ranks.

```
Function BlockingCommunicationTime()
    for width in WIDTHS do
        for H in HALO_ROWS do
            MPI_Barrier()
            start_time ⟵ time.now
            BorderExchangeSend(width, H)
            BorderExchangeRecv(width, H)
            MPI_Wait()
            timing(width, H) ⟵ time.now − start_time
        end
    end
End Function
```

Algorithm 2: Pseudo-code for timing the blocking communication time of the border exchanges.

4.3.3 Non-blocking data gathering

Algorithm 3 measures the border exchange costs for send (B_s), receive (B_r) and wait (B_v). Each timer is responsible for measuring the duration of the border exchange parameters associated with send, receive and wait. We are timing the MPI function calls for each combination of width and border exchange thicknesses, and are not timing the LogGP parameters individually. The resulting data provide insights into how long the CPU is busy sending, receiving and verifying the border exchanges for all the different widths and border thickness combinations.

```
Function BorderExchangeCost()
    for width in WIDTHS do
        for H in HALO_ROWS do
            MPI_Barrier()
            send_time ⟵ time.now
            BorderExchangeSend(width, H)
            send_timing(width, H) ⟵ time.now − send_time
            sleep()
            recv_time ⟵ time.now
            BorderExchangeRecv(width, H)
            send_timing(width, H) ⟵ time.now − recv_time
            sleep()
            recv_time ⟵ time.now
            MPI_Wait()
            timing(width, H) ⟵ time.now − start_time
        end
    end
End Function
```

Algorithm 3: Pseudo-code for timing the border exchange costs.

4.3.4 Edge data gathering

Algorithm 4 illustrates how edge calculation (E_t) timings for all the widths and border exchange combinations are measured.

```
Function EdgeCalculation():
    for width in WIDTHS do
        for H in HALO_ROWS do
            start_time ⟵ time.now
            CalculateEdge(width, H)
            timing(width, H) ⟵ time.now − start_time
        end
    end
    return timing
End Function
```

Algorithm 4: Pseudo-code for retrieving the edge calculation time.

4.3.5 Core data gathering

Algorithm 5 shows how the core calculation (C_t) timings for all the widths, height and border exchange combinations are measured.

```
Function CoreCalculation():
    for width in WIDTHS do
        for height in HEIGHTS do
            for H in HALO_ROWS do
                start_time ⟵ time.now
                CalculateCore(width, height, H)
                timing(width, H) ⟵ time.now − start_time
            end
        end
        return timing
    end
End Function
```

Algorithm 5: Pseudo-code for retrieving the core calculation time.

4.4 Analyzing data

The output files of the data collection process are analyzed by the performance model, which is written in Python[16] programming language. Then the data is processed and the averages, standard deviations and medians of the measurements, from Section 4.3, are extracted using NumPy[17]. Finally, we visualize the results using the Matplotlib library[18].

During the processing of the data, we sort the data by size, and the top 0.5% and lowest 0.5% of the measurements are removed. This has been implemented to eliminate extreme values that influence the average of the data-sets. To estimate the total running time, we use Equation (4.3).

$$S_{cost}(H) \;=\; E_t(w,h) \;+\; C_t(w,h,H) \;+\; B_s(w,H) \;+\; B_r(w,H) \;+\; B_v(w,H)$$

Within the $E_t(w,h)$ we have the additional stencil computation cost for the super-step as mentioned in Section 4.2.2, and the cost of the border exchanges and verification is present in the $B_s(w,H) + B_r(w,H) + B_v(w,H)$ variables.

In contrast to the BSP model, this performance model does not use the maximum values of each super-step. Instead, it utilizes the averaged values of the data-sets to estimate the total cost of the super-steps.

Algorithm 6 outlines the requirement to ensure that $C_t(w,h,H)$ can overlap with the border exchanges, as discussed in Section 2.7. It demonstrates that 10 percent margin on top of the communication time ensures that the core calculation has enough data to process. If the function returns False, then the result is marked as not suitable for a performance benefit.

Function `VerifyEnoughComputation` (C_t, B_b, w, h, H)
  if $C_t(w,h,H) > B_b(w,h) * 1.10$ **then**
    return True
  else
    return False
  end
End Function

Algorithm 6: Pseudo-code verifying enough calculation time.

4.4.1 Result tables

Appendix F, Appendix G and Appendix H shows how the output of the performance model are displayed. There are a total of 8000 rows of result tables for each cluster. These can be found in the attached files.

4.4.2 Manual data inspection

To ensure the validity of the data used in the performance model, we perform a manual inspection. We inspect the box-plots of each of the collected timings to look for inconsistencies that could impact the prediction. For the following examples we have used the data gathered for the Idun cluster with a sub-domain size of 150w and 900h. The graphs for the other sub-domain sizes can be found in Appendix C, Appendix D and Appendix E.

4.4.2.1 Border exchange costs

In Figure 4.1, the border exchange send cost graph, the data is similarly grouped across the varying thicknesses. Some of the data have substantially bigger boxes, but it is consistent across the whole data set. It is important to note that the messages with a halo thickness(H) are H times larger than those with halo thickness 1. We see similar grouped data in the receive border exchange graph and verify border exchange graph in Figure 4.2 and Figure 4.3.

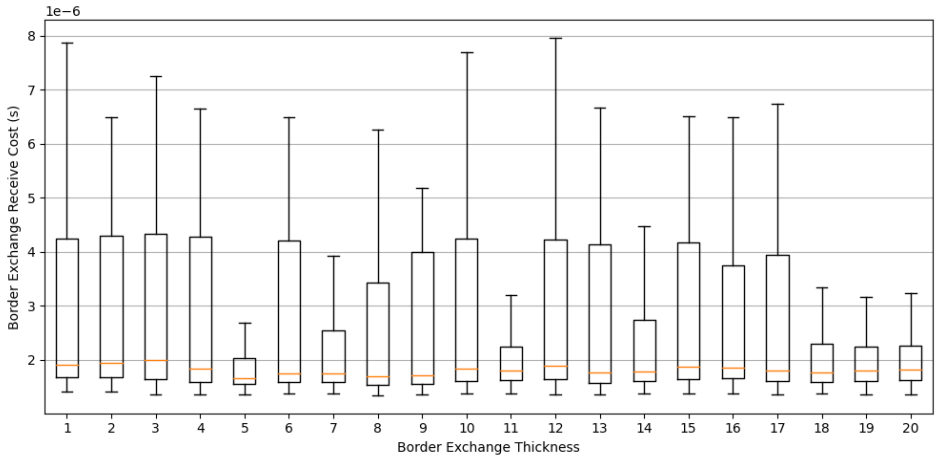


Figure 4.1: Border exchange send cost for each super-step when sending messages with width 150 for Idun.

When considering the total border exchange cost as shown in Figure 4.4, the variations for the border exchange costs are low. We also see that the total cost increases a little as the border exchange thickness increases. The increase is not a factor from 1 to 20, which

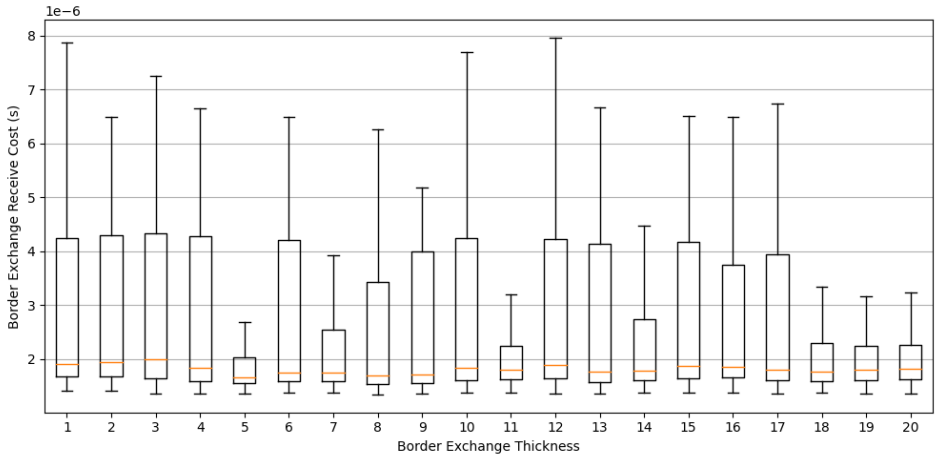


Figure 4.2: Border exchange receive cost for each super-step when receiving messages with width 150 for Idun.

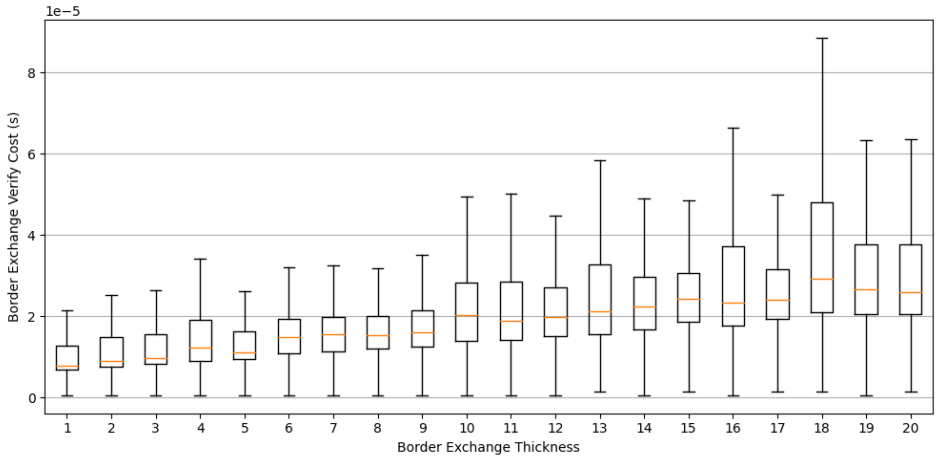


Figure 4.3: Border exchange verify cost for each super-step when sending messages with width 150 for Idun.

indicates that sending bigger messages less often can reduce the overhead of the border exchanges.
It is important to note that the outliers have been disabled for all the border exchange graphs.

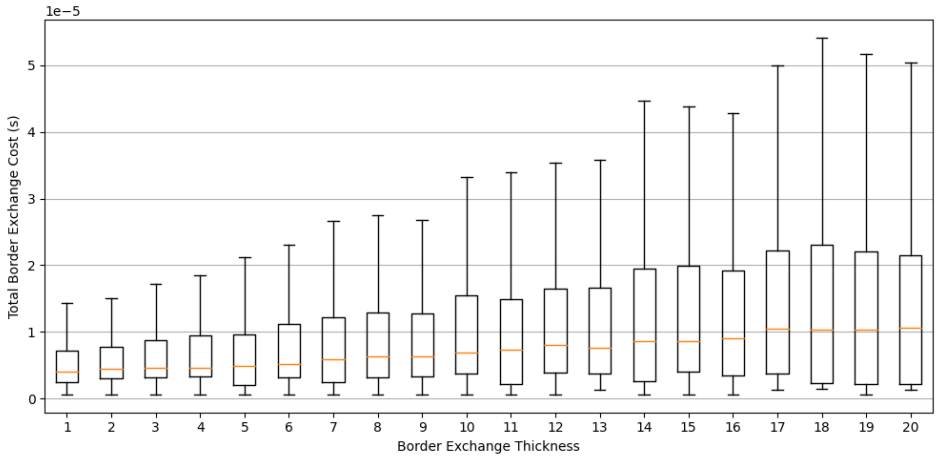


Figure 4.4: Total border exchange cost for each super-step when sending messages with width 150 for Idun.

4.4.2.2 Core calculation cost

Figure 4.5 represents the time spent on calculating the core stencil points using Equation (4.7). The data has low variation in the box plots, and are consistent across thicknesses.

4.4.2.3 Edge calculation cost

Figure 4.6 shows the edge calculation data, and how they also demonstrate low variance in the boxes.

4.4.2.4 Blocking communication time

Figure 4.7 presents an example of the blocking communication time. As shown in Algorithm 5 it is necessary to have sufficient data to calculate while the border exchanges are completed by the NIC. Comparing the blocking communication time to the core calculation times in Figure 4.5, we observe that the communication times are below the core calculation times. This verifies that the core calculation has enough data to compute while the communication is finished.

4.4.3 Test predicted run-times

The prediction model calculates the expected run-time for 2520 iterations. In certain cases, the predicted run-time for specific sub-domains may be very low. To address this, the number of iterations is multiplied until the expected run-time falls within the range of 10-20 seconds. The selected multipliers for the the proxy application tests can be found in Section 5.3. The non multiplied running time for each of sub-domains for 2520 iterations can be found in Appendix F, Appendix G, and Appendix H.

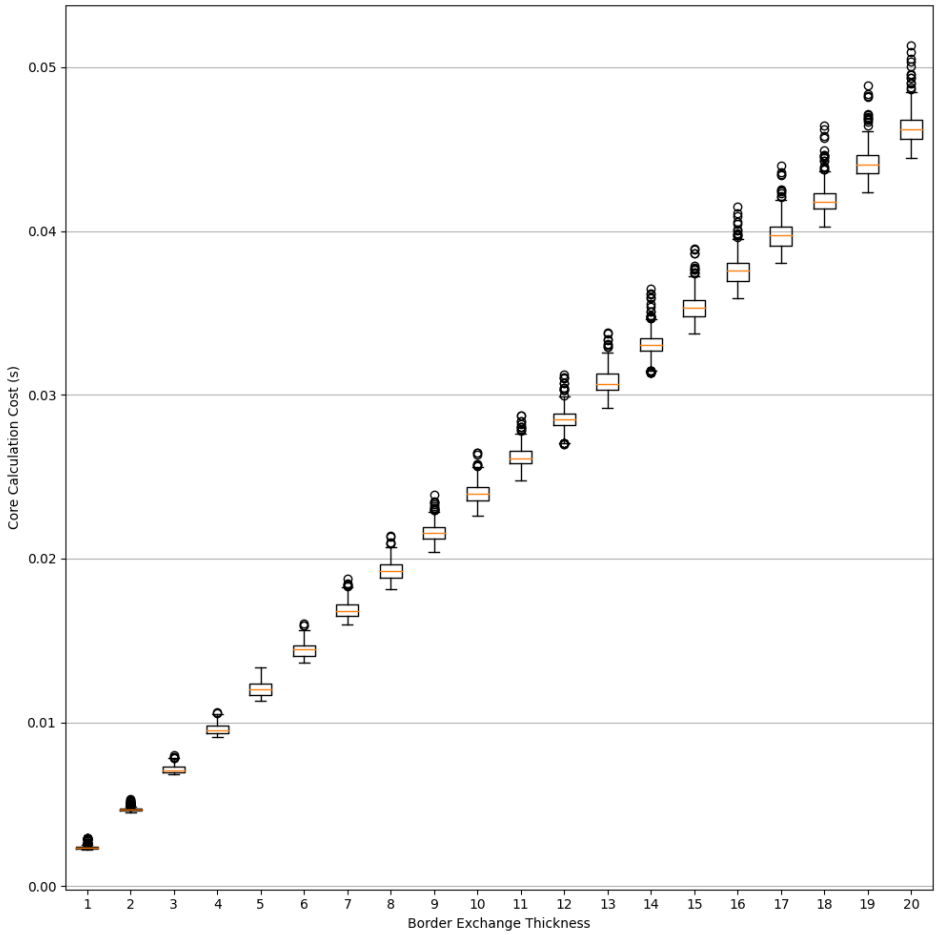


Figure 4.5: Core calculation cost for each super-step for a 150x900 sub-domain for Idun.

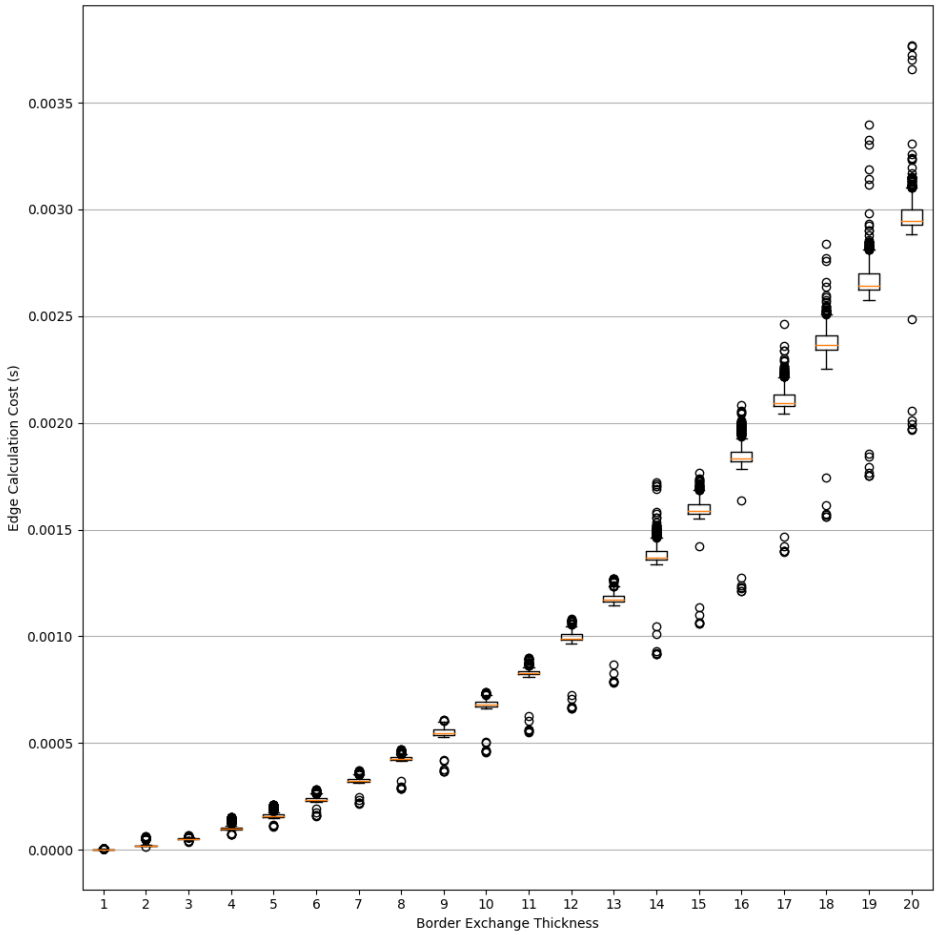


Figure 4.6: Edge calculation cost for each super-step for a 150w sub-domain for Idun.

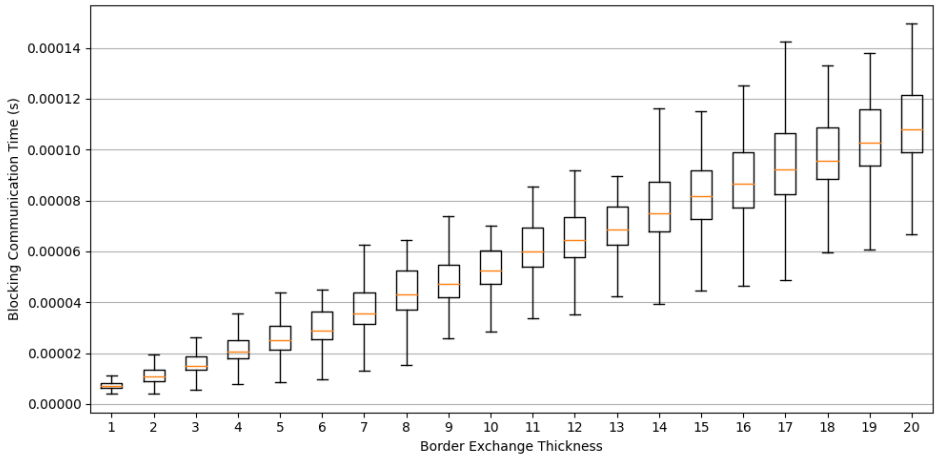


Figure 4.7: Blocking communication time when sending messages of width 150 for Idun.

Chapter 5

Experimental Setup

In this chapter, we look at the setup of the HPC clusters and their hardware, followed by an overview of the nodes and MPI ranks involved in the performance models' data gathering, and the configurations of the proxy application tests. Each of the conducted tests are done for their domain with varying border thicknesses from 1 to 20.

5.1 Clusters & configurations

5.1.1 Idun cluster

Idun is hosted at the Norwegian University of Science and Technology. It combines the resources of the individual departments resources to create a cluster for rapid testing and HPC prototyping. The hardware specification used for the computation are listed in Table 5.1[19].

Property	Value
Compiler version	GCC 10.2.0
Compiler Flags	-O3 -lm
MPI Version	Intel(R) MPI Library for Linux* OS, Version 2019 Update 9 Build 20200923
Processor	Intel Xeon Gold 6348
Node type	Dell PE730
Processors/node	2
Cores/node	56
Memory/node	256 GB

Table 5.1: Hardware and attributes used for the computations on Idun.

5.1.2 Fram cluster

Fram is hosted at the Arctic University of Norway[20]. The hardware specification used for the computation are listed in Table 5.2.

Property	Value
Compiler version	GCC 10.2.0
Compiler Flags	-O3 -lm
MPI Version	Intel(R) MPI Library for Linux* OS, Version 2019 Update 9 Build 20200923
Processor	Intel E5-2683v4 2.1 GHz
System	Lenovo NeXtScale nx360
Processors/node	2
Cores/node	32
Memory/node	64 GB

Table 5.2: Hardware and attributes used for the computations on Fram.

5.1.3 Betzy cluster

Betzy is located at NTNU in Trondheim and is the biggest supercomputer used for testing in this project[20]. The hardware specification used for the computation are listed in Table 5.3.

Property	Value
Compiler version	GCC 10.2.0
Compiler Flags	-O3 -lm
MPI Version	Intel(R) MPI Library for Linux* OS, Version 2019 Update 9 Build 20200923
Processor	AMD® Epyc™ 7742 2.25GHz
System	BullSequana XH2000
Processors/node	2
Cores/node	128
Memory/node	256 GB

Table 5.3: Hardware and attributes used for the computations on Betzy.

