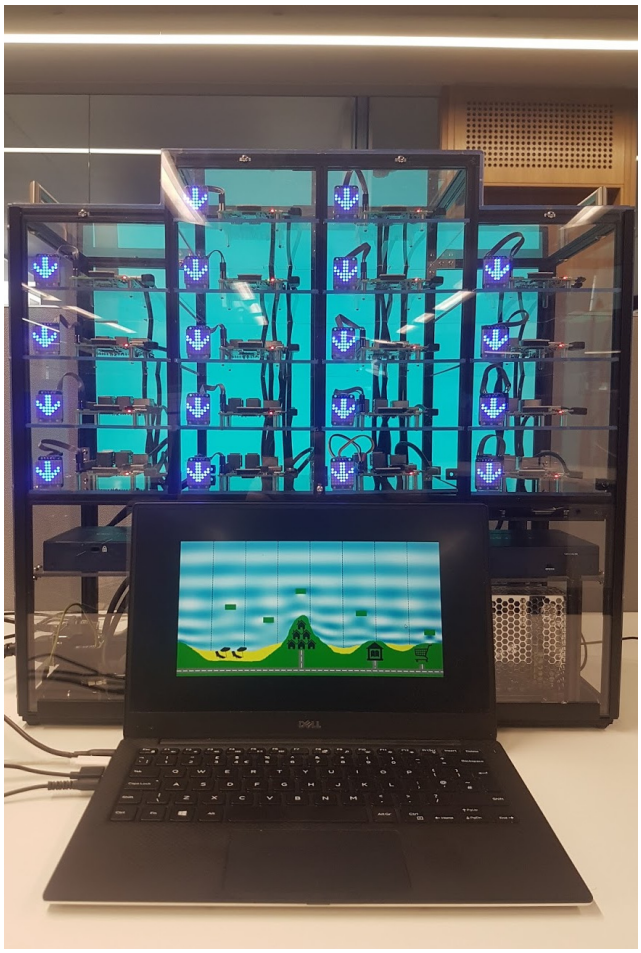
Parallel Computing Demos on Wee Archie

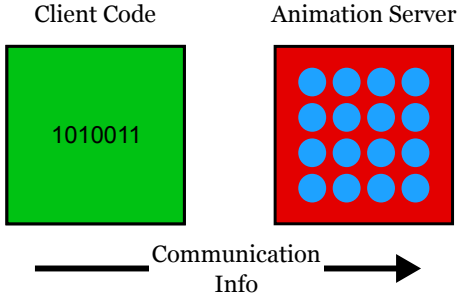
Caelen Feller



I created a framework for visualising parallel communications on a Raspberry-Pi based parallel computer Wee Archie. I used this to create demonstrations illustrating basic parallel concepts, and an interactive coastal defence simulation.

EPCC has developed a small, portable Raspberry-Pi based "supercomputer" which is taken to schools, science festivals etc. to illustrate how parallel computers work. It is called Wee Archie because it is a smaller version of the UK national supercomputer ARCHER. It is actually a standard Linux cluster and it is straightforward to port C, C++ and Fortran code to it. There are already a number of demonstrators which run on Wee Archie that demonstrate the usefulness of running demonstrations on a parallel computer, but they do not specifically demonstrate how parallel computing works.

In this project, I developed a framework for creating and enhancing new, existing and in-development demonstrations that show more explicitly how a parallel program runs. This is done by showing a real-time visualisation on the front-end of Wee Archie, or by programming the LED lights attached to each of the 16 Wee Archie Pis to indicate when communication is taking place and where it is going (e.g. by displaying arrows). All of these visualisations are triggered via the MPI (Message Passing Interface) profiling interface, a standard feature on all modern HPC systems, making this a drop-in solution for most existing code.



I developed visualisations for a tutorial illustrating the basics of parallel communications and a coastline defence demonstration. I also developed a web interface for the tutorials to create a more cohesive user experience. In these demonstrations, I aimed to be able to make it clear what is happening on the computer to a general audience.

Animation Server

To display communications via the LED panels (created by Adafruit), I used the official Adafruit Python library. As such, my visualisations are done in Python.

To allow all demonstrations to share the panels safely, I start a queue on each Pi on a separate background process, where any demo can add an 8px $\times$ 8px image to be displayed on the 8×8 LED panels. I can also give these images properties such as how long to be

displayed and create parcels of images together which form my various animations.

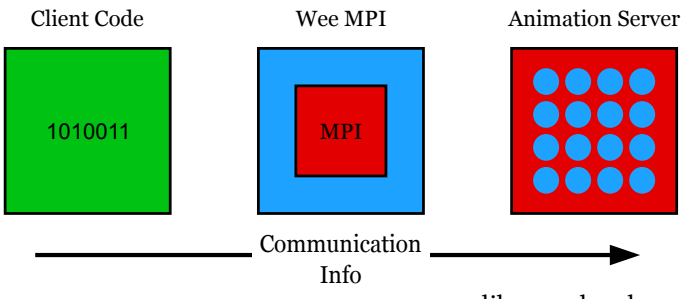
I had two major requirements from my animation system. I need to allow demonstrations developed in any programming language to create an animation, and to be able to coordinate animations between Pis. To do this, I made a web server on each Pi using Flask which will place an animation in queue when you make a certain web request against it. You must provide options for animation length, type, and type-specific options as discussed below.