5.2 Data collection

Table 5.5, Table 5.6 and Table 5.7 lists the combination of nodes and processors used for the data gathering and performance model. Table 5.4 visualizes that the number of conducted tests varies on the number of processors (P).

# Test Property	Formula
CORE_TESTS	$x \cdot P$
EDGE_TESTS	$x \cdot P$
CPU_LOCK_TESTS	$x \cdot P$
BLOCK_COM_TESTS	$x \cdot P$

Table 5.4: Tests depend on the number of MPI ranks (P).

5.2.1 Idun data collection

# Node(s)	# MPI Ranks	Core tests	Edge tests	Lock tests	B. Com tests
2	112	$5 \cdot 112$	$5 \cdot 112$	$20 \cdot 112$	$10 \cdot 112$

Table 5.5: Data collection for each width, height and border exchange thickness combinations for Idun.

5.2.2 Fram data collection

# Node(s)	# MPI Ranks	Core tests	Edge tests	Lock tests	B. Com tests
2	64	$5 \cdot 64$	$5 \cdot 64$	$20 \cdot 64$	$10 \cdot 64$

Table 5.6: Data collection for each width, height and border exchange thickness combinations for Fram.

5.2.3 Betzy data collection

# Node(s)	# MPI Ranks	Core tests	Edge tests	Lock tests	B. Com tests
4	512	$2 \cdot 512$	$2 \cdot 512$	$10 \cdot 512$	$10 \cdot 512$

Table 5.7: Data collection for each width, height and border exchange thickness combinations for Betzy.

5.3 Proxy application configuration

Table 5.8, Table 5.9 and Table 5.10 shows the selected domains, combination of nodes, tests and processors for running the proxy application. The performance models' analysis graphs for the respective domains can be found in appendix, referenced in column eight. It is worth noting that the domains are rectangular, and resemble a line. This is not helpfull for practical problems, but should not impact the analysis of how the varying border exchange thickness impacts the overall performance of the proxy application.

5.3.1 Idun proxy application tests

Width	Height	$\frac{\text{Height}}{\text{MPI\_Ranks}}$	# Iterations	# Node(s)	# MPI Ranks	# Tests	Appendix
100	2800	50	$2520 \cdot 50$	1	56	25	C.1
100	5600	50	$2520 \cdot 50$	2	112	25	C.1
100	11200	50	$2520 \cdot 50$	4	224	22	C.1
300	2800	50	$2520 \cdot 20$	1	56	25	C.2
300	5600	50	$2520 \cdot 20$	2	112	25	C.2
300	11200	50	$2520 \cdot 20$	4	224	24	C.2
150	50400	900	$2520 \cdot 2$	1	56	25	C.3
150	100800	900	$2520 \cdot 2$	2	112	24	C.3
150	201600	900	$2520 \cdot 2$	4	224	24	C.3

Table 5.8: List of parameters used to run the proxy application for Idun.

5.3.2 Fram proxy application tests

Width	Height	$\frac{\text{Height}}{\text{MPI\_Ranks}}$	# Iterations	# Node(s)	# MPI Ranks	# Tests	Appendix
50	11200	350	$2520 \cdot 10$	1	32	25	D.1
50	22400	350	$2520 \cdot 10$	2	64	25	D.1
50	89600	350	$2520 \cdot 10$	8	256	8	D.1
300	32000	1000	2520	1	32	25	D.2
300	64000	1000	2520	2	64	25	D.2
300	256000	1000	2520	8	256	10	D.2

Table 5.9: List of parameters used to run the proxy application for Fram.

5.3.3 Betzy proxy application tests

Width	Height	$\frac{\text{Height}}{\text{MPI\_Ranks}}$	# Iterations	# Node(s)	# MPI Ranks	# Tests	Appendix
300	256000	500	2520	4	512	10	E.1

Table 5.10: List of parameters used to run the proxy application for Betzy.

Chapter 6

Results & Discussion

In this chapter we will compare the predictions of the performance model, in Chapter 4, to the actual tests of the proxy application, explained in Chapter 3. We have observed that we are able to achieve a speedup for certain halo thicknesses, particularly $H = 2$. The modeling of the core and edge calculations have good accuracy, while they sometimes deviate from the actual run-times.

Throughout all the tests, we have noticed that our performance models' prediction of communication times are optimistic, primarily capturing the lower outliers rather than the actual time spent waiting on communication. This makes sense as the model has been created using the average values of the communication, and does not take data variation and interference of the interconnect into account.

6.1 Graph details

For the run-time plots each run reports data for each individual process. They have the following properties:

- *Runtime*: The total run-time.
- *Wait time*: The time spent on waiting and verifying communication. This indirectly includes synchronization of the MPI ranks.
- *Core calculation time*: The time spent on calculating the core stencil points.
- *Edge calculation time*: The time spent on calculating edge stencil points.

When plotting the data each output of a sub-domain is mapped by the slowest run-time, as this is how fast the entire domain would be finished. The ID of the slowest process is then used to get the time for the core, edge and wait times used for the comparison plots.

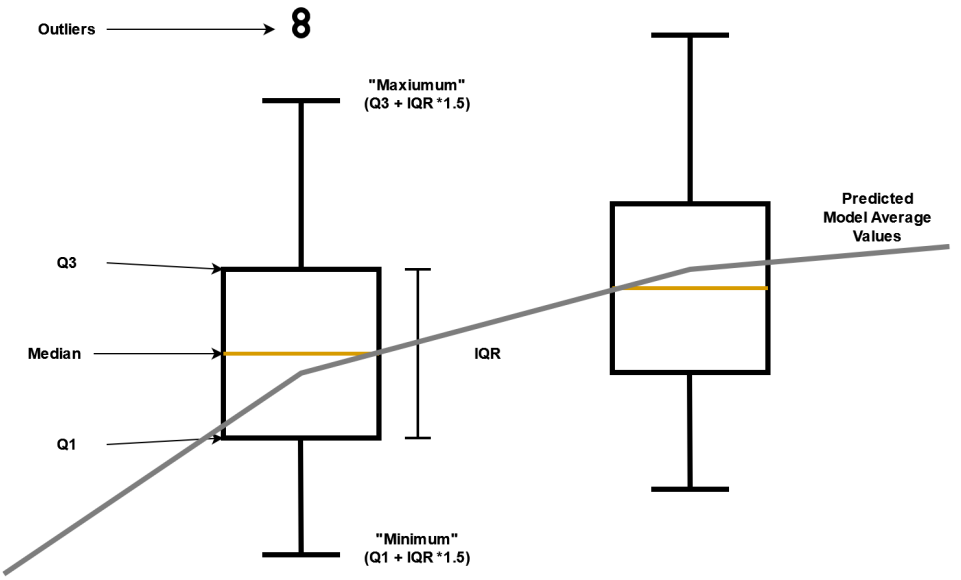


Figure 6.1: Graph plot explanation.

Figure 6.1 demonstrates how the results are presented.

- $Q1$ is the 25th percentile of the box.
- $Q3$ is the 75th percentile of the box.
- IQR (Interquartile range).
- $Median$ is the median of the values.
- $Lineplot$ is the average of the prediction value explained in Chapter 4. This has the same color as the run-time values it is predicting.
- $Percent$: In the result graphs there will be a percentage on the x-axis labels. This is how the median value of the border exchange thicknesses compare to the $H = 1$ value.

6.2 Idun results

For the Idun results we have bad predictions of the total run-time for the sub-domains width a height of 50. The performance model is predicting speedup for most of the border exchange thicknesses, but we are only able to achieve it for $H = 2$. In contrast, the sub-domain of 150w, and 900h is accurately modeled. While we do have issues with modeling the communication, and that the results deviate from the prediction, we see that some of the actual computations give a performance benefit.

The number of tests conducted for each of the domains and node combinations are found in Table 5.8.

6.2.1 Sub-domain of 100w, 50h

Figure 6.2 illustrates that the total running time increased with an increase in the border exchange thickness. The core calculation data aligns well with the prediction. The edge calculation line falls slightly below the median of the run-times. Overall, the core and edge prediction aligns well with measured values.

The communication prediction is following the gray box plots for the first 2 border exchange thicknesses, but do not accurately represent the actual wait time in the computation for the other thicknesses. As seen in Figure 6.2, they follow the outliers of the prediction when $H > 2$ We see that the wait timing is reduced in the range of $H = 2$ to $H = 8$, which can indicate the there is a reduction in the communication overhead. The overall runtime prediction predicts speedup for all the border exchange thicknesses. We only see a speedup of 2.4% and 1.3% for $H = 2$ and $H = 3$, respectively. The actual run-time is slowly increasing as the thicknesses increase.

The discrepancy for the prediction and actual run-time can be attributed to the optimistic communication estimate, and that the core and edge results vary as the border

exchanges' thickness changes. Similar observation are seen in the tests for 1 node and 4 nodes in Appendix A.1.

H	1	2	3	4	5	6	7	8	9	10
Run %	100.0	97.6	98.7	100.1	101.6	103.4	105.2	107.4	109.4	111.4
Pred %	100.0	91.2	87.2	85.7	82.4	81.0	80.5	81.0	79.5	79.6
H	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Run %	114.2	116.5	119.1	123.3	128.1	133.5	140.4	148.8	153.2	158.9
Pred %	80.1	80.3	80.4	81.3	81.2	84.4	86.0	91.1	93.7	99.0

Table 6.1: Actual runtime compared to prediction. Combined from Figure 6.2 and Result table, Appendix F.1.

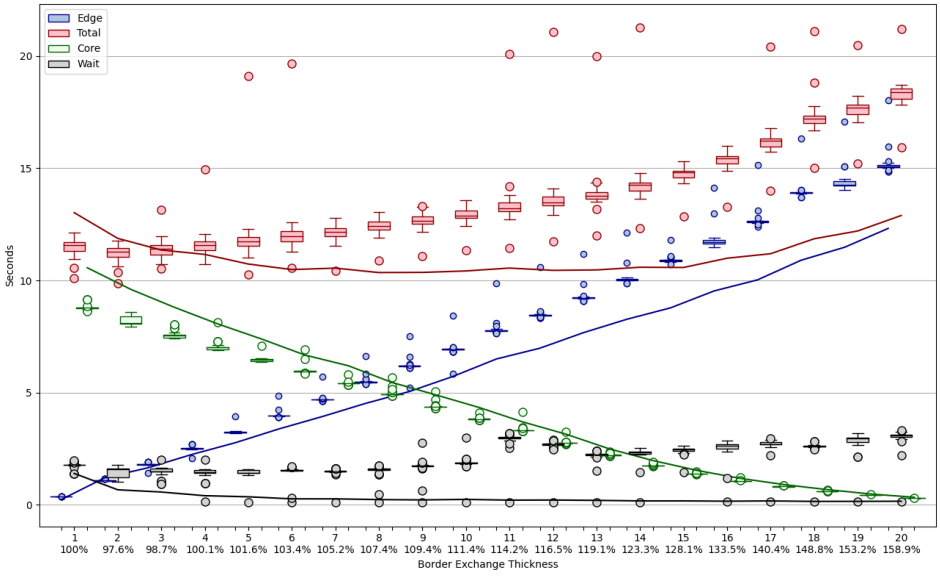


Figure 6.2: Results for Idun 100w, 50h sub-domain for 2 nodes, and 112 MPI ranks.

6.2.2 Sub-domain of 300w ,50h

In this sub-domain size, increasing the border exchange thickness do not provide any benefits in terms of reducing the network overhead. We have similar observations for the edge, core and wait comparisons between the prediction and the actual run-times, as we have seen in Section 6.2.1. The same behavior is observed in the tests for 1 and 4 nodes seen in Appendix A.2, this further supports the previous observation that the communication prediction is optimistic.

H	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Run %	100.0	103.0	105.4	142.2	112.4	117.2	119.9	121.8	124.8	128.6
Pred %	100.00	93.3	92.7	92.4	90.8	91.0	90.4	91.0	90.6	90.9
H	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Run %	131.5	135.1	137.4	140.6	146.3	151.3	157.1	170.9	179.3	188.4
Pred %	91.3	91.8	92.3	93.4	95.0	97.8	101.6	108.5	109.5	115.0

Table 6.2: Percentage of run-time compared to prediction. Combined from Figure 6.3 and Result Table, Appendix F.2.

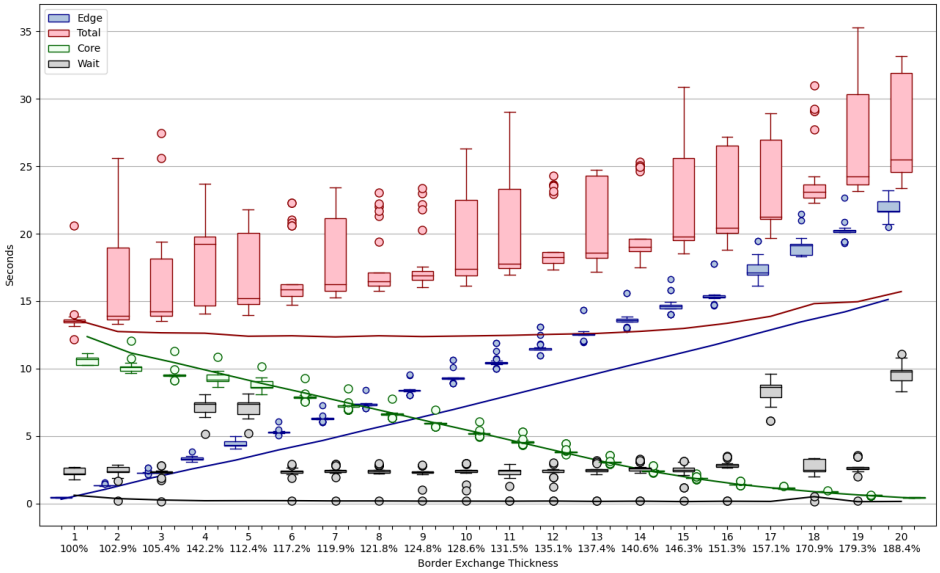


Figure 6.3: Results for Idun 300w, 50h sub-domain for 2 nodes, and 112 MPI ranks.

6.2.3 Sub-domain of 150w, 900h

As previously seen this sub-domain size continues the pattern where the communication model is following the outliers. Table F.3 shows a speedup of 3.5% in the actual run-time compared to a predicted value of 2.1%. In this sub-domain the edge and core values are accurately modeled, but the communication timing is not accurate, as it is very close to the $y = 0$ axis.

There is a fluctuating pattern observed in the results for every odd number, which stem from the variation of the core calculation times. We do see these fluctuations and performance benefits in the tests for 1 and 4 nodes too, presented in Appendix C.3, where a speedup of 4.5% and 5.4% is achieved for border exchange thickness 2.

H	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Run %	100.0	96.5	101.1	101.6	97.8	101.8	98.2	101.5	97.6	101.8
Pred %	100.0	98.9	100.7	101.6	102.3	102.5	103.0	103.1	102.9	103.1
H	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Run %	102.5	98.9	101.4	97.5	102.2	102.3	98.8	102.4	99.0	102.2
Pred %	102.8	102.9	102.9	103.1	103.1	103.1	103.2	103.1	103.1	103.1

Table 6.3: Actual runtime compared to prediction. Combined from Figure 6.4 and Result table, Appendix F.3.

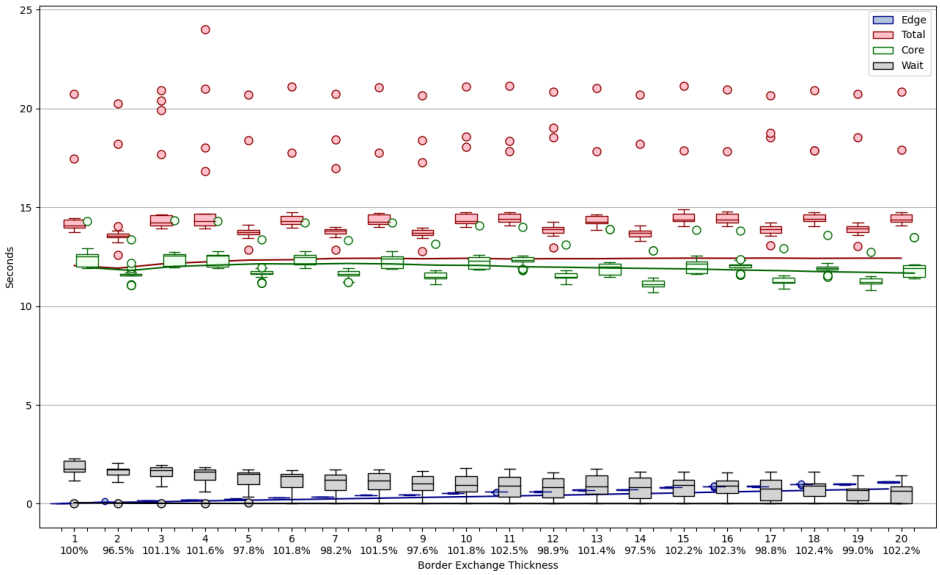


Figure 6.4: Results for Idun 150w, 900h sub-domain for 2 nodes, and 112 MPI ranks.

6.3 Fram results

For the Fram results we perceive a speedup for the sub-domains when $H = 2$, and that the core and edge prediction accurately models the actual values, while the communication model continues to fall short. The number of tests conducted for each of the domains and node combinations are found in Table 5.9.

6.3.1 Sub-domain of 50w, 350h

Table 6.4 visualizes the predicted run-times in percentage compared to the actual run-time in percentages. We see that it is only a run-time speedup for $H = 2$, while the predicted run-time gives speedup for all border exchange thicknesses. We observe in the table and Figure 6.5 that the predicted values are increasing slowly from $H = 2$ to $H = 20$, as the run-time also does. The predicted value of $H = 1$ is high, which is the reason of the optimistic prediction percentages.

We also observe that the core calculation follows the curvature of the measured values, but tends to be slightly higher and aligned with some outliers instead of the IQR of the box plots. The edge calculation is also modeling the actual compute-time, but is below the IQR of the boxes. As we have seen in the other sub-domain tests, the communication is following the outliers of the actual timings, instead of being in the IQR. The median of the gray communicating boxes are reduced from approximately 1.7 to 1.5 as the thicknesses increase, which indicates a reduction of the verification overhead.

Additional tests for 1 and 4 nodes, shown in Appendix B.1, also demonstrate a speedup of 2.1% and 2.7% for $H = 2$, and demonstrate the same discrepancies between the core, edge and communication predictions.

H	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Run %	100.0	98.1	99.7	100.1	100.9	101.8	102.7	103.2	103.7	104.2
Pred %	100.0	95.5	95.8	96.3	96.6	97.3	97.3	97.7	97.9	98.1
H	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Run %	104.4	103.1	105.0	103.7	105.9	105.8	104.4	104.7	106.4	105.3
Pred %	98.3	98.1	98.1	98.2	98.1	98.2	98.1	98.1	98.1	98.2

Table 6.4: Actual runtime compared to prediction. Combined from Figure 6.5 and Result table, Appendix G.1.

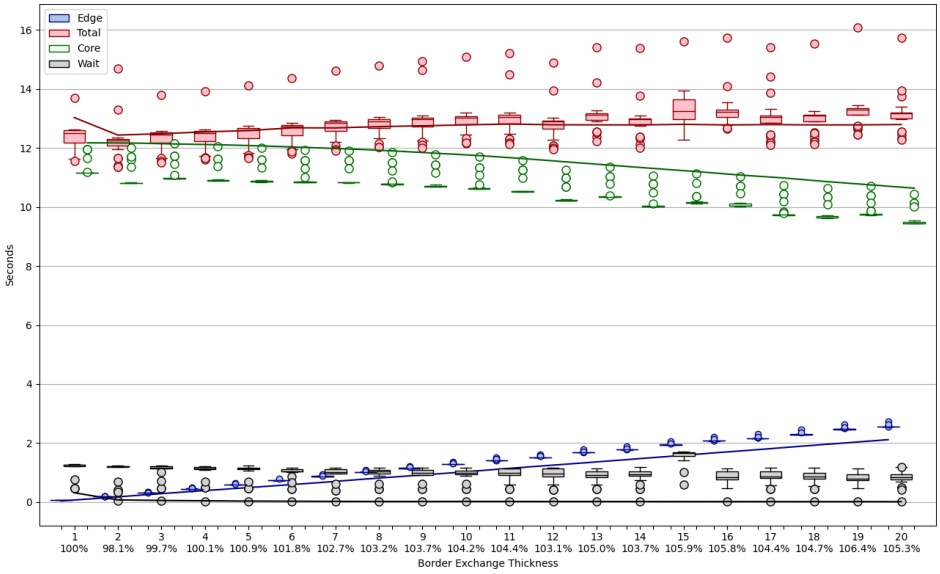


Figure 6.5: Results for Fram 50w, 350h sub-domain for 2 nodes, and 64 MPI ranks.

6.3.2 Sub-domain of 300w, 1000h

Table 6.5 demonstrates that we have a speedup of 0.3% compared to the predicted 0.8%. The speedup is not convincing, and we see in Appendix B.2 that the tests for 1 and 4 nodes show no speedup for $H = 2$, but the Q1 and Q3 for $H = 2$ are grouped closer than $H = 1$, so there should be some runs where speedup is achieved. In Figure 6.6 we see that the core and edge calculations are modeled accurately, and that our prediction is following the actual run-time, but the actual computation is about a percent higher when $H >= 5$.

Here, it is also evident that the communication wait time is being reduced as the border exchange thickness increases. Indicating that there might be a benefit of increased overlap between the computation and communication for some of the thicknesses.

H	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Run %	100.0	99.7	100.1	100.1	100.8	100.2	100.5	100.4	100.2	100.5
Pred %	100.0	99.2	99.7	99.9	100.0	100.1	100.2	100.2	100.3	100.3
H	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Run %	100.9	100.5	100.8	100.9	100.9	101.3	101.6	101.9	101.4	101.4
Pred %	100.3	100.4	100.5	100.5	100.4	100.4	100.3	100.4	100.5	100.4

Table 6.5: Actual run-time compared to prediction. Combined from Figure 6.6 and Result table, Appendix G.2.

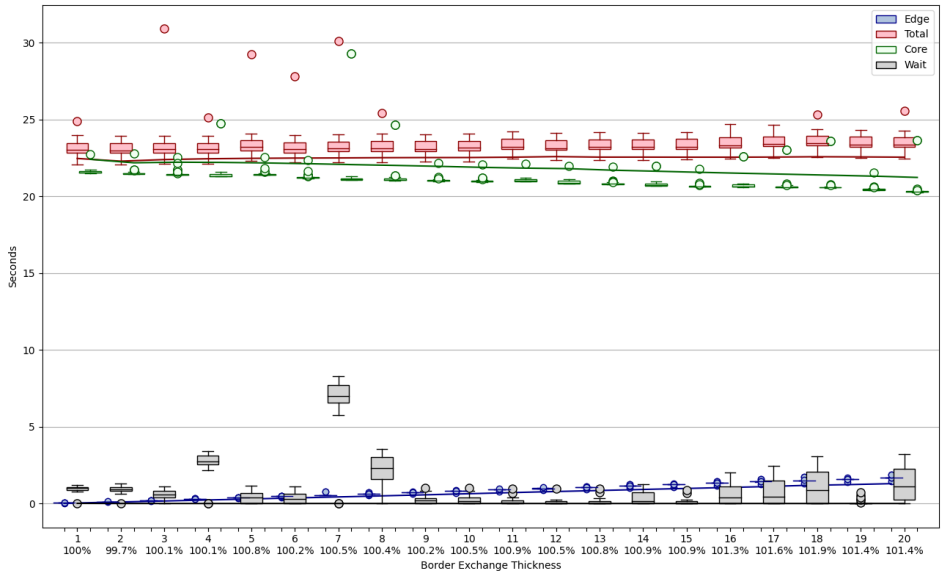


Figure 6.6: Results for Fram 300w, 1000h sub-domain for 2 nodes, and 64 MPI ranks.

6.4 Betzy results

6.4.1 Sub-domain of 350w, 500h

In the Betzy tests, as indicated in Table 6.6, a speedup is predicted for all thicknesses where $H > 1$. The prediction is optimistic, and we see that the actual speedup is only 1.6%.

Upon closer examination, we find that the individual models for the edge and core calculations are accurate. As seen in the other clusters, the communication prediction, represented by the gray line, consistently underestimates the actual communication time and does not accurately reflect its behavior. The communication wait time is reduced for $H = 2$ through $H = 10$, and shows that the MPI ranks are spending less time waiting, when it is calculating that many time steps ahead between each border exchange. While this might affect the results, we see from the overall run-time that it does not increase and decrease at the same H, and is therefore not the sole reason for the reduced overall run-time.

This discrepancy between the predicted and actual values suggests that the performance model needs further refinements in the communication predictions to provide more accurate predictions for the Betzy cluster.

H	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Run %	100.0	99.7	99.3	99.0	99.3	99.5	99.4	99.2	99.2	98.9
Pred %	100.0	97.0	97.1	96.6	96.1	95.7	95.2	94.8	93.9	93.4
H	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Run %	99.3	98.9	98.8	98.7	98.7	98.7	98.5	98.5	98.2	98.4
Pred %	93.0	92.5	91.9	91.4	91.2	90.9	90.6	90.4	89.7	89.2

Table 6.6: Actual runtime compared to prediction. Combined from Figure 6.4.1 and Result table, Appendix H.1.

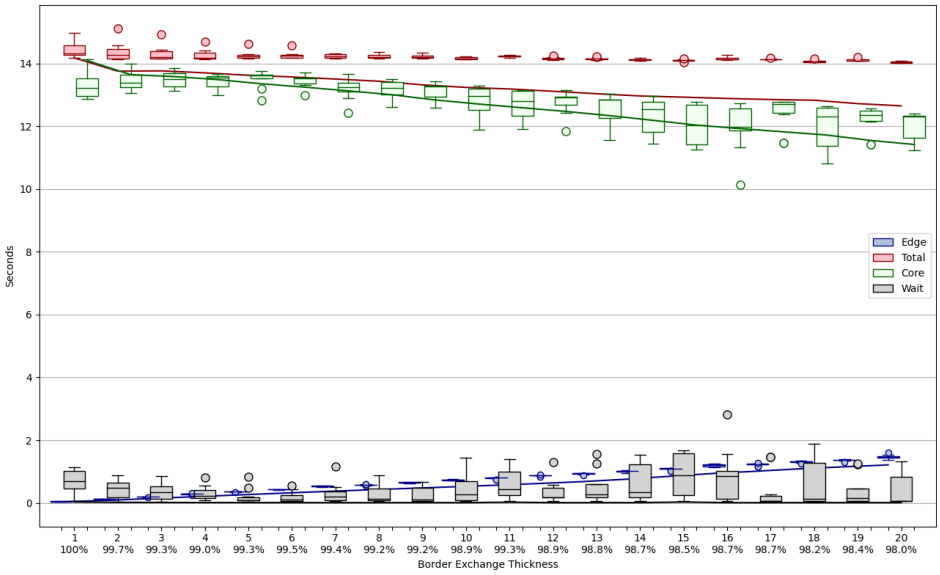


Figure 6.7: Results for Betzy 350w, 500h sub-domain for 4 nodes, and 512 MPI ranks.

6.5 Sources of error

The performance model relies on average values to predict the run-time. It is important to note that there is variance in the data-sets, as seen in Appendix C, Appendix D, and Appendix E. The average values used in the performance model can found in the result tables, presented in Appendix F, Appendix G and Appendix H.

It is worth considering that the communication pattern is influenced by the number of nodes participating in application. The current performance model utilizes data collected from two nodes for Idun and Fram, and four for Betzy. When scaling to a different number of nodes for the proxy application, the limit of the interconnect bandwidth can change the communication time for the border exchanges, which makes it so that the amount of unrelated work we need in the core calculation state to achieve overlap may change.

Because we use a multiple of 2520 iterations for each run, we have border exchange thicknesses greater than 10 where 2520 is not evenly dividable. This can make the program run an additional super-step until the total iteration count is over the multiple of 2520, and for a worst case scenario ($H = 17$) a program can run 2533 iterations before terminating.

Chapter 7

Conclusion

In this thesis, we demonstrate how we can reduce the total amount of communication overhead by computing more stencil points and exchange data between MPI rank less often. We develop a proxy application that solves the Shallow Water Equations using a finite difference method and create a performance model of the proxy application by separating its procedures into costs per super-step.

The results demonstrate that we were able to achieve an overall speedup by reducing the frequency of data exchanges to every 2nd. iteration. However, it became evident that while the performance modeling of the stencil computations are characterizing the proxy application, the modeling of the border exchange costs are optimistic and based on the least costly and least probable communication time between MPI ranks.

In the results of Fram we see a reduction in the border exchange verification time for all border exchange thicknesses (H), and for $H = 2$ to $H = 17$ for the sub-domain sizes of 50w, 350h and 300w, 1000h. For the Idun sub-domain size of 150w, 900h, all the border exchange thicknesses has a reduction in the border exchange verification time. The sub-domain tested on Betzy also shows a reduction in the border exchange cost for $H = 2$ to $H = 10$. These observations indicate that there is less overhead in the higher border exchange verification thicknesses.

Furthermore, as anticipated in the performance model described in Chapter 4, we observed an increase in computational cost for the core and edge computations because of additional stencil points for each sub-domain. This aligns with our estimations and supports that we can employ the BSP model in this way to estimate the cost.

In this thesis, we have learned that it is possible to mask the communication overhead by trading additional stencil computations against fewer, but larger message transmissions. We have also experienced that modeling the communication pattern after average results provide a model that takes the best case scenario into account, and does not estimate the interference present in an interconnected network.

7.1 Future work

The results obtained for our sub-domain demonstrate that there can be benefits of reducing the frequency of initiated communications. In order to improve the accuracy of the performance model, the following topics could be investigated further:

- $Cartesian\ Topology$: In order to increase the usability and scalability of the proxy application and performance model, one could implement a Cartesian topology. The current horizontal topology is less scalable, and the domains are hard to keep quadratic for problems where many MPI ranks are employed.
- $Communication\ Model$: The current solution to modelling the communication as described in Section 4.2.1.4 only gives an optimistic timing for the border exchange patterns employed in the proxy application. A more in-depth and perhaps theoretical solution based on benchmarks of the network and the individual LogGP parameters can increase the accuracy.
- $Tune\ Performance\ Model$: The current implementation by employing the average for the entire prediction model provided a general estimation of the modeled values. However, as discussed in the results, Chapter 6, a few of the tests have core and edge results which differ slightly from the actual model.

Bibliography

[1] M. J. Quinn, *Parallell Programming: in C with MPI and OpenMP*. Mc. Graw Hill, first ed., 2004.

[2] S. Mailund Svendsen, "In search of lost time: A deep dive in overlapping computation and communication in memory bound mpi applications," p. 19, 2021.

[3] E. J. Kubatko, S. Bunya, C. Dawson, J. J. Westerink, and C. Mirabito, "A performance comparison of continuous and discontinuous finite element shallow water models," *Journal of Scientific Computing*, vol. 40, pp. 315–339, 2009.

[4] D. J. K. C. Kafle J., Bagale L.P, "Numerical solution of parabolic partial differential equation by using finite difference method," vol. 6, December 2020.

[5] B. N. Biswas, C. S., M. S.P, and S. PAL, "A discussion on euler method: A review," July 2013.

[6] A. H.-D. Cheng and D. T. Cheng, "Heritage and early history of the boundary element method," *Engineering Analysis with Boundary Elements*, vol. 29, no. 3, pp. 268–302, 2005.

[7] M. J. Quinn, *Parallell Programming: in C with MPI and OpenMP*. Mc. Graw Hill, first ed., 2004.

[8] A. Lastovetsky, V. Rychkov, and M. O'Flynn, "Mpi forum." `https://www.mpi-forum.org/docs/`. Accessed: 11.12.2022.

[9] P. Pacheco, *An Introduction to Parallel Programming*. Elsevier, 2011.

[10] L. G. Valiant, "A bridging model for parallel computation," vol. 33, no. 28, pp. 103–111, 1990.

[11] V. Cardellini, A. Fanfarillo, and S. Filippone, "Overlapping communication with computation in mpi applications," february 2016.

[12] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: Incorporating long messages into the logp model," 1997.

[13] D. Culler, R. Karp, A. Patterson, D. andSahay, K. E. Schauser, E. Santos, R. Subramonion, and T. von Eicken, "Logp: Towards a realistic model of parallel computation," 1993.

[14] F. B. Kjolstad and M. Snir, "Ghost cell pattern," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, pp. 1–9, 2010. Accessed: 11.12.2022.

[15] V. Subotic, J. Carlos Sancho, J. Labarta, and M. Valero, "A simulation framework to automatically analyze the communication-computation overlap in scientific applications," 2010.

[16] P. S. Foundation, "Python 3.10 documentation." `https://docs.python.org/3.10/`. Accessed: 17.5.2023.

[17] NumPy, "Numpy." `https://numpy.org/`. Accessed: 23.5.2023.

[18] plotlib, "Matplotlib." `https://matplotlib.org/`. Accessed: 23.5.2023.

[19] M. Själander, M. Jahre, G. Tufte, and N. Reissmann, "EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure," 2019.

[20] Sigma2, "Fram hardware." `https://documentation.sigma2.no/hpc_machines/fram.html`. Accessed: 24.5.2023.

Appendix A

Additional Idun Results

A.1 100w, 50h sub-domain

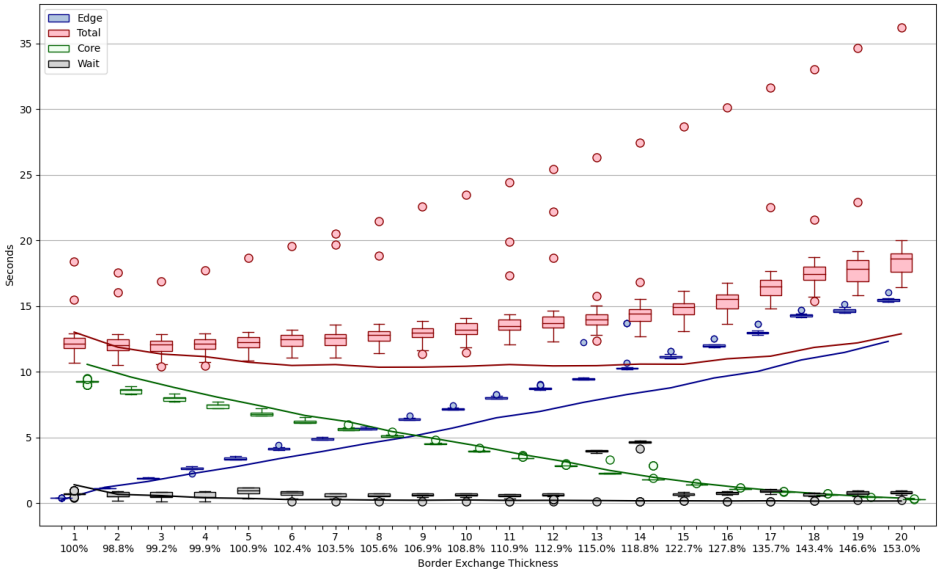


Figure A.1: Results for Idun 100w, 50h sub-domain for 1 node, and 56 MPI ranks.

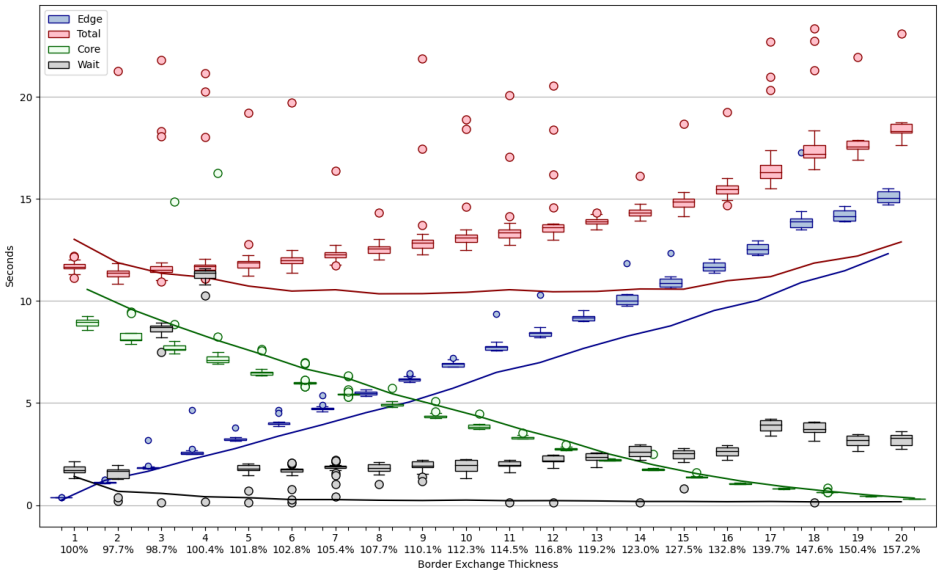


Figure A.2: Results for Idun 100w, 50h sub-domain for 4 nodes, and 224 MPI ranks.

A.2 300w, 50h sub-domain

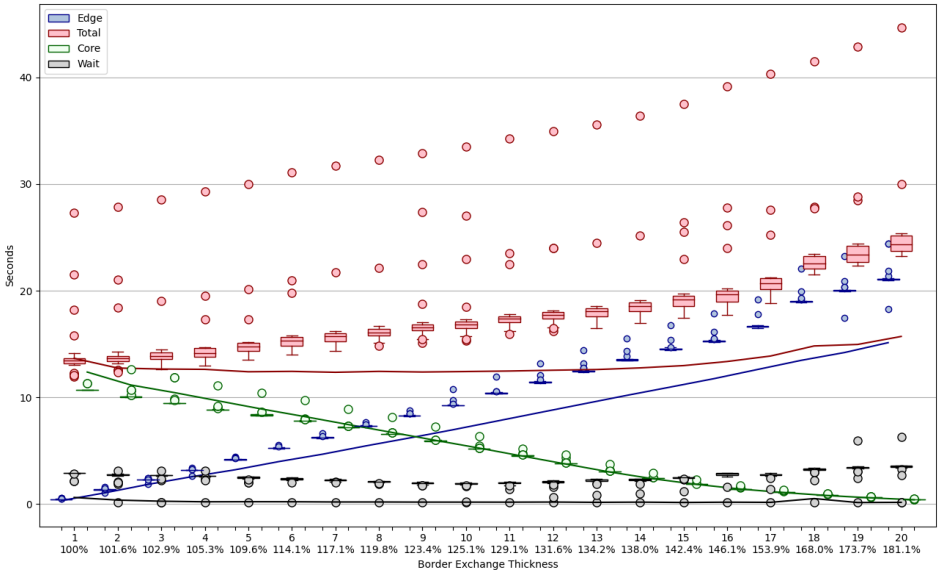


Figure A.3: Results for Idun 300w, 50h sub-domain for 1 node, and 56 MPI ranks.

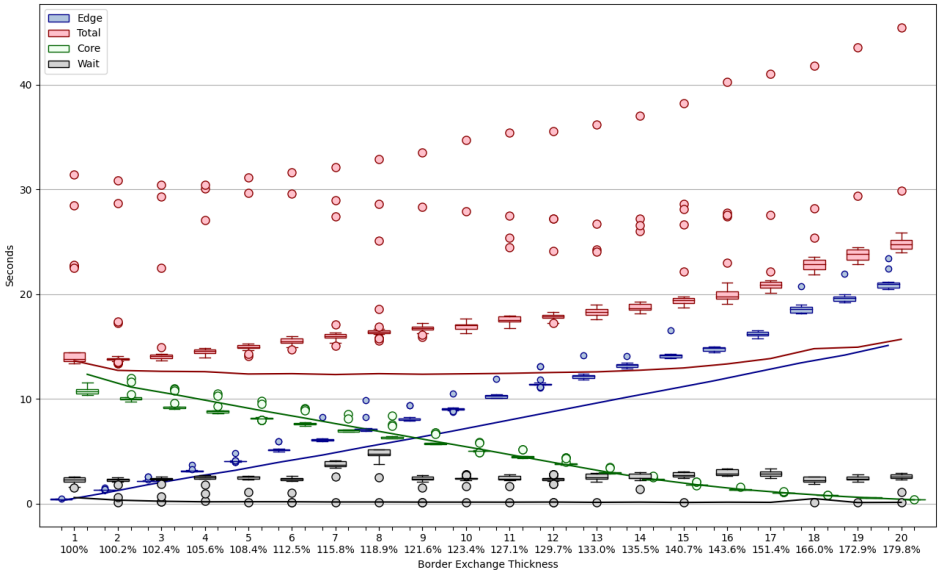


Figure A.4: Results for Idun 300w, 50h sub-domain for 4 nodes, and 224 MPI ranks.

A.3 150w, 900h sub-domain

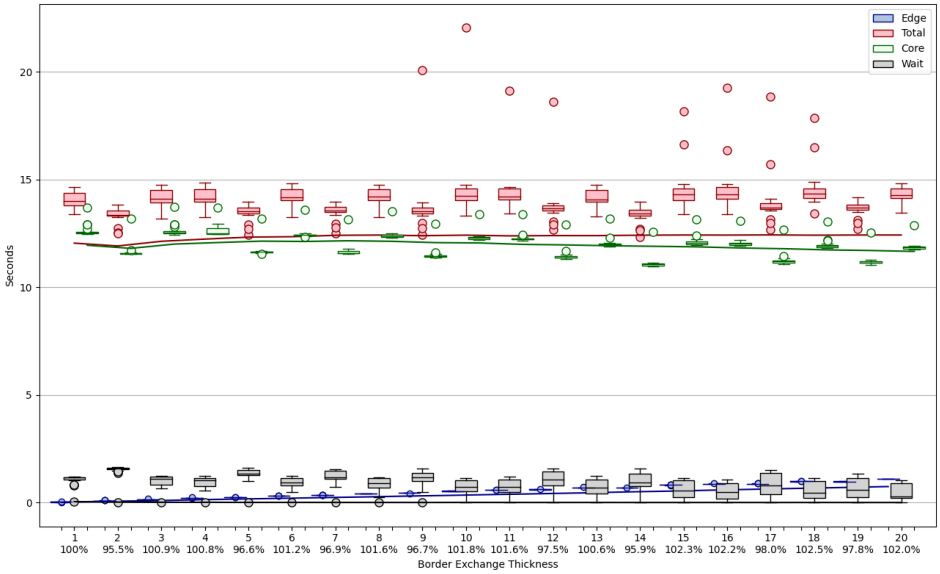


Figure A.5: Results for Idun 150w, 900h sub-domain for 1 node, and 56 MPI ranks.

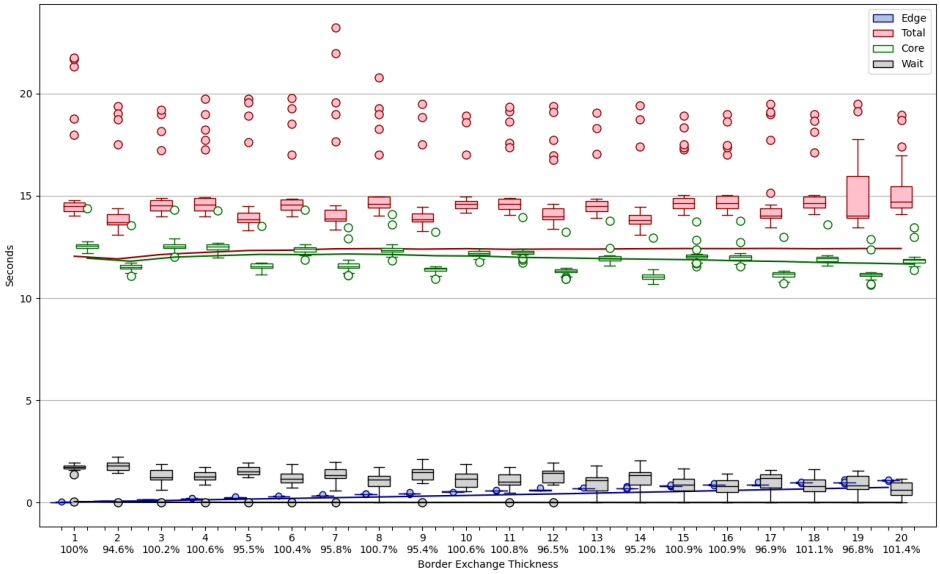


Figure A.6: Results for Idun 150w, 900h sub-domain for 4 nodes, and 224 MPI ranks.