Point to Point Visualisations

In MPI, an important class of communications are “point to point” communications. These are when a message is passed from one node (here meaning a Raspberry Pi) to another directly. In its most basic form, a node will start to send to some destination, and wait until that destination has begun to receive from the correct source before beginning to transfer the message.

Given the ability to add a sequence of images to a queue, how can I accurately visualise a "send" and "receive" operation between two Pis? My approach was to use a Python “pipe”. When an animation is reached in queue, it will show an “entry” and stop, using the pipe to wait until the server allows it to continue. This allowed me to synchronise and wait on animations between Pis. This accurately replicates the behaviour of MPI messaging.

This behaviour is known as a “synchronous” or “blocking” send. There also exists a “non-blocking” send. The main difference between blocking and non-blocking communications is that using non-blocking communication, the Pi



will continue working, not waiting for the communication to start, and do the communication in the background. This is a more efficient but less safe form of communication, and I provided visualisation for it as it is commonly used.

Other differences are discussed in the MPI standard.[1]

Collective Visualisations

The next major class of MPI communications are “collectives” - when many nodes want to communicate others at once. Say I want to distribute some result to every node - this is done using a broadcast. According to the standard,[1] this will cause every node participating to wait until it has the correct output before continuing, though the exact timing varies.

For clarity and consistency, in collective communication *visualisations*, all participants wait until the communication is done overall before continuing. The implementation is similar to that of waiting in point to point.

There were several other types of communication I visualised (gathering, scattering and reduction), whose implementation is similar. With gathering, the result is collected from each node and stored on one. With scattering, the result on one node is split into small, even parts and distributed to many. With reduction, the result is gathered but an operation is done to it as it is collected such as a sum or product. For more details on these, see the MPI standard.[1]

C and Fortran Framework

While demonstrations for Wee Archie typically use Python for their user interface, they use the C and Fortran languages in computations for performance reasons. Thus, it was desirable to create a wrapper around MPI in these languages which will automatically let the animation server know when a a communication is started.

I did this using the MPI profiling interface. MPI internally refers to functions using “weak symbols”, which allows you to override the functions provided by the library and allows a library developer to call their visualisation and logging code. The framework includes visualisations for most MPI communications, all shown on the next page.

Application-Specific Animations

Often a demonstration will require unique animations, such as a context-appropriate “working” animation, or a visualisation of some communication at a higher level of abstraction than MPI, such as the “haloswaps” of the coastal defence demonstration. Here, the animation server can be contacted using a non-MPI process which directly contacts the server and requires modification of the source code.

Unified Web Interface

As these demonstrations are used for outreach, the surrounding narrative is important for audience engagement. To improve this aspect of the Wee Archie interface is an important aspect of explaining more complex concepts such as parallel communication. In previous demonstrations for Wee Archie, a framework written in Python is used to display a user interface on a connected computer. This starts the demonstration by contacting a demonstration web server running on Wee Archie, which in turn uses MPI to run the code on all of the other Raspberry Pis. It returns any results to the client continuously.

I created an internal website for Wee Archie, but due to the performance constraints of serving complicated websites from a Raspberry Pi while it’s handling so many communications already, I opted to make it a static website - one which does not require processing by the server other than providing the correct files. I did this using the Gatsby framework.

In order to allow the static website to start a demonstration, I wrote my own version of the Wee Archie framework in JavaScript. This framework uses the Axios library to manage communication with the demonstration server, and the React framework to provide a generic demonstration web interface.

Basic MPI Tutorials

This series of tutorials consist of a set of ten demonstrations to be run on Wee Archie and accompanying text. They are aimed at a complete beginner, who does not have programming experience, but can understand the concept of a program doing work, and take them through all of the concepts Wee MPI has to offer.