Appendix B

Additional Fram Results

B.1 50w, 350h sub-domain

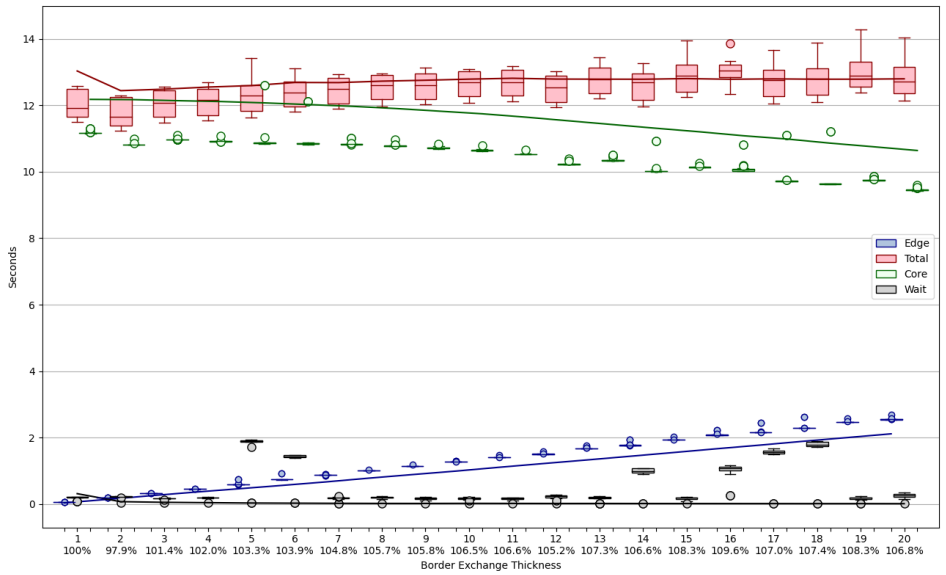


Figure B.1: Results for Fram 50w, 350h sub-domain for 1 node, and 32 MPI ranks.

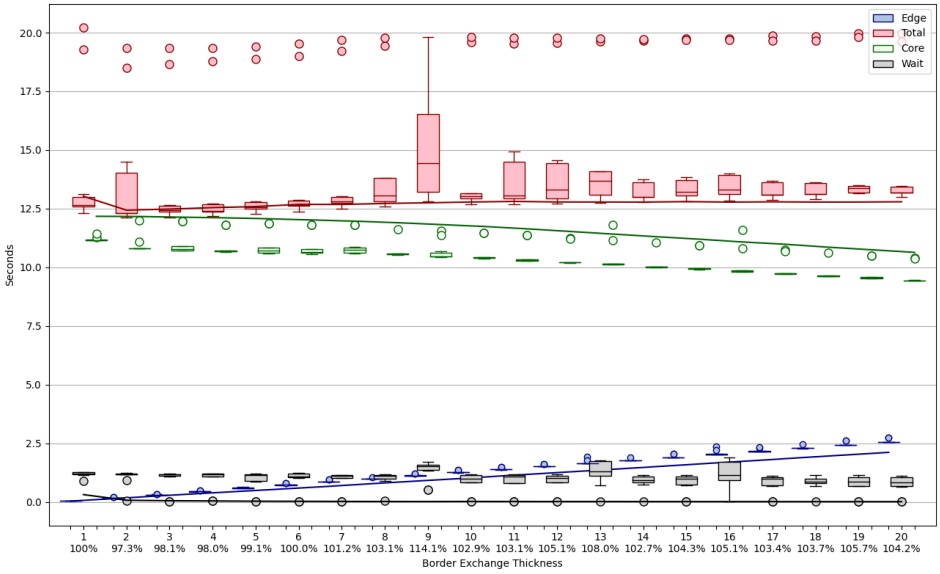


Figure B.2: Results for Fram 50w, 350h sub-domain for 8 node, and 256 MPI ranks.

B.2 300w, 1000h sub-domain

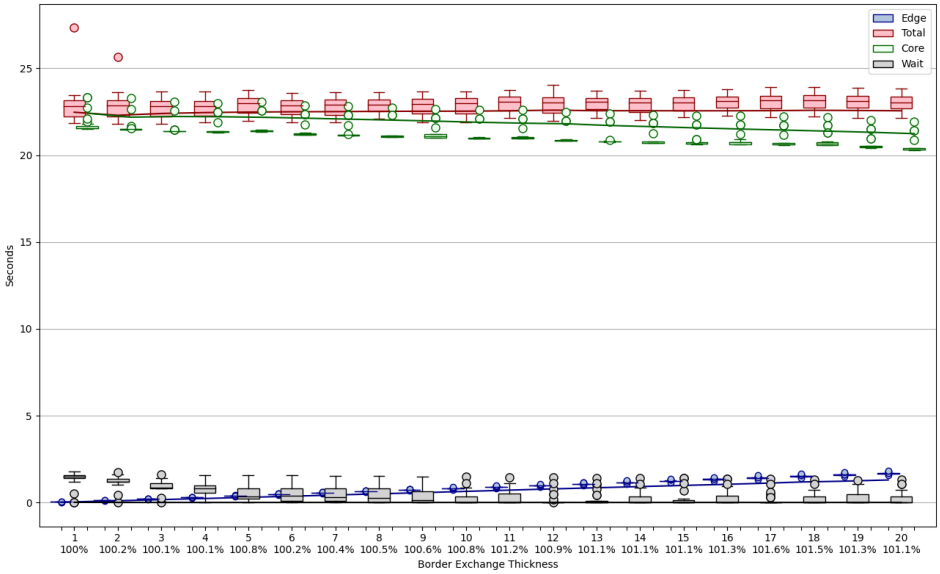


Figure B.3: Results for Fram 300w, 1000h sub-domain for 1 node, and 32 MPI ranks.

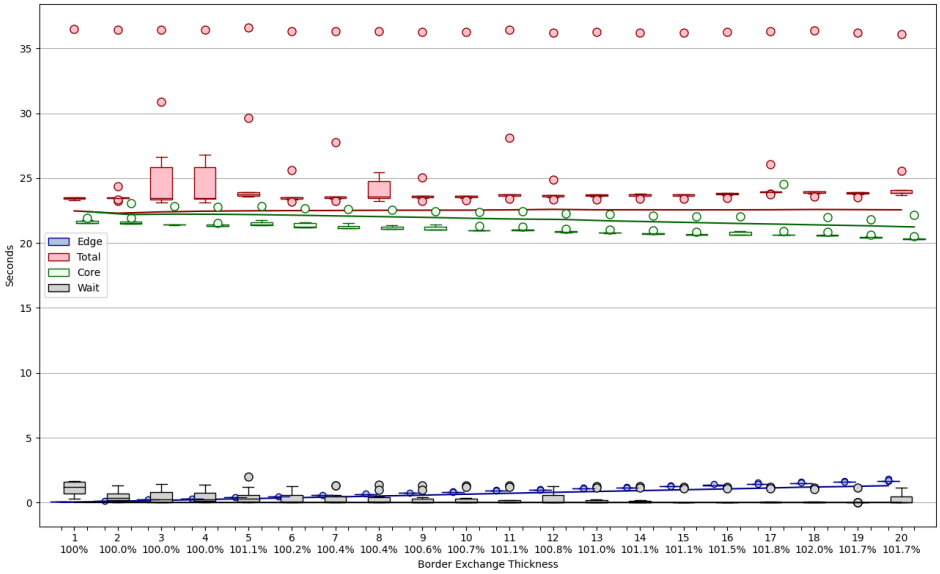


Figure B.4: Results for Fram 300w, 1000h sub-domain for 8 node, and 256 MPI ranks.

Appendix C

Idun Prediction Data

C.1 100w, 50h sub-domain.

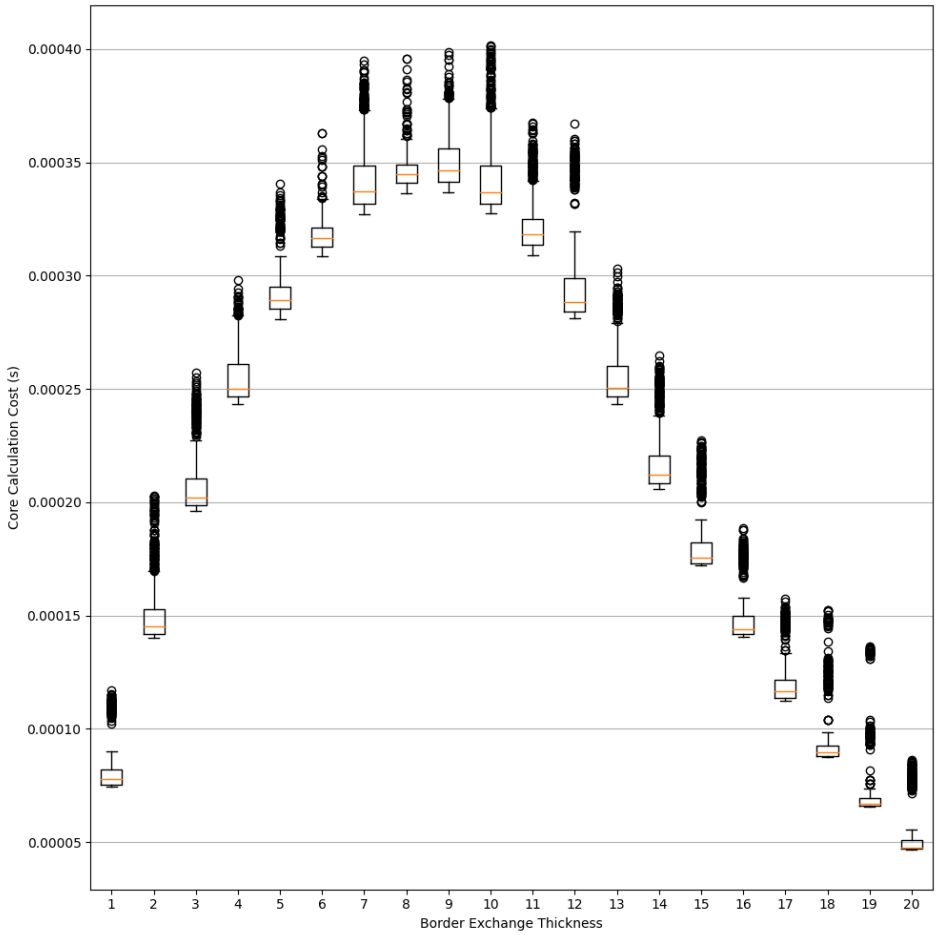


Figure C.1: Core calculation cost for each super-step for 100w, 50h sub-domain.

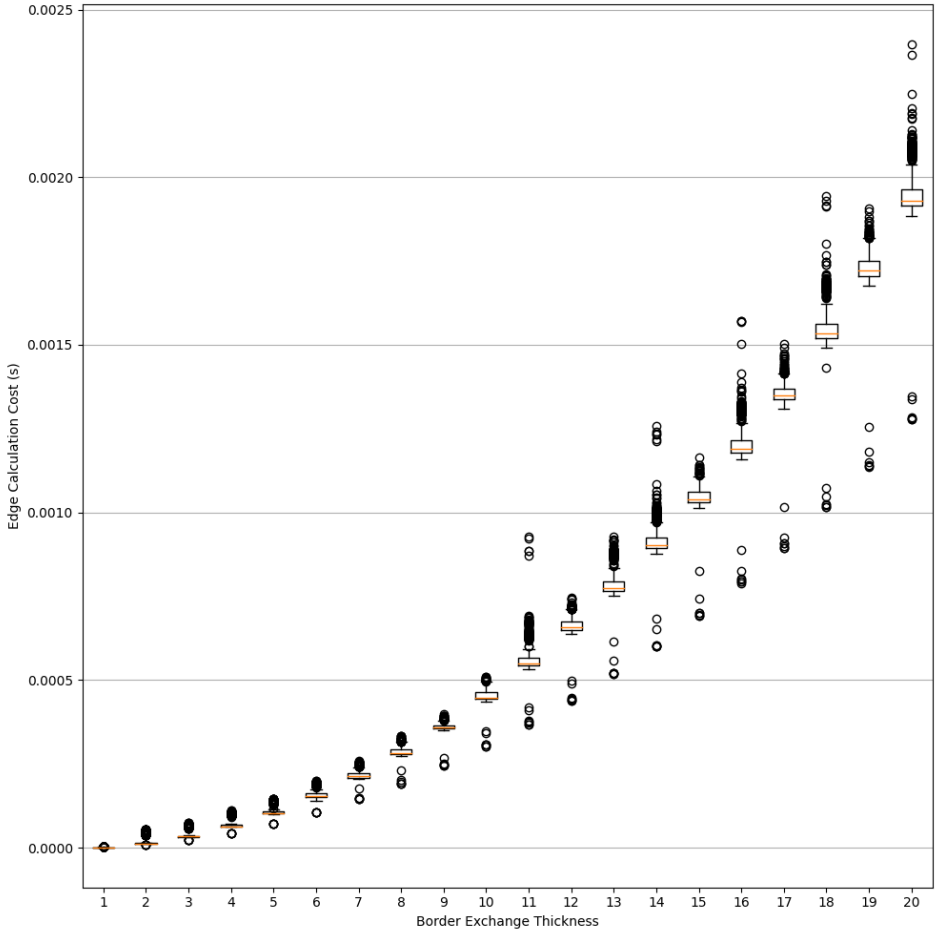


Figure C.2: Edge calculation cost for each super-step for 100w, 50h sub-domain.

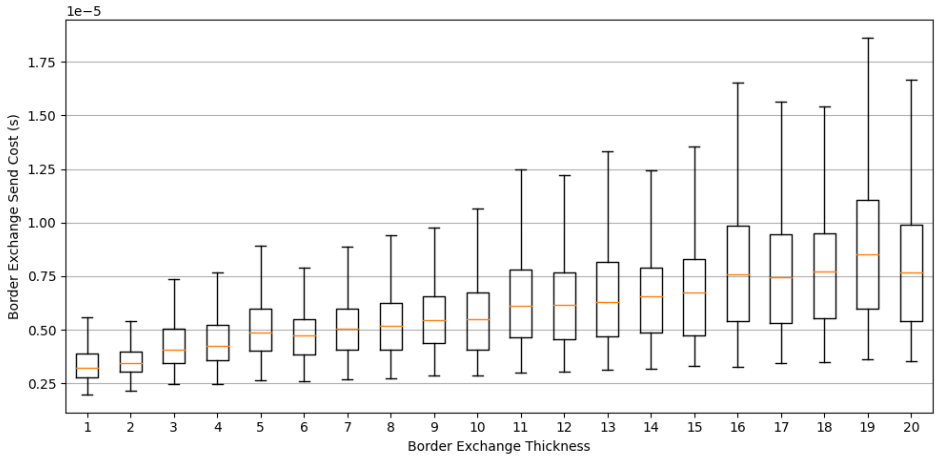


Figure C.3: Border exchange send cost for each super-step for 100w, 50h sub-domain.

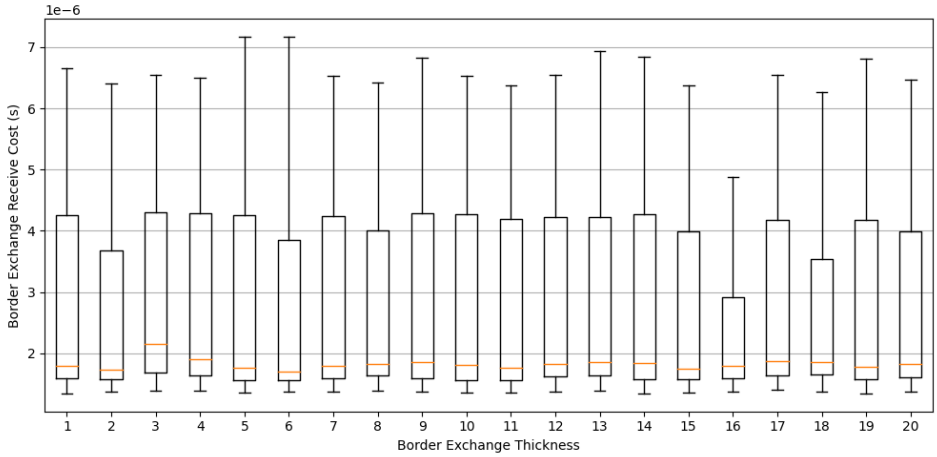


Figure C.4: Border exchange receive cost for each super-step for 100w, 50h sub-domain.

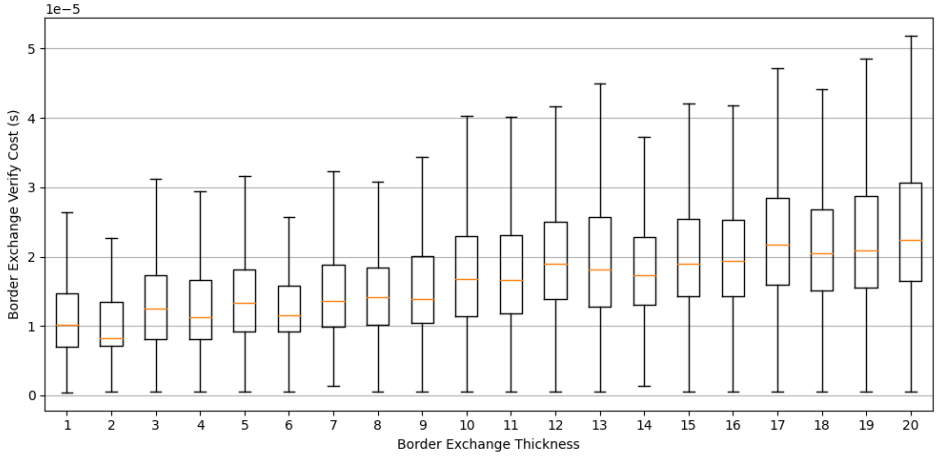


Figure C.5: Border exchange verify cost for each super-step for 100w, 50h sub-domain.

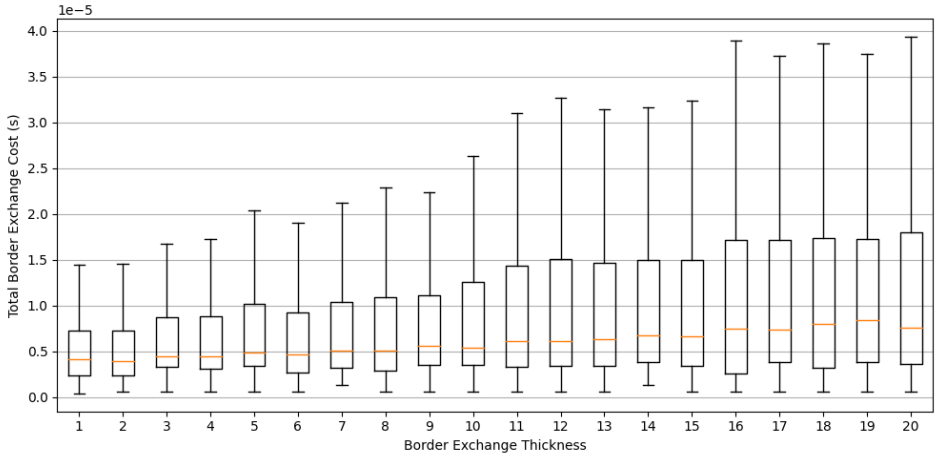


Figure C.6: Total border exchange cost for each super-step for 100w, 50h sub-domain.

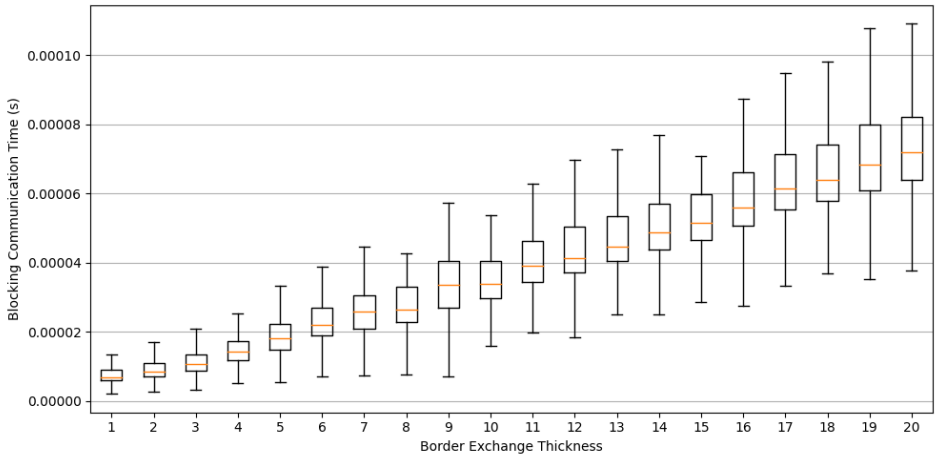


Figure C.7: Blocking communication time for each super-step for 100w, 50h sub-domain for Idun.

C.2 300w, 50h sub-domain.

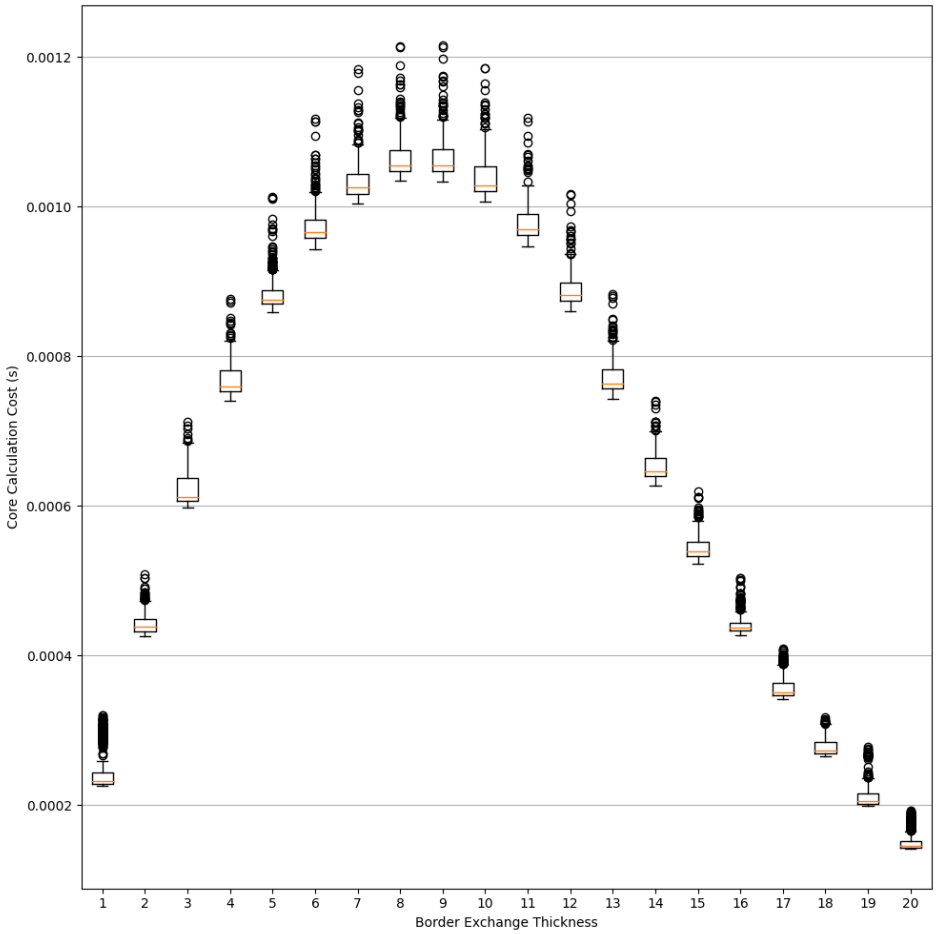


Figure C.8: Core calculation cost for each super-step for 300w, 50h sub-domain for Idun.

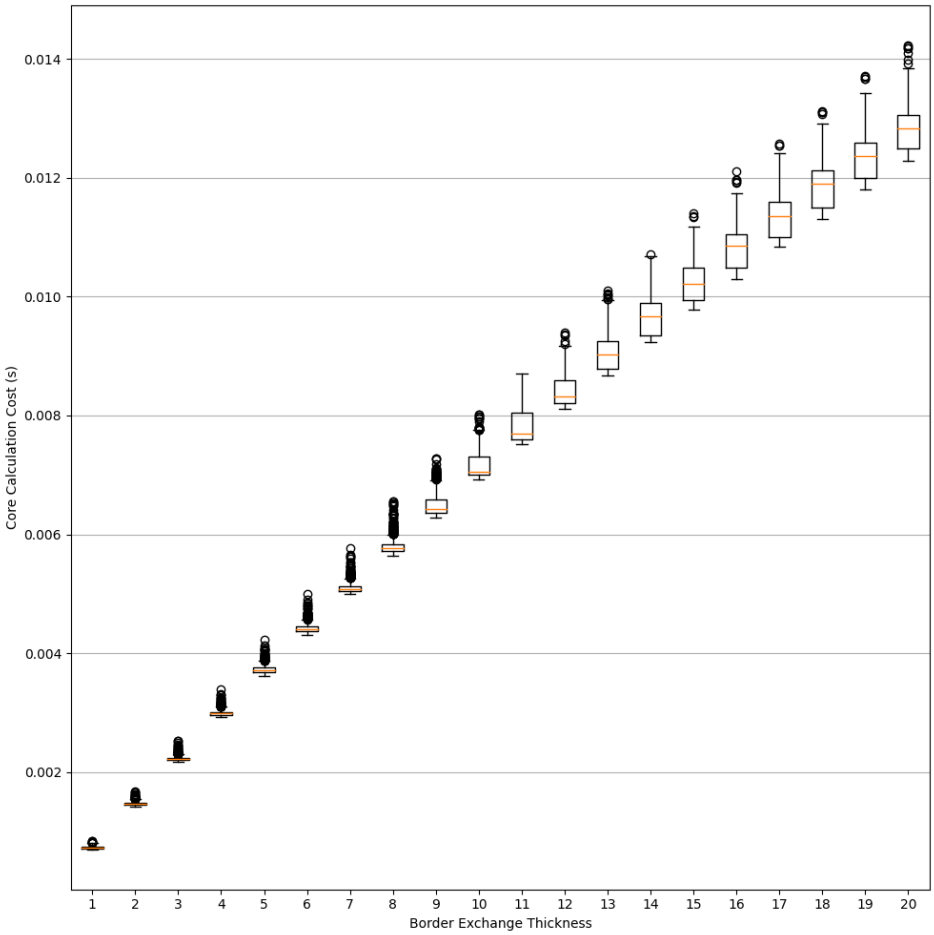


Figure C.9: Edge calculation cost for each super-step for 300w, 50h sub-domain for Idun.

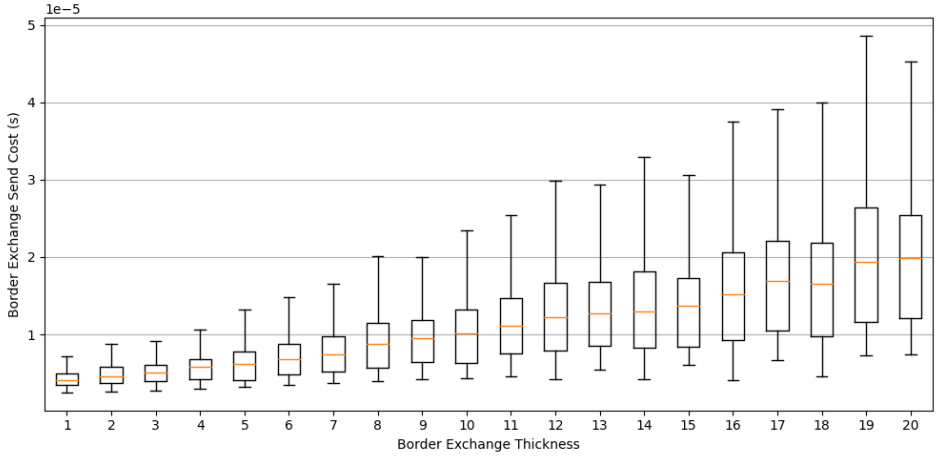


Figure C.10: Border exchange send cost for each super-step for 300w, 50h sub-domain for Idun.

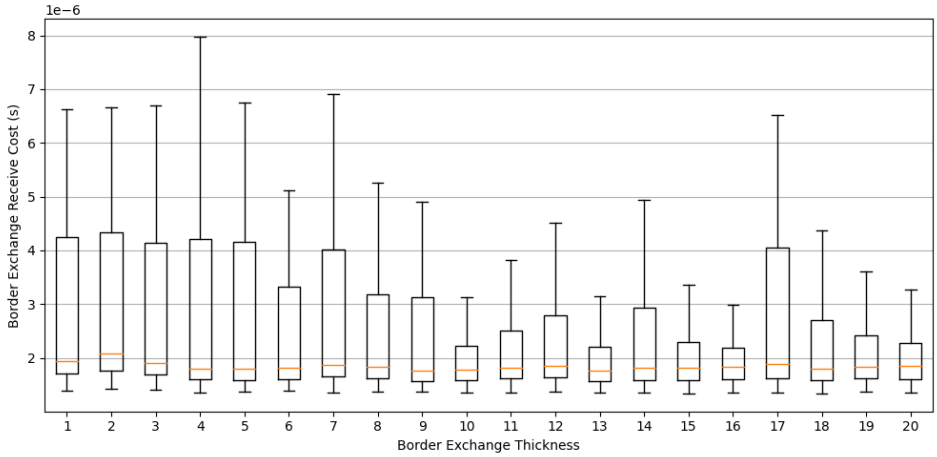


Figure C.11: Border exchange receive cost for each super-step for 300w, 50h sub-domain for Idun.

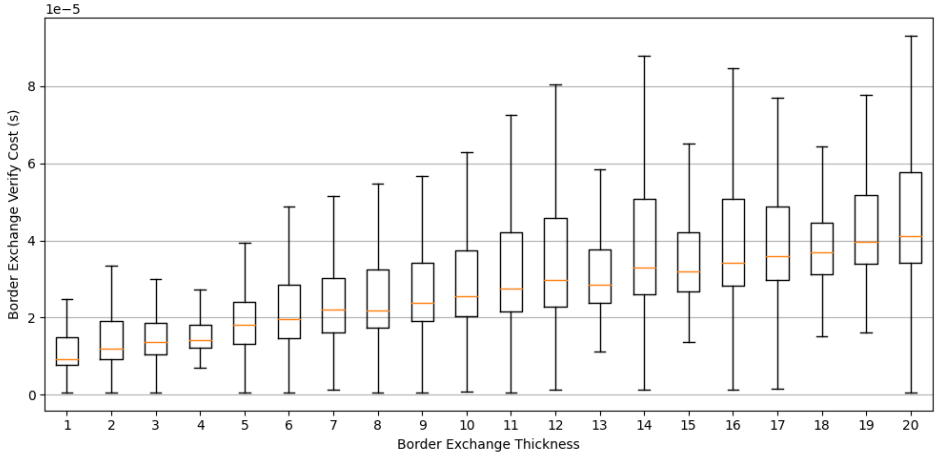


Figure C.12: Border exchange verify cost for each super-step for 300w, 50h sub-domain for Idun.

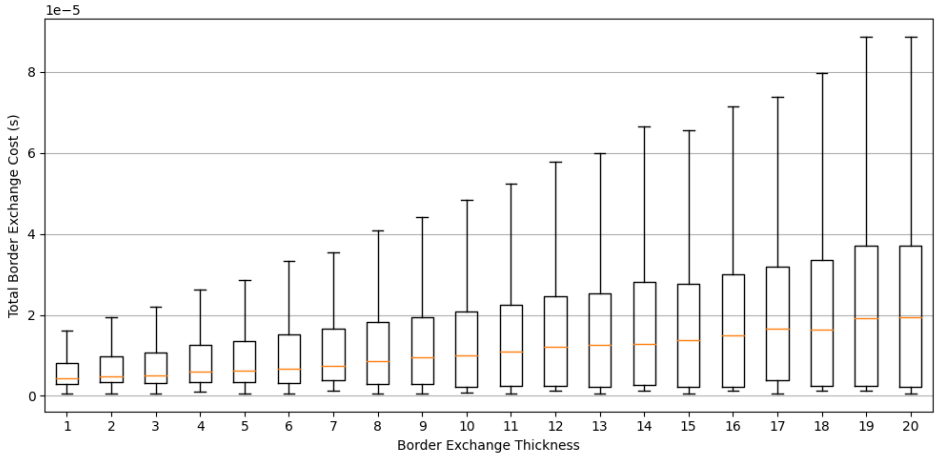


Figure C.13: Total border exchange cost for each super-step for 300w, 50h sub-domain for Idun.

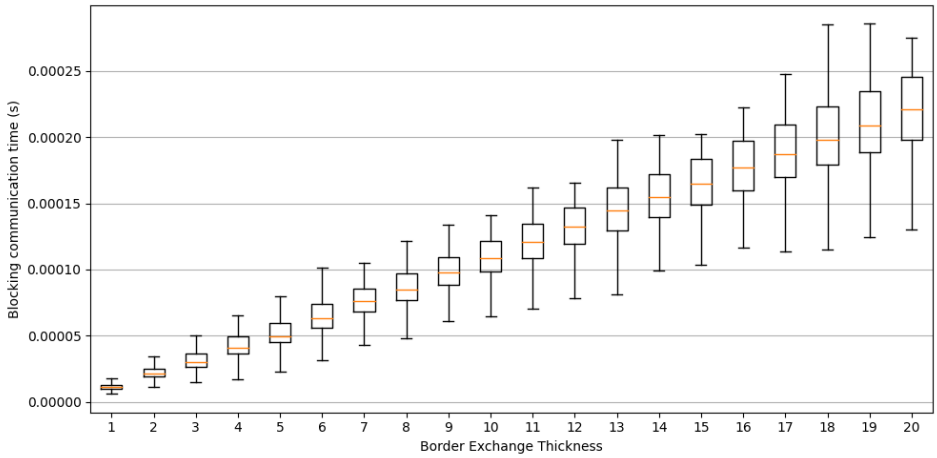


Figure C.14: Blocking communication time per super-step for 300w, 50h sub-domain for Idun.

C.3 150, 900h sub-domain.

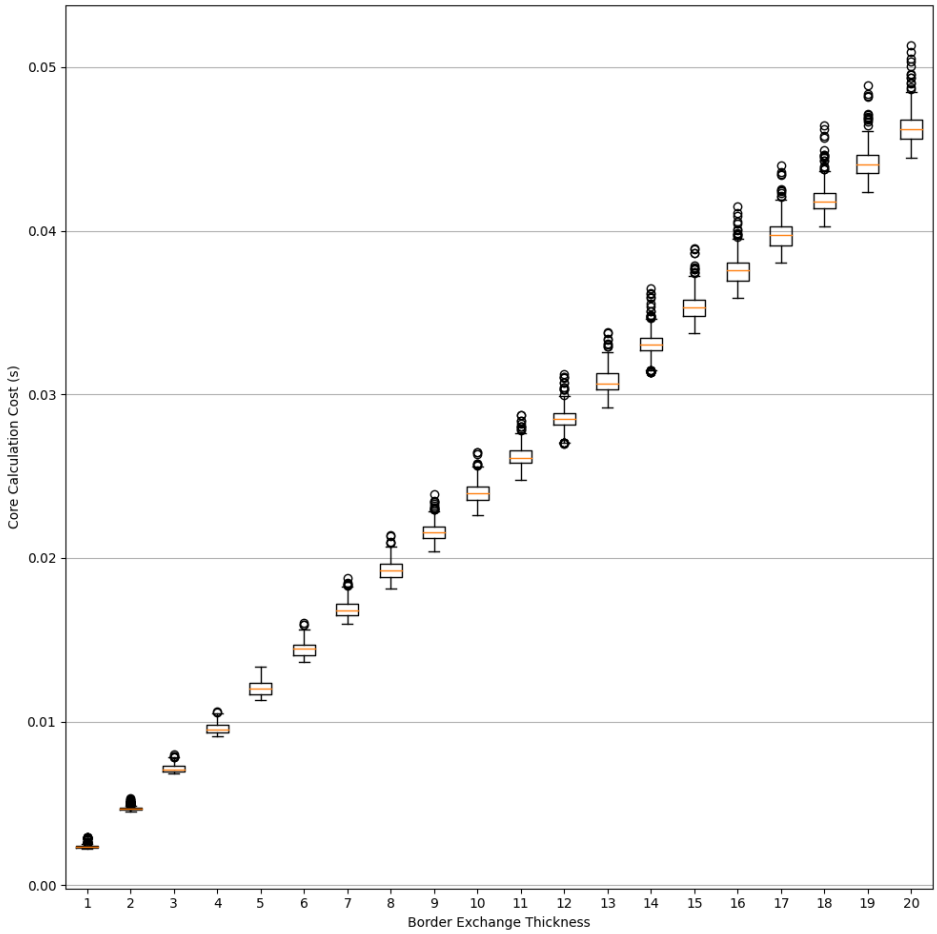


Figure C.15: Core calculation cost for each super-step for 150w, 900h sub-domain for Idun.

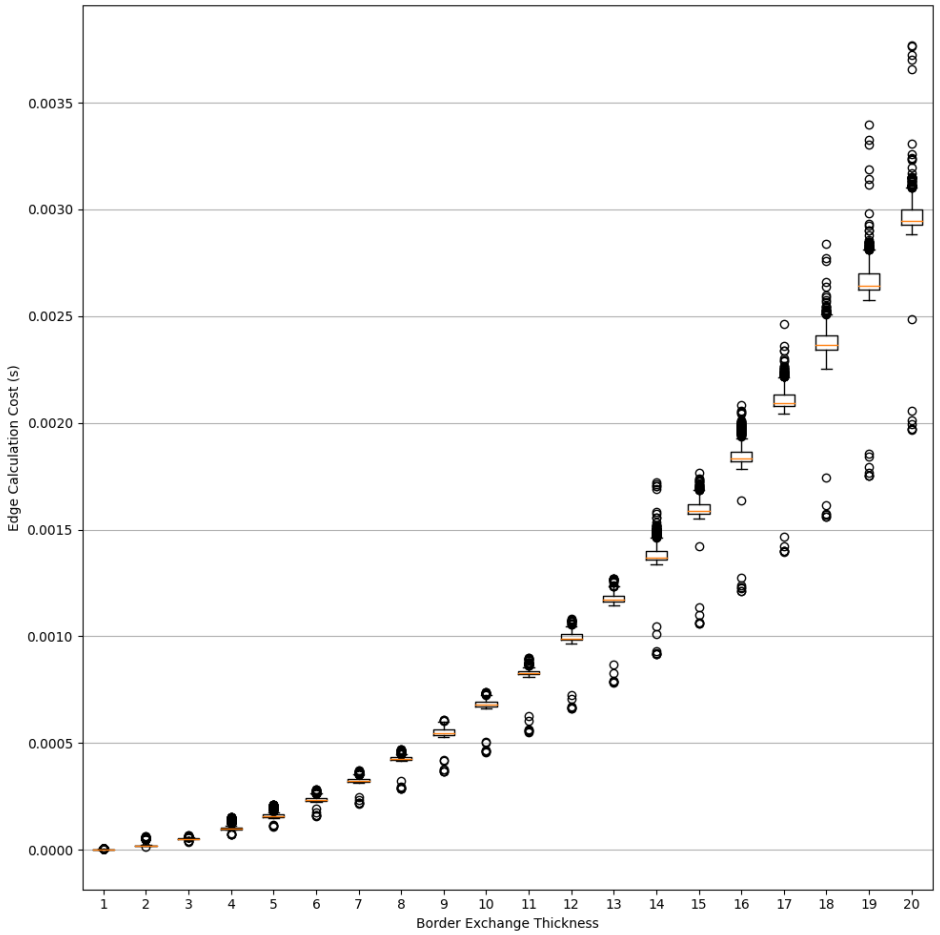


Figure C.16: Edge calculation cost for each super-step for 150w, 900h sub-domain for Idun.

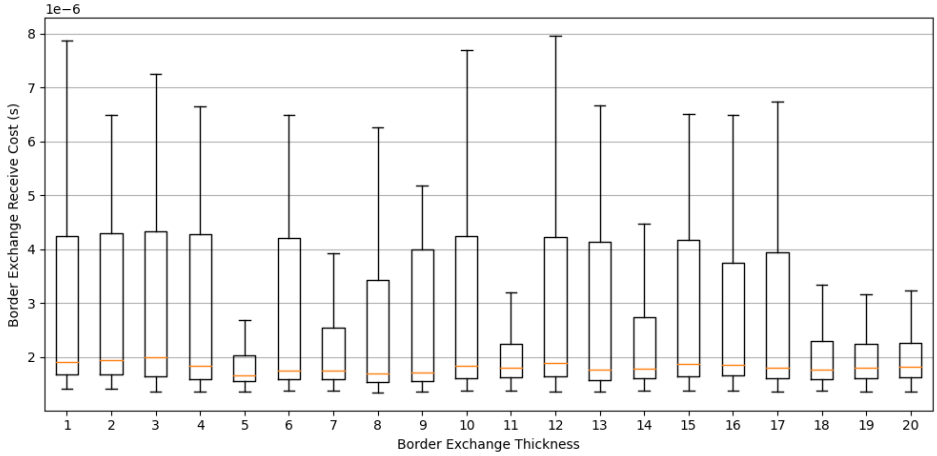


Figure C.17: Border exchange send cost for each super-step for 150w, 900h sub-domain for Idun.

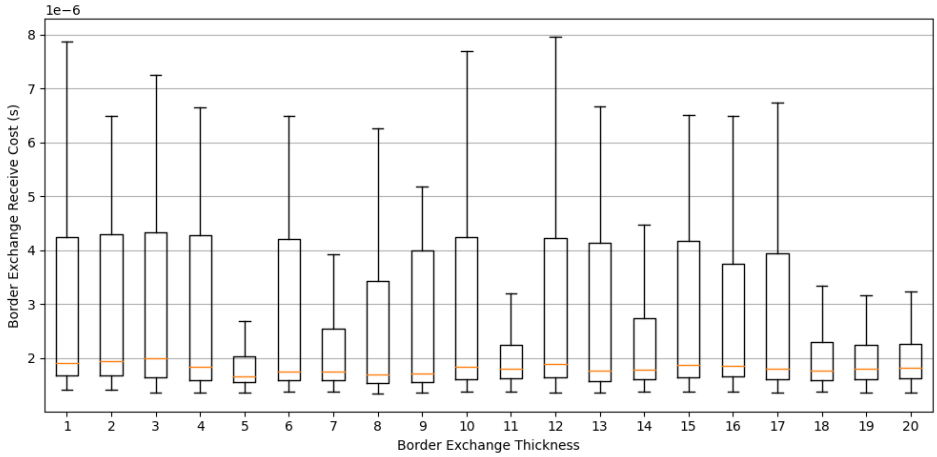


Figure C.18: Border exchange receive cost for each super-step for 150w, 900h sub-domain for Idun.