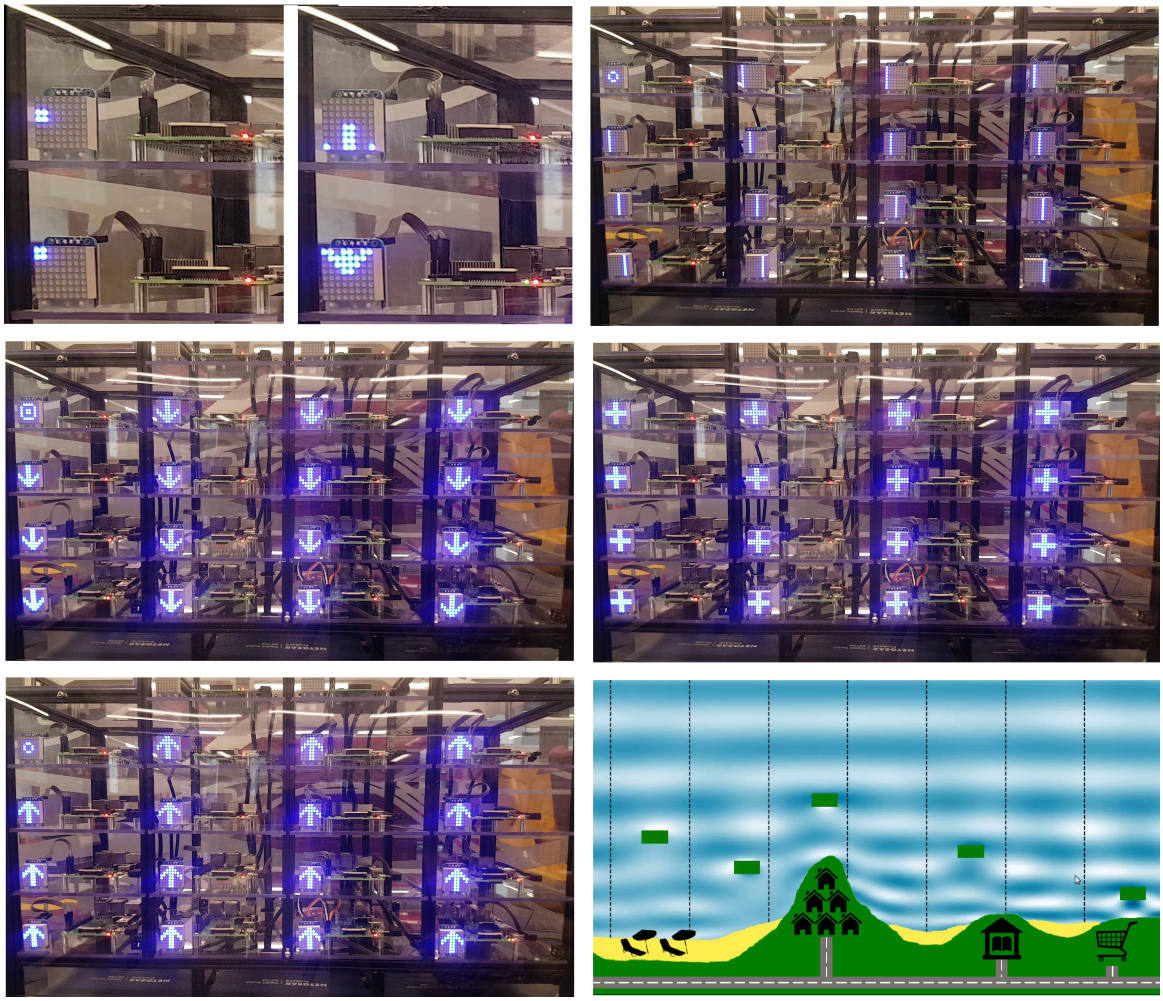


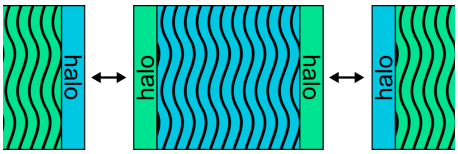
Figure 1: Left: Top: Send and Receive, Middle: Broadcast, Bottom: Gather. Right: Top: Scatter, Middle: Reduce (Sum), Bottom: Wave demonstration in progress.

The first two demonstrate the benefits of parallel computing through the analogy of cooking, showing an embarrassingly parallel problem and then, introducing conflict and making the problem less perfectly parallel, showing the need for efficient communication. The next series go through point to point communications, first showing a loop of blocking and then non-blocking sends and receives. The final set discusses collective communications, demonstrating what each do and showing their animations. It also shows the way that a broadcast could created using point to point communications.

Coastal Defence Demonstration

In the demo, the ocean is broken up into wide horizontal strips.[2] Each strip is processed by a single Raspberry Pi. However, there needs to be some communication so that waves can propagate throughout the simulation.

To do this, after every tick of the equation solver that is run in the simulation, the edges of these strips are exchanged between Pis. This is known as a "haloswap" and is shown by a custom animation. As many thousands of these occur during the simulation, I only show every hundredth haloswap.



Recommendations

The main goal of the project was to create an extensible and easily used framework to visualise parallel communications in any language. By using the animation server-client architecture and the profiling interface, this goal has been accomplished.

As demonstrated in the tutorials and coastal defence simulation, this functions in practice, and as the animations can easily be modified or turned off, there is nothing stopping adoption in future demos.

It also would be an improvement were all demonstrations for Wee Archie done using the web framework, as this would allow users to easily switch between demonstrations, and provide a surrounding explanation. It also allows for novel, interactive visualisations with the use of new features such as WebGL, and the D3 JavaScript library.

References

[1] Message Passing Interface Forum (2015). MPI: A Message-Passing Interface Standard Version 3.1

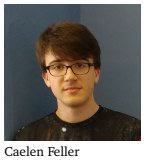
[2] EPCC, The University of Edinburgh (2018). Wee Archie Github Repository

PRACE SoHPCProject Title
Parallel Computing Demonstrators on Wee Archie

PRACE SoHPCSite
EPCC, Scotland

PRACE SoHPCAuthors
Caelen Feller, TCD, Ireland

PRACE SoHPCMentor
Gordon Gibb, EPCC, Scotland


Caelen Feller

PRACE SoHPCProject ID
1907