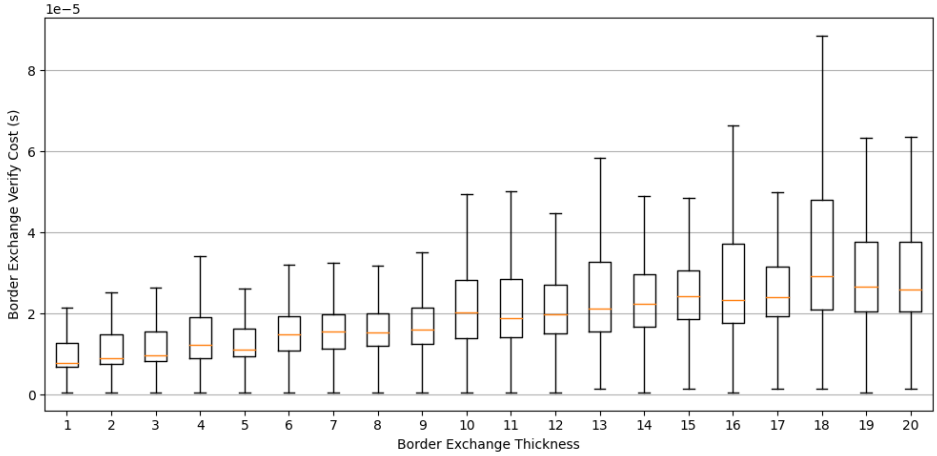


Figure C.19: Border exchange verify cost for each super-step for 150w, 900h sub-domain for Idun.

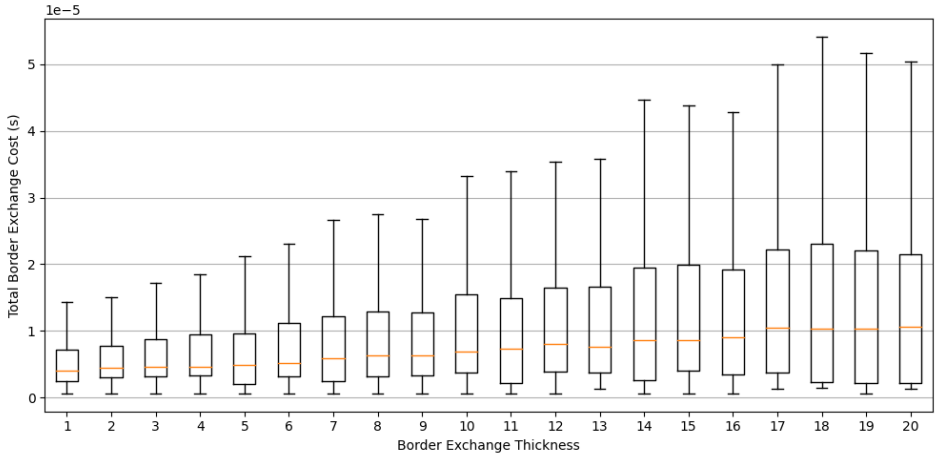


Figure C.20: Total border exchange cost for each super-step for 150w, 900h sub-domain for Idun.

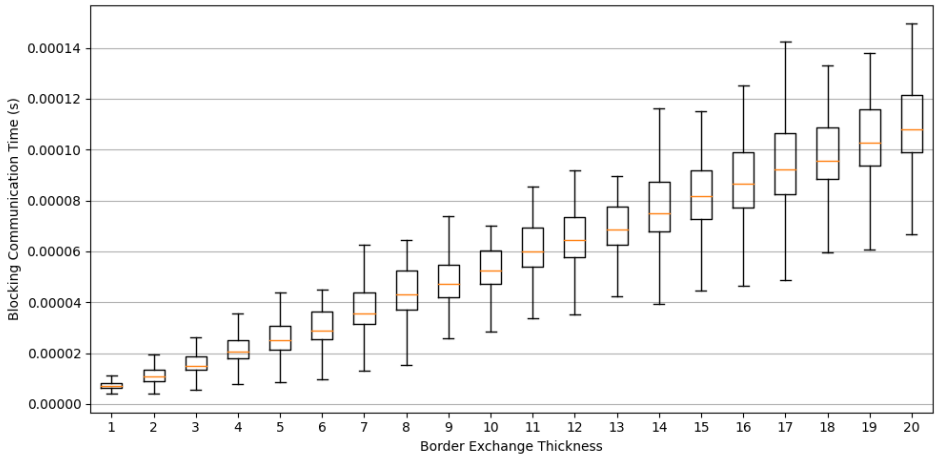


Figure C.21: Blocking communication time per super-step for 150w, 900h sub-domain for Idun.

Appendix D

Fram Prediction data

D.1 50w, 500h sub-domain.

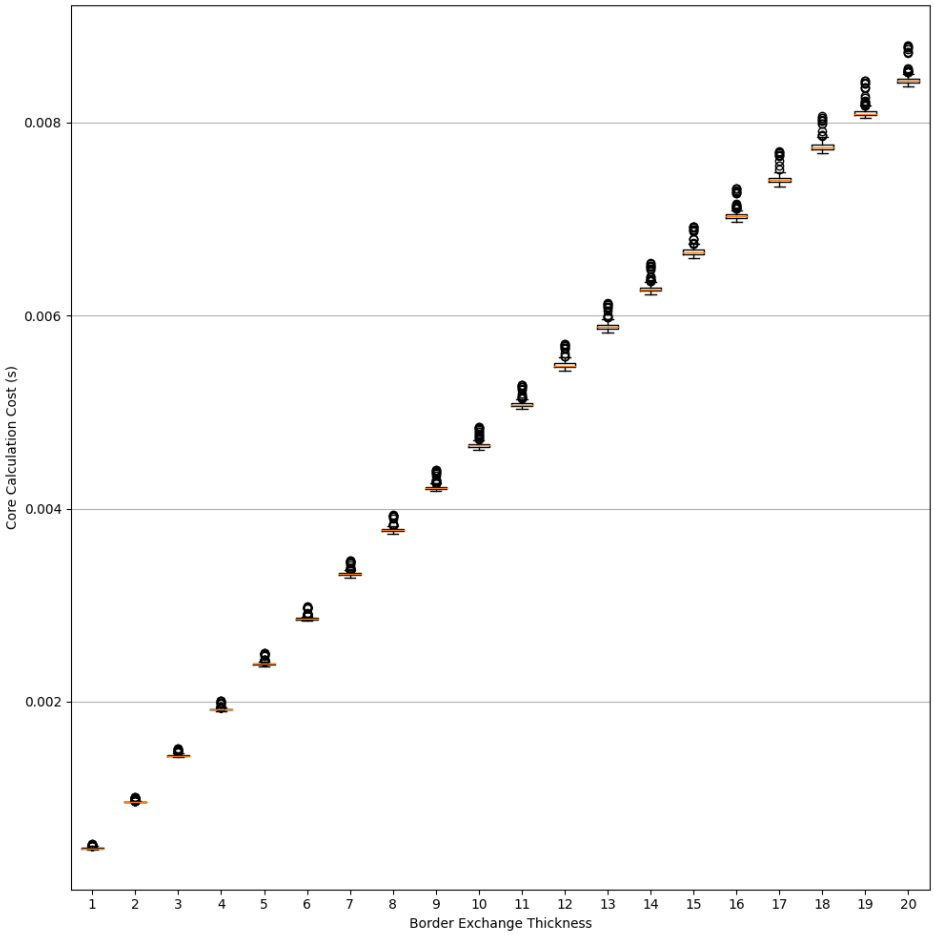


Figure D.1: Core calculation cost for each super-step for 50w, 350h sub-domain for Fram.

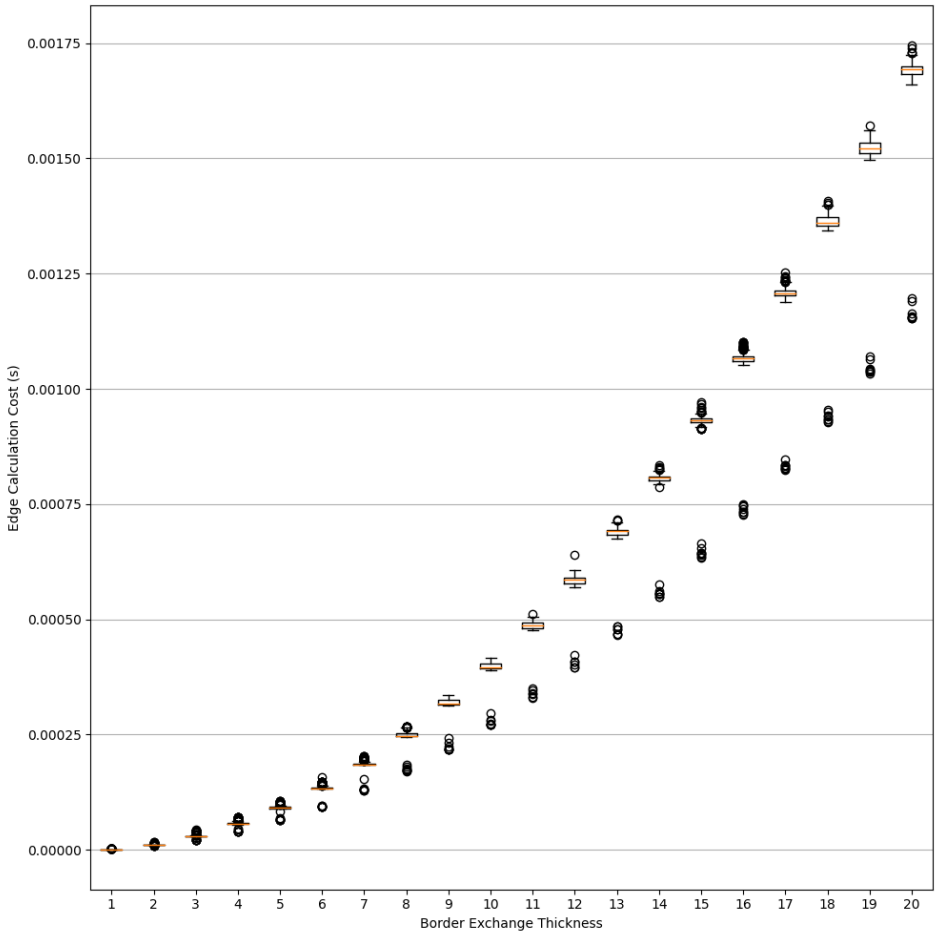


Figure D.2: Edge calculation cost for each super-step for 50w, 350h sub-domain for Fram.

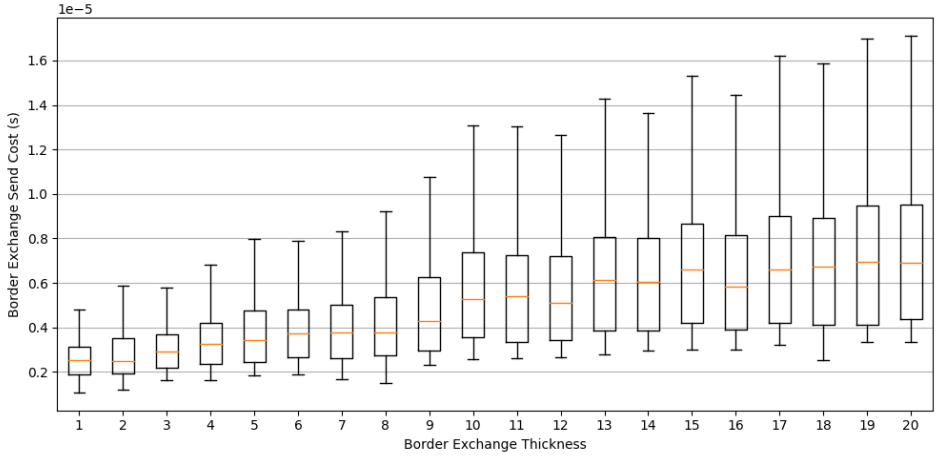


Figure D.3: Border exchange send cost for each super-step for 50w, 350h sub-domain for Fram.

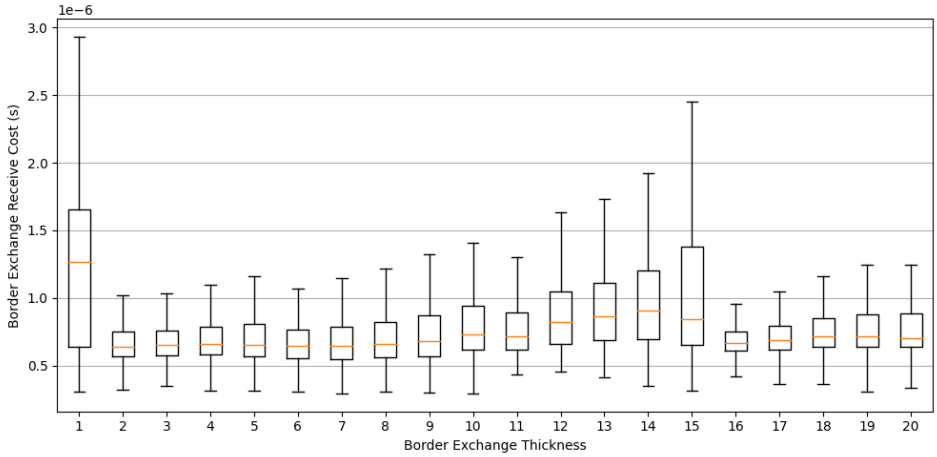


Figure D.4: Border exchange receive cost for each super-step for 50w, 350h sub-domain for Fram.

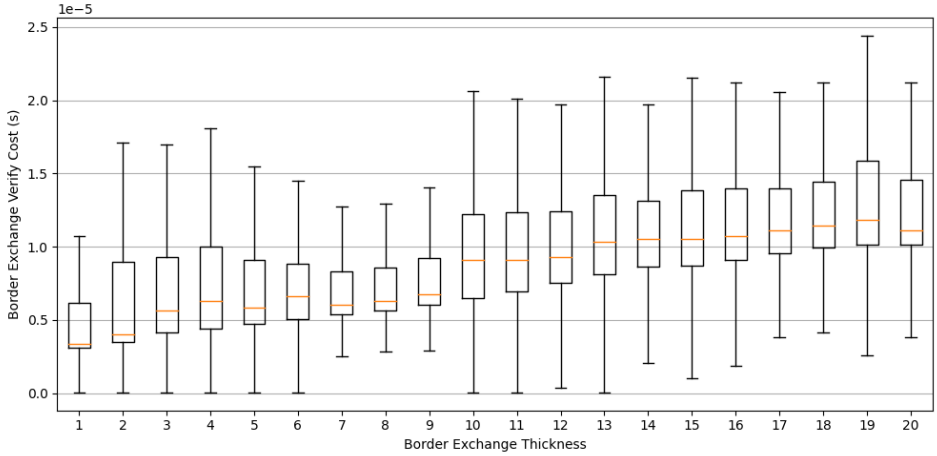


Figure D.5: Border exchange verify cost for each super-step for 50w, 350h sub-domain for Fram.

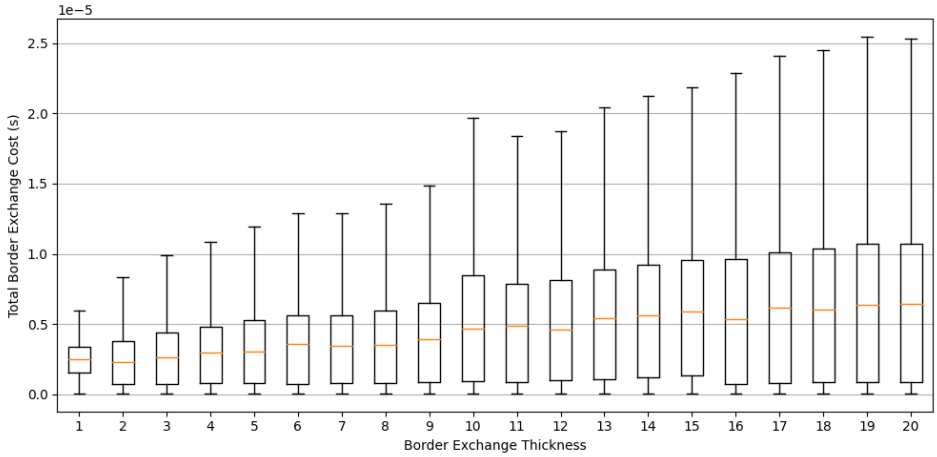


Figure D.7: Total border exchange cost for each super-step for 50w, 350h sub-domain for Fram.

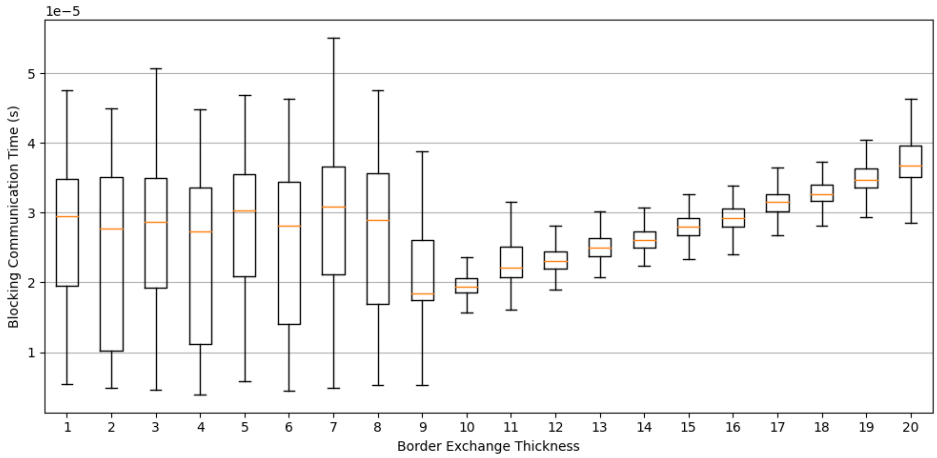


Figure D.8: Blocking communication time per super-step for 50w, 350h sub-domain for Fram.

D.2 300w, 1000h sub-domain.

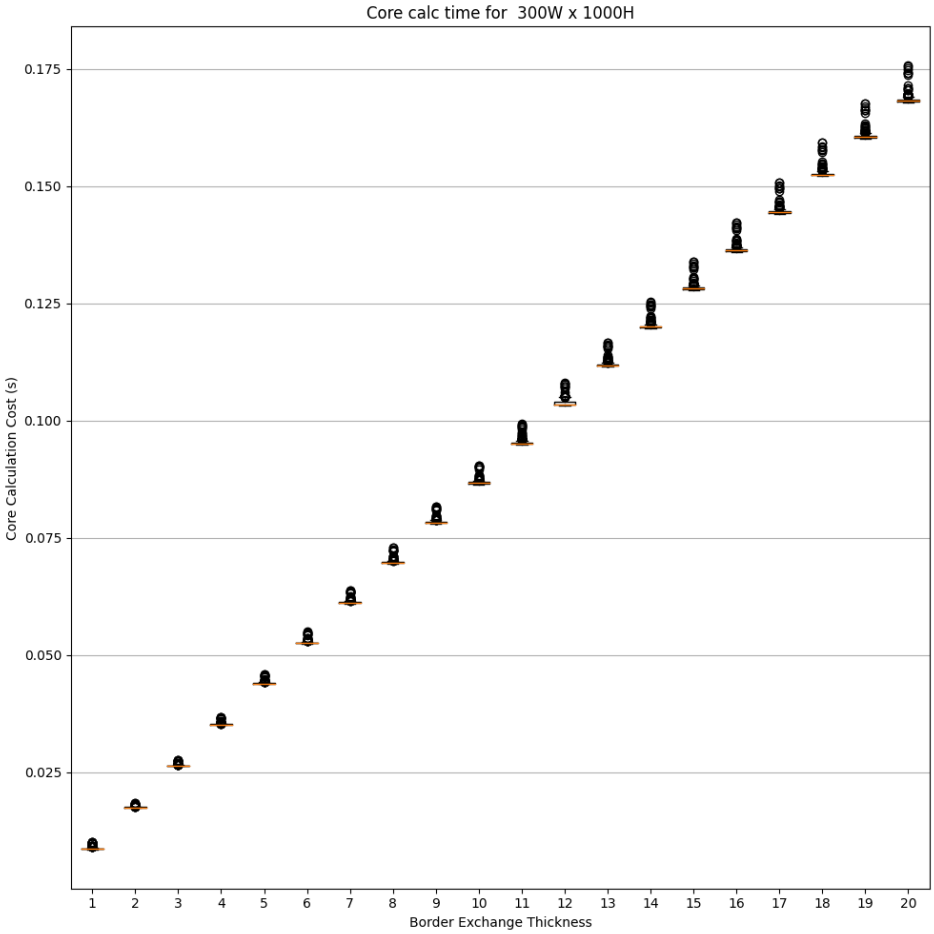


Figure D.9: Core calculation cost for each super-step for 300w, 1000h sub-domain for Fram.

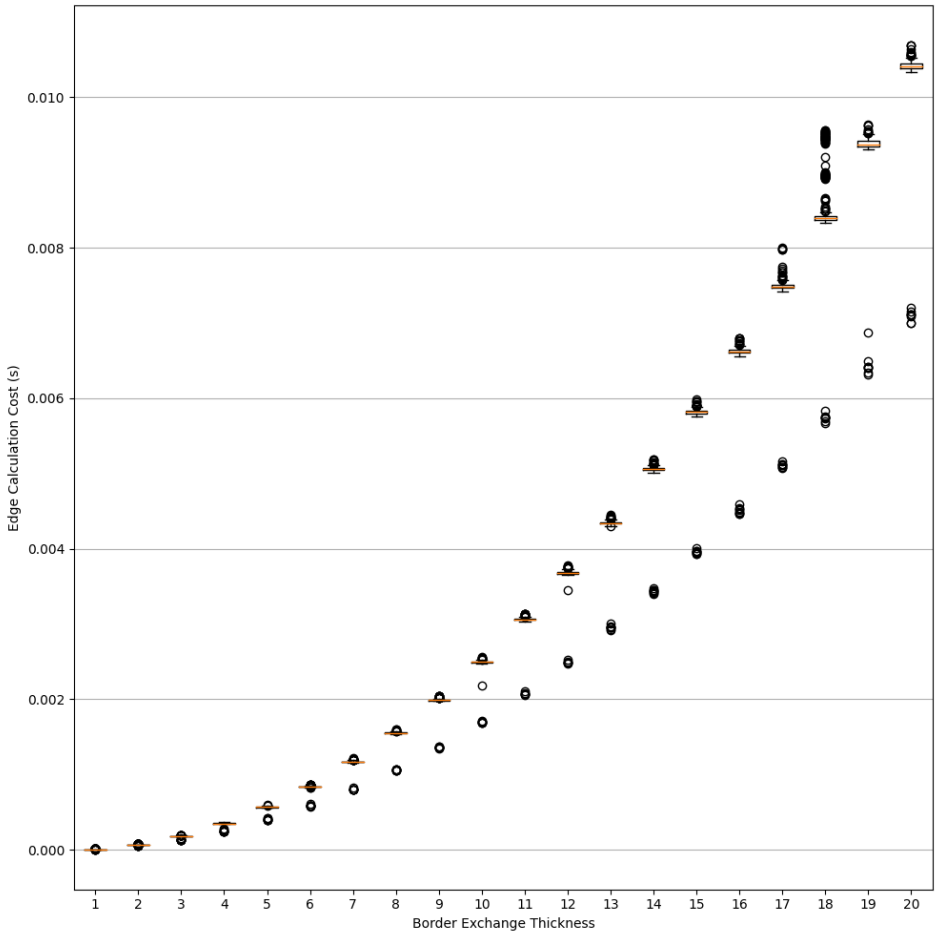


Figure D.10: Edge calculation cost for each super-step for 300w, 1000h sub-domain for Fram.

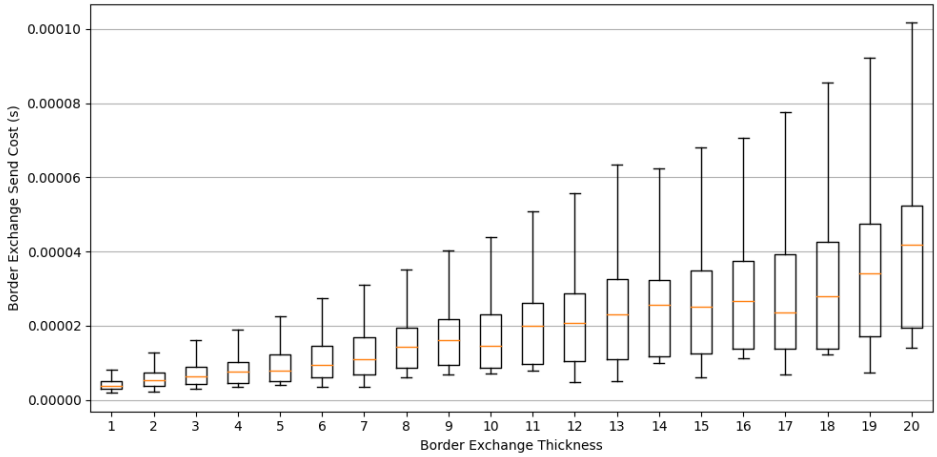


Figure D.11: Border exchange send cost for each super-step for 300w, 1000h sub-domain for Fram.

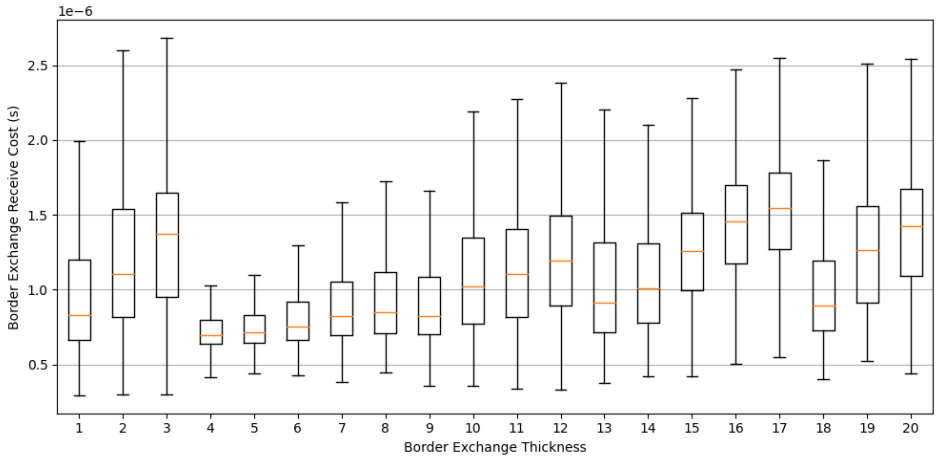


Figure D.12: Border exchange receive cost for each super-step for 300w, 1000h sub-domain for Fram.

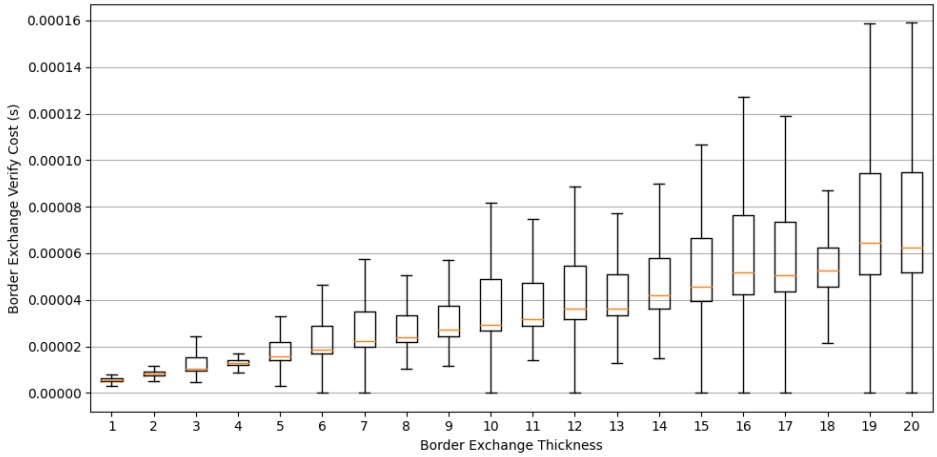


Figure D.13: Border exchange verify cost for each super-step for 300w, 1000h sub-domain for Fram.

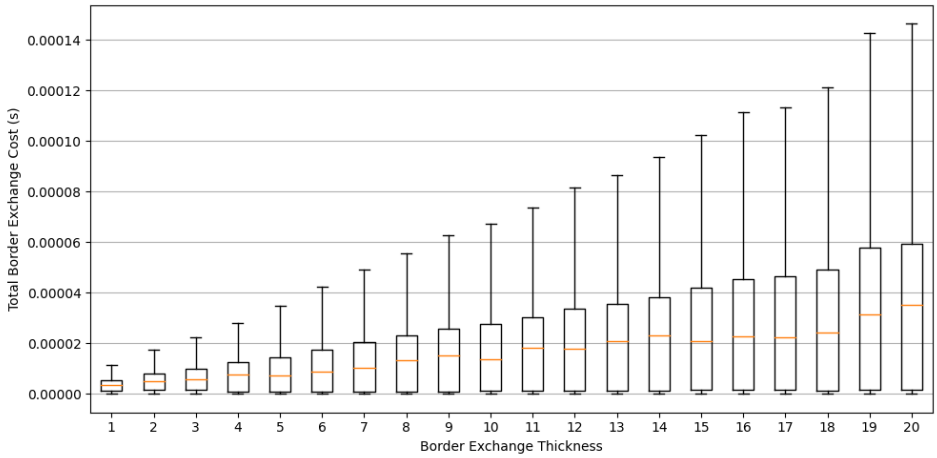


Figure D.14: Total border exchange cost for each super-step for 300w, 1000h sub-domain for Fram.

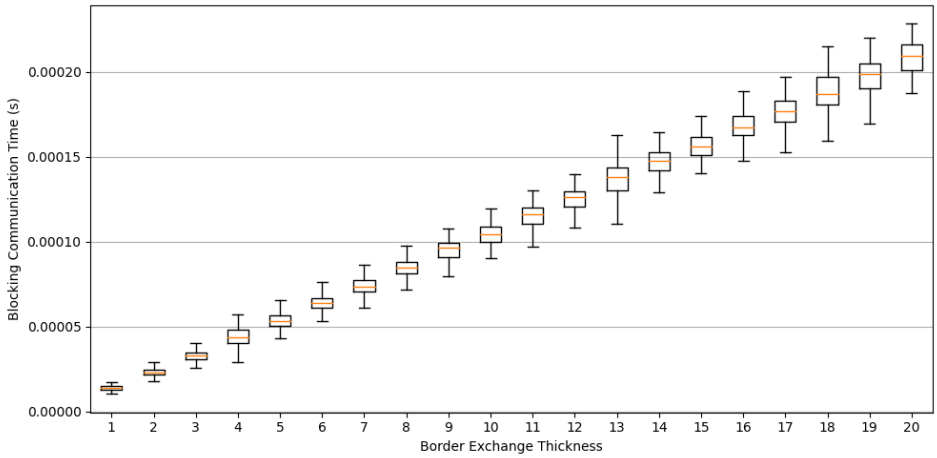


Figure D.15: Blocking communication time per super-step for 300w, 1000h sub-domain for Fram.

Appendix E

Betzy Prediction Data

E.1 - 350w, 500h sub-domain.

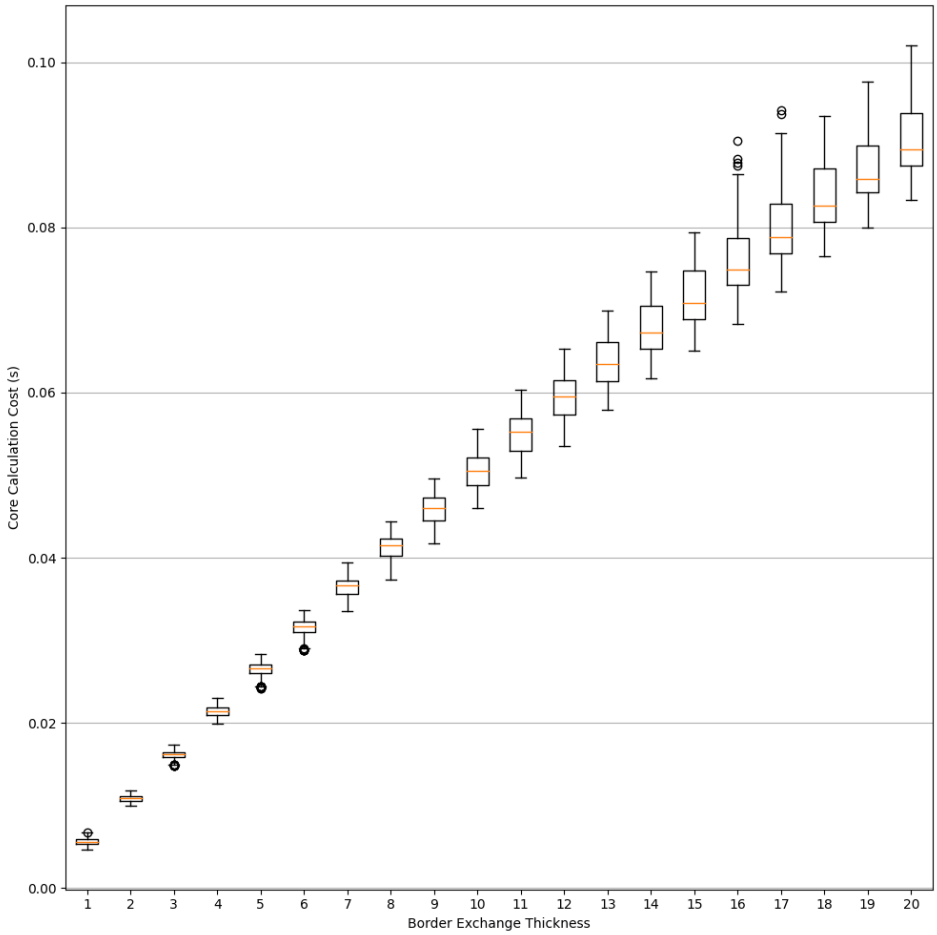


Figure E.1: Core calculation cost for each super-step for 350w, 500h sub-domain for Betzy.

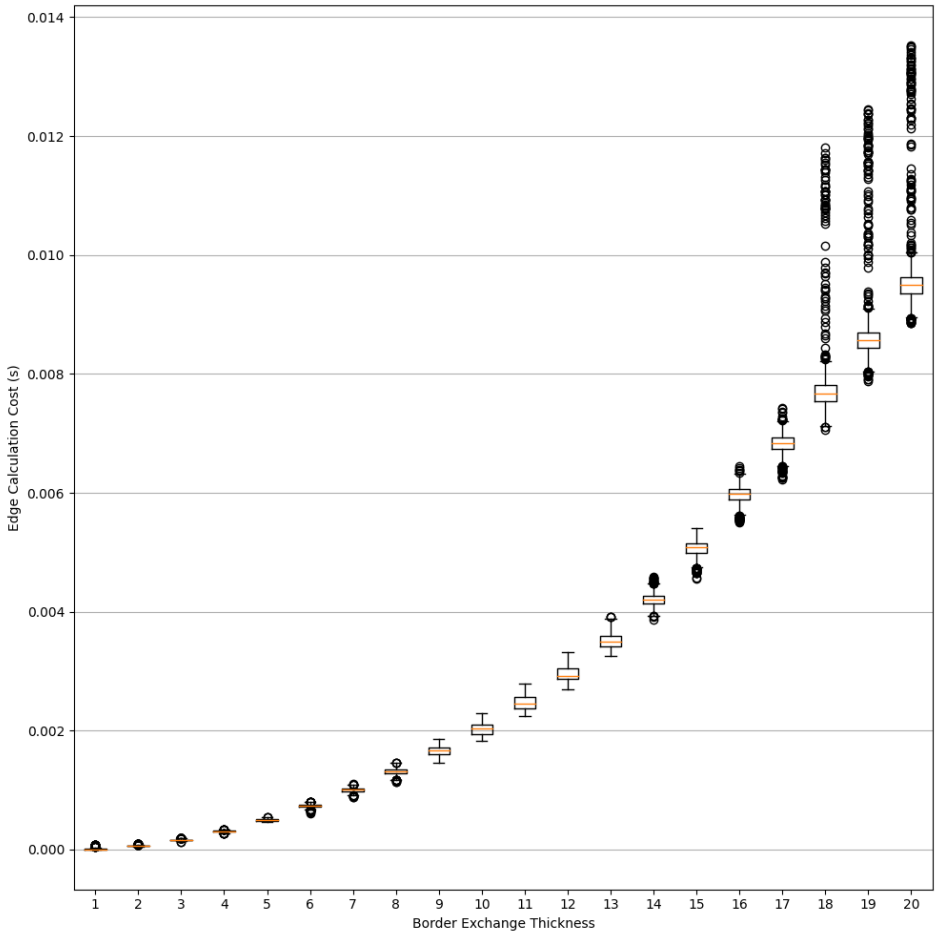


Figure E.2: Edge calculation cost for each super-step for 350w, 500h sub-domain for Betzy.

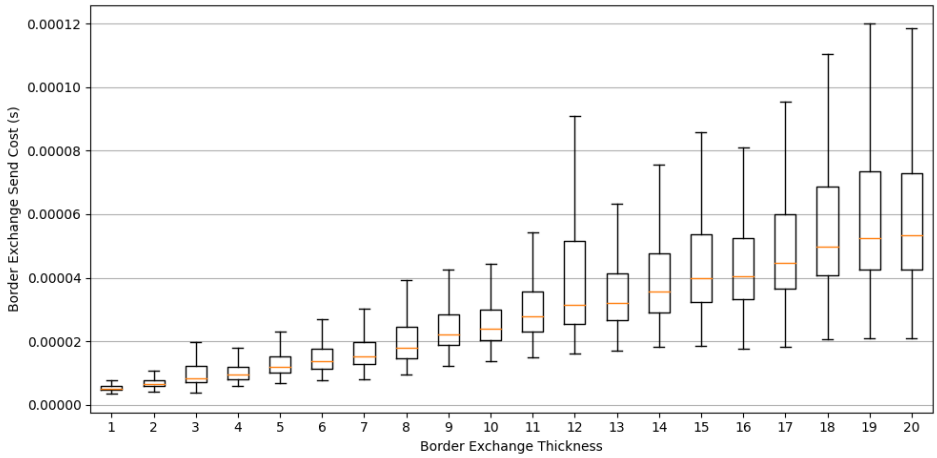


Figure E.3: Border exchange send cost for each super-step for 350w, 500h sub-domain for Betzy.

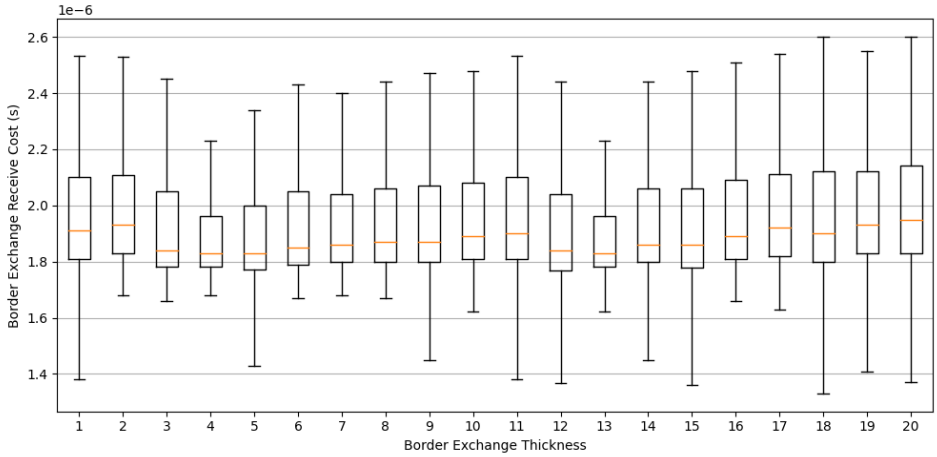


Figure E.4: Border exchange receive cost for each super-step for 350w, 500h sub-domain for Betzy.

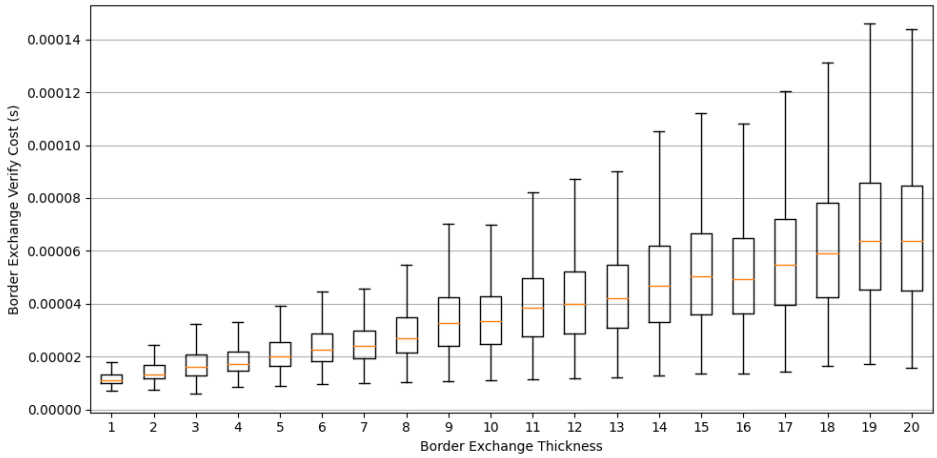


Figure E.5: Border exchange verify cost for each super-step for 350w, 500h sub-domain for Betzy.

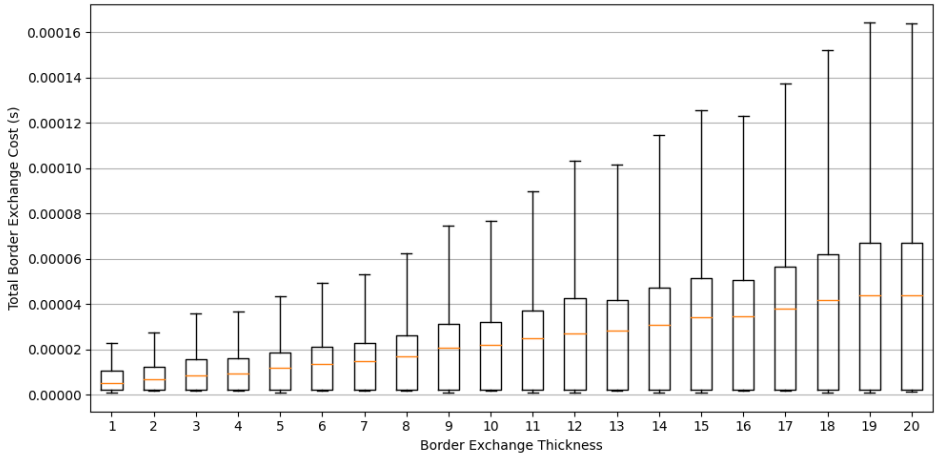


Figure E.6: Total border exchange cost for each super-step for 350w, 500h sub-domain for Betzy.

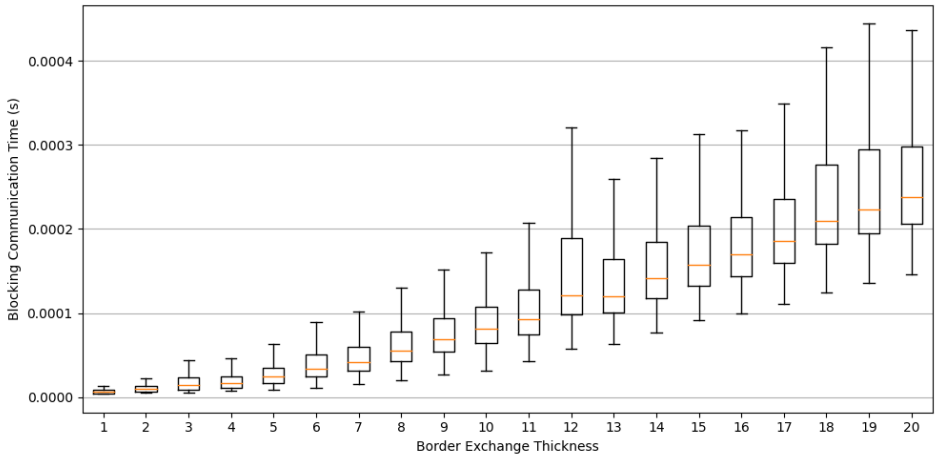


Figure E.7: Blocking communication time per super-step for 350w, 500h sub-domain for Betzy.

Appendix F

Idun Prediction Tables

Width	Height	HALO	$\frac{BExccost}{Sstep}$	$\frac{BComTime}{Sstep}$	$\frac{CoreCalc}{Sstep}$	$\frac{EdgeCalc}{Sstep}$	super-steps	$\frac{CCalc}{BCom} > 1.10$	Runtime	Percentage	H=1 runtime
100	50	1	1.76e-05	7.79e-06	8.38e-05	1.88e-06	2520.0	True	0.26	100	0.26
100	50	2	1.69e-05	9.38e-06	0.00015	1.92e-05	1260.0	True	0.24	91.21	0.26
100	50	3	2.13e-05	1.15e-05	0.00021	3.94e-05	840.0	True	0.23	87.22	0.26
100	50	4	2.70e-05	1.54e-05	0.00026	7.13e-05	630.0	True	0.22	85.71	0.26
100	50	5	2.25e-05	1.96e-05	0.00029	1.10e-04	504.0	True	0.21	82.42	0.26
100	50	6	2.02e-05	2.28e-05	0.00032	1.61e-04	420.0	True	0.21	80.52	0.26
100	50	7	2.30e-05	2.60e-05	0.00034	2.19e-04	360.0	True	0.21	81.02	0.26
100	50	8	2.33e-05	2.74e-05	0.00035	2.87e-04	315.0	True	0.21	79.50	0.26
100	50	9	2.85e-05	5.62e-05	0.00035	3.61e-04	280.0	True	0.21	79.56	0.26
100	50	10	2.84e-05	3.50e-05	0.00034	4.54e-04	252.0	True	0.21	80.06	0.26
100	50	11	3.05e-05	4.33e-05	0.00032	5.68e-04	229.09	True	0.21	81.04	0.26
100	50	12	3.02e-05	4.44e-05	0.00030	6.65e-04	210.0	True	0.21	80.26	0.26
100	50	13	3.09e-05	4.70e-05	0.00026	7.92e-04	193.85	True	0.21	80.40	0.26
100	50	14	3.79e-05	5.06e-05	0.00022	9.19e-04	180.0	True	0.21	81.32	0.26
100	50	15	3.08e-05	5.26e-05	0.00018	1.04e-03	168.0	True	0.21	81.24	0.26
100	50	16	3.48e-05	6.02e-05	0.00015	1.21e-03	157.5	True	0.21	84.41	0.26
100	50	17	3.47e-05	6.41e-05	0.00012	1.35e-03	148.24	True	0.22	85.95	0.26
100	50	18	3.92e-05	6.60e-05	9.74e-05	1.55e-03	140.0	True	0.24	91.06	0.26
100	50	19	3.60e-05	9.47e-05	7.33e-05	1.73e-03	132.63	False	0.24	93.73	0.26
100	50	20	3.72e-05	7.29e-05	5.41e-05	1.95e-03	126.0	False	0.26	99.02	0.26

Table F.1: Idun prediction data and results for sub-domain of 100w, 50h.

Width	Height	HALO	$\frac{BExccost}{Sstep}$	$\frac{BComTime}{Sstep}$	$\frac{CoreCalc}{Sstep}$	$\frac{EdgeCalc}{Sstep}$	super-steps	$\frac{CCalc}{BCom} > 1.10$	Runtime	Percentage	H=1 runtime
300	50	1	1.96e-05	1.17e-05	2.45e-04	6.01e-06	2520.0	True	0.68	100	0.68
300	50	2	2.22e-05	2.22e-05	4.43e-04	4.08e-05	1260.0	True	0.64	93.33	0.68
300	50	3	2.27e-05	3.16e-05	6.21e-04	1.09e-04	840.0	True	0.63	92.65	0.68
300	50	4	3.19e-05	5.86e-05	7.67e-04	2.02e-04	630.0	True	0.63	92.43	0.68
300	50	5	2.89e-05	5.15e-05	8.83e-04	3.17e-04	504.0	True	0.62	90.78	0.68
300	50	6	3.35e-05	6.42e-05	9.74e-04	4.72e-04	420.0	True	0.62	91.03	0.68
300	50	7	3.51e-05	7.63e-05	1.03e-03	6.46e-04	360.0	True	0.61	90.41	0.68
300	50	8	4.59e-05	8.62e-05	1.06e-03	8.63e-04	315.0	True	0.62	91.03	0.68
300	50	9	4.17e-05	9.81e-05	1.06e-03	1.10e-03	280.0	True	0.62	90.61	0.68
300	50	10	4.52e-05	1.09e-04	1.03e-03	1.38e-03	252.0	True	0.62	90.93	0.68
300	50	11	4.93e-05	1.20e-04	9.77e-04	1.69e-03	229.09	True	0.62	91.27	0.68
300	50	12	5.65e-05	1.32e-04	8.89e-04	2.04e-03	210.0	True	0.63	91.83	0.68
300	50	13	5.99e-05	1.61e-04	7.72e-04	2.42e-03	193.85	True	0.63	92.29	0.68
300	50	14	6.21e-05	1.53e-04	6.53e-04	2.83e-03	180.0	True	0.64	93.43	0.68
300	50	15	5.53e-05	1.64e-04	5.44e-04	3.26e-03	168.0	True	0.64	95.01	0.68
300	50	16	6.88e-05	1.76e-04	4.41e-04	3.73e-03	157.5	True	0.67	97.81	0.68
300	50	17	6.84e-05	1.87e-04	3.58e-04	4.25e-03	148.24	True	0.69	101.59	0.68
300	50	18	2.05e-04	1.99e-04	2.78e-04	4.81e-03	140.0	True	0.74	108.51	0.68
300	50	19	7.29e-05	2.08e-04	2.12e-04	5.35e-03	132.63	False	0.75	109.54	0.68
300	50	20	7.98e-05	2.19e-04	1.51e-04	6.00e-03	126.0	False	0.79	115.04	0.68

Table F.2: Idun prediction data and results for sub-domain of 300w, 50h.

Width	Height	HALO	$\frac{BExccost}{Sstep}$	$\frac{BComTime}{Sstep}$	$\frac{CoreCalc}{Sstep}$	$\frac{EdgeCalc}{Sstep}$	super-steps	$\frac{CCalc}{BCom} > 1.10$	Runtime	Percentage	H=1 runtime
150	900	1	1.64e-05	8.25e-06	0.00237	2.84e-06	2520.0	True	6.02	100.0	6.02
150	900	2	1.85e-05	1.22e-05	0.00468	2.61e-05	1260.0	True	5.96	98.91	6.02
150	900	3	2.42e-05	1.59e-05	0.00715	5.21e-05	840.0	True	6.07	100.71	6.02
150	900	4	2.21e-05	2.17e-05	0.00958	0.00010	630.0	True	6.12	101.56	6.02
150	900	5	2.06e-05	2.62e-05	0.01204	0.00016	504.0	True	6.16	102.32	6.02
150	900	6	2.37e-05	3.03e-05	0.01443	0.00024	420.0	True	6.17	102.47	6.02
150	900	7	2.57e-05	3.75e-05	0.01688	0.00033	360.0	True	6.21	103.02	6.02
150	900	8	3.39e-05	5.59e-05	0.01926	0.00043	315.0	True	6.21	103.11	6.02
150	900	9	2.72e-05	4.83e-05	0.02155	0.00055	280.0	True	6.20	102.88	6.02
150	900	10	3.41e-05	5.30e-05	0.02392	0.00068	252.0	True	6.21	103.07	6.02
150	900	11	3.39e-05	6.11e-05	0.02617	0.00083	229.09	True	6.19	102.81	6.02
150	900	12	4.01e-05	6.54e-05	0.02848	0.00100	210.0	True	6.19	102.70	6.02
150	900	13	4.30e-05	7.51e-05	0.03078	0.00119	194.18	True	6.19	102.61	6.02
150	900	14	4.79e-05	8.42e-05	0.03310	0.00140	180.0	True	6.20	102.55	6.02
150	900	15	5.05e-05	9.44e-05	0.03542	0.00163	168.0	True	6.19	102.47	6.02
150	900	16	5.38e-05	0.00010	0.03777	0.00187	157.5	True	6.18	102.39	6.02
150	900	17	5.68e-05	0.00011	0.04007	0.00213	148.24	True	6.18	102.34	6.02
150	900	18	6.02e-05	0.00012	0.04239	0.00240	140.0	True	6.17	102.29	6.02
150	900	19	6.22e-05	0.00013	0.04469	0.00269	132.63	True	6.17	102.24	6.02
150	900	20	6.41e-05	0.00014	0.04700	0.00300	126.0	True	6.17	102.19	6.02

Table F.3: Idun prediction data and results for sub-domain of 150w, 900h.

Appendix G

Fram Prediction Tables

Width	Height	HALO	$\frac{BExccost}{Sstep}$	$\frac{BComTime}{Sstep}$	$\frac{CoreCalc}{Sstep}$	$\frac{EdgeCalc}{Sstep}$	super-steps	$\frac{CCalc}{BCom} > 1.10$	Runtime	Percentage	H=1 runtime
50	350	1	3.23e-05	7.01e-05	0.00048	1.57e-06	2520.0	True	1.30	100	1.30
50	350	2	9.93e-06	3.74e-05	0.00097	1.15e-05	1260.0	True	1.24	95.46	1.30
50	350	3	1.07e-05	2.67e-05	0.00144	3.06e-05	840.0	True	1.25	95.81	1.30
50	350	4	1.16e-05	2.44e-05	0.00192	5.75e-05	630.0	True	1.25	96.29	1.30
50	350	5	1.14e-05	2.78e-05	0.00239	9.18e-05	504.0	True	1.26	96.63	1.30
50	350	6	2.35e-05	2.59e-05	0.00286	1.34e-04	420.0	True	1.27	97.34	1.30
50	350	7	1.18e-05	2.93e-05	0.00332	1.86e-04	360.0	True	1.27	97.32	1.30
50	350	8	1.21e-05	2.76e-05	0.00378	2.49e-04	315.0	True	1.27	97.65	1.30
50	350	9	1.34e-05	2.20e-05	0.00422	3.18e-04	280.0	True	1.28	97.86	1.30
50	350	10	1.81e-05	2.10e-05	0.00466	3.95e-04	252.0	True	1.28	98.12	1.30
50	350	11	2.55e-05	9.98e-05	0.00508	4.85e-04	229.09	True	1.28	98.32	1.30
50	350	12	1.67e-05	2.35e-05	0.00549	5.82e-04	210.0	True	1.28	98.13	1.30
50	350	13	1.84e-05	2.53e-05	0.00589	6.86e-04	193.85	True	1.28	98.11	1.30
50	350	14	1.85e-05	2.60e-05	0.00628	8.01e-04	180.0	True	1.28	98.10	1.30
50	350	15	2.75e-05	2.77e-05	0.00667	9.25e-04	168.0	True	1.28	98.24	1.30
50	350	16	1.90e-05	2.89e-05	0.00704	1.06e-03	157.5	True	1.28	98.09	1.30
50	350	17	2.00e-05	3.11e-05	0.00741	1.20e-03	148.24	True	1.28	98.18	1.30
50	350	18	2.00e-05	3.23e-05	0.00776	1.35e-03	140.0	True	1.28	98.10	1.30
50	350	19	2.13e-05	3.46e-05	0.00810	1.51e-03	132.63	True	1.28	98.11	1.30
50	350	20	3.14e-05	5.79e-05	0.00844	1.68e-03	126.0	True	1.28	98.19	1.30

Table G.1: Fram prediction data and results for sub-domain of 50w, 350h.

Width	Height	HALO	$\frac{BExccost}{Sstep}$	$\frac{BComTime}{Sstep}$	$\frac{CoreCalc}{Sstep}$	$\frac{EdgeCalc}{Sstep}$	super-steps	$\frac{CCalc}{BCom} > 1.10$	Runtime	Percentage	H=1 runtime
300	1000	1	1.37e-05	1.44e-05	0.00889	8.99e-06	2520.0	True	22.47	100	22.47
300	1000	2	1.56e-05	2.32e-05	0.01761	7.02e-05	1260.0	True	22.29	99.22	22.47
300	1000	3	2.09e-05	3.26e-05	0.02646	0.00018	840.0	True	22.40	99.67	22.47
300	1000	4	3.04e-05	6.97e-05	0.03526	0.00035	630.0	True	22.45	99.91	22.47
300	1000	5	2.86e-05	5.35e-05	0.04401	0.00057	504.0	True	22.48	100.04	22.47
300	1000	6	3.59e-05	6.35e-05	0.05269	0.00083	420.0	True	22.50	100.11	22.47
300	1000	7	4.14e-05	7.28e-05	0.06131	0.00116	360.0	True	22.50	100.15	22.47
300	1000	8	5.66e-05	8.42e-05	0.06989	0.00154	315.0	True	22.52	100.22	22.47
300	1000	9	5.16e-05	9.38e-05	0.07843	0.00198	280.0	True	22.53	100.25	22.47
300	1000	10	5.71e-05	1.03e-04	0.08687	0.00248	252.0	True	22.53	100.27	22.47
300	1000	11	6.25e-05	1.14e-04	0.09534	0.00305	229.09	True	22.55	100.37	22.47
300	1000	12	6.85e-05	1.23e-04	0.10384	0.00366	210.0	True	22.59	100.53	22.47
300	1000	13	8.22e-05	1.58e-04	0.11197	0.00431	193.85	True	22.56	100.39	22.47
300	1000	14	7.77e-05	1.45e-04	0.12022	0.00502	180.0	True	22.56	100.39	22.47
300	1000	15	8.45e-05	1.53e-04	0.12839	0.00578	168.0	True	22.55	100.37	22.47
300	1000	16	9.31e-05	1.65e-04	0.13653	0.00658	157.5	True	22.55	100.37	22.47
300	1000	17	9.38e-05	1.73e-04	0.14466	0.00744	148.24	True	22.56	100.40	22.47
300	1000	18	9.80e-05	1.84e-04	0.15272	0.00848	140.0	True	22.58	100.50	22.47
300	1000	19	1.16e-04	1.94e-04	0.16074	0.00932	132.63	True	22.56	100.44	22.47
300	1000	20	1.18e-04	2.04e-04	0.16854	0.01034	126.0	True	22.55	100.37	22.47

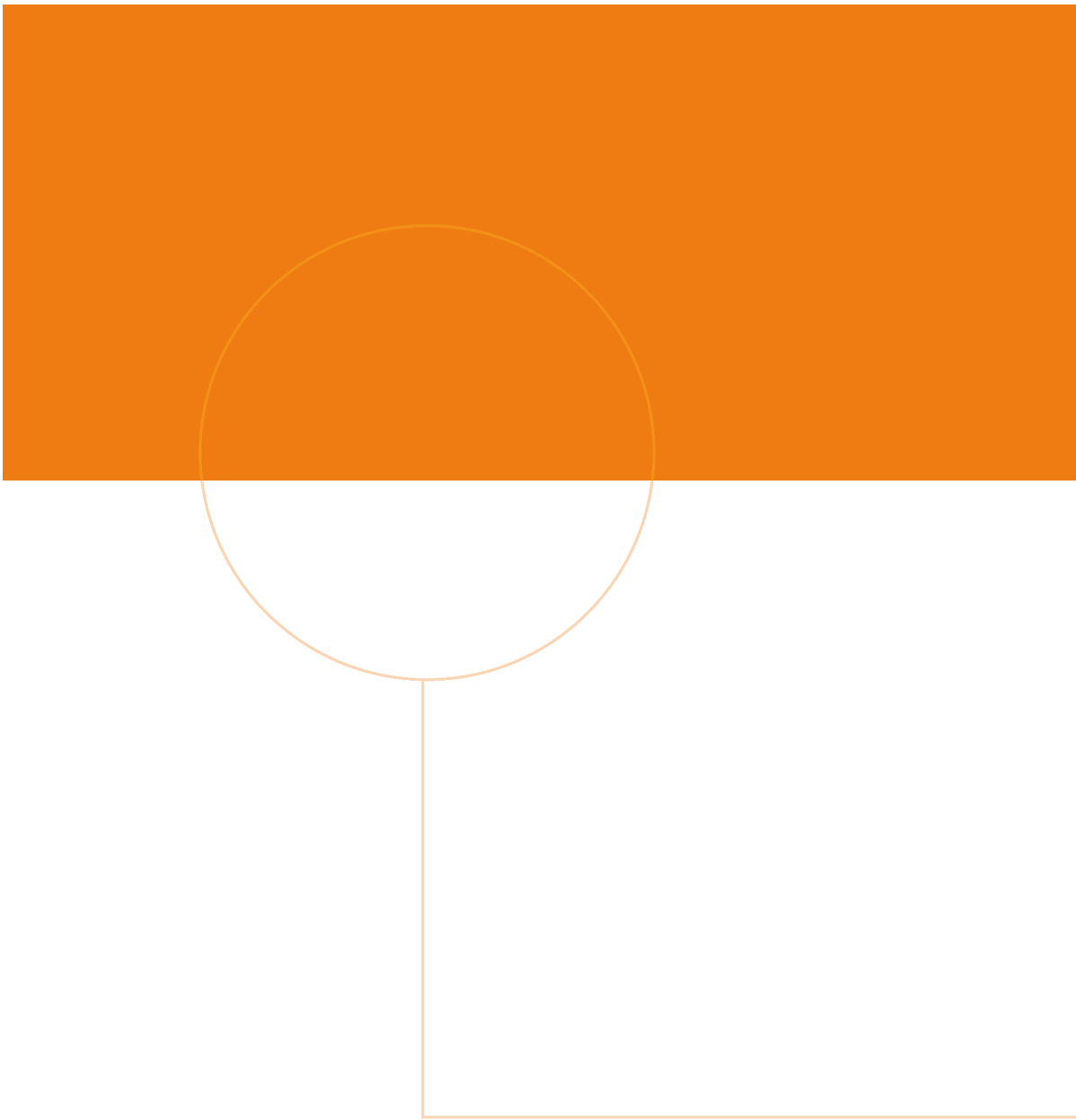
Table G.2: Fram prediction data and results for sub-domain of 300w, 1000h.

Appendix H

Betzy Prediciton Tables

Width	Height	HALO	$\frac{BExccost}{Sstep}$	$\frac{BComTime}{Sstep}$	$\frac{CoreCalc}{Sstep}$	$\frac{EdgeCalc}{Sstep}$	supersteps	$\frac{CCalc}{BCom} > 1.10$	Runtime	Percentage	H=1 runtime
350	500	1	2.09e-05	9.03e-06	0.00559	1.35e-05	2520.0	True	14.18	100	14.18
350	500	2	2.57e-05	1.24e-05	0.01083	6.29e-05	1260.0	True	13.75	97.01	14.18
350	500	3	5.29e-05	4.92e-05	0.01617	1.63e-04	840.0	True	13.76	97.08	14.18
350	500	4	3.38e-05	2.16e-05	0.02141	3.08e-04	630.0	True	13.70	96.64	14.18
350	500	5	4.04e-05	3.07e-05	0.02650	4.98e-04	504.0	True	13.62	96.10	14.18
350	500	6	4.44e-05	4.13e-05	0.03154	7.31e-04	420.0	True	13.57	95.72	14.18
350	500	7	4.72e-05	5.03e-05	0.03646	1.00e-03	360.0	True	13.50	95.25	14.18
350	500	8	5.60e-05	6.71e-05	0.04128	1.31e-03	315.0	True	13.43	94.76	14.18
350	500	9	6.62e-05	8.16e-05	0.04585	1.66e-03	280.0	True	13.31	93.94	14.18
350	500	10	7.28e-05	9.37e-05	0.05042	2.03e-03	252.0	True	13.24	93.37	14.18
350	500	11	1.45e-04	1.11e-04	0.05495	2.47e-03	229.09	True	13.19	93.02	14.18
350	500	12	1.10e-04	1.73e-04	0.05939	2.95e-03	210.0	True	13.11	92.50	14.18
350	500	13	9.47e-05	1.45e-04	0.06365	3.51e-03	193.85	True	13.03	91.95	14.18
350	500	14	1.09e-04	1.66e-04	0.06771	4.21e-03	180.0	True	12.96	91.44	14.18
350	500	15	2.35e-04	1.86e-04	0.07164	5.07e-03	168.0	True	12.93	91.17	14.18
350	500	16	1.14e-04	2.03e-04	0.07572	5.98e-03	157.5	True	12.88	90.88	14.18
350	500	17	1.27e-04	2.20e-04	0.07975	6.83e-03	148.24	True	12.85	90.65	14.18
350	500	18	1.54e-04	2.48e-04	0.08370	7.78e-03	140.0	True	12.82	90.49	14.18
350	500	19	1.62e-04	2.63e-04	0.08707	8.68e-03	132.63	True	12.72	89.72	14.18
350	500	20	1.54e-04	2.75e-04	0.09063	9.61e-03	126.0	True	12.65	89.23	14.18

Table H.1: Betzy prediction data and results for sub-domain of 350w, 500h.



NTNU

Norwegian University of
Science and Technology