Communicating Process Architectures 2012
P.H. Welch et al. (Eds.)
Open Channel Publishing Ltd., 2012

A Comparison of Message Passing Interface and Communicating Process Architecture Networking Communication Performance

Kevin CHALMERS
School of Computing, Edinburgh Napier University, UK

Abstract. Message Passing Interface (MPI) is a popular approach to enable Single Process, Multiple Data (SPMD) style parallel computing, particularly in cluster computing environments. Communicating Process Architecture (CPA) Networking on the other hand, has been developed to enable channel based semantics across a communication mechanism in a transparent manner. However, in both cases the concept of a message passing infrastructure is fundamental. This paper compares the performance of both of these frameworks at a base communication level, also discussing some of the similarities between the two mechanisms. From the experiments, it can be seen that although MPI is a more mature technology, in many regards CPA Networking can perform comparably if the correct communication is used.

Keywords. CPA Networking, distributed systems, MPI

Introduction

CPA Networking [1] is an approach to provide channel semantics across distributed communication mechanisms in a manner that is transparent to the user. Currently, CPA Networking has been developed for Communicating Processes for Java (JCSP) [2] and CSP for .NET [3]. In this paper, the communication performance of CPA Networking is compared to that of the Message Passing Interface (MPI) [4]. MPI is a popular mechanism for developing parallel applications that utilize message passing to provide Single Program, Multiple Data (SPMD) solutions. The aim is to discover how well CPA Networking performs in comparison to a similar framework.

The rest of this paper is broken up as follows. In Section 1 the background of this work is presented, examining both CPA Networking and MPI goals. In Section 2 a comparison of some of the operations performed by both frameworks is presented, indicating how certain operations can be achieved in both CPA Networking and MPI. In Section 3 the approach taken to compare both frameworks is presented, and Section 4 presents the results gathered. Finally, Section 5 presents conclusions and future work.

1. Background

In this section, a discussion around both CPA Networking and MPI will be presented. Both of these frameworks have been developed with different aims, although they do share a number of commonalities.

1.1 CPA Networking

The original model for distributed CPA communications used in JCSP comes from the T9000 virtual channel model [5], which was implemented in the original version of JCSP Networking [6]. Although the original JCSP Networking did provide a virtual channel model, it was found to utilize resources poorly [7], and was also highly reliant on Java serialization.

More recent work has tried to move CPA Networking towards a more language-independent standard [1], which has allowed the development of CPA Networking currently for both JCSP and CSP for .NET [3]. Most of this work has focused on developing a standard protocol which allows CPA Networking to be platform agnostic, being separate from both the application and communication layer.

Although CPA Networking has tried to bring together other different CPA inspired platforms, there are still currently a number of different approaches to having distributed CPA applications. occam-π has both pony [8] and a shared communication mechanism with PyCSP [9]. C++CSP also provides a networking mechanism [10]. In all these instances, networking has generally been done to suit either the particular platform, or to solve a particular problem. The CPA Networking approach tries to provide general networked channel mechanisms which operate with as little overhead as possible, but are not optimized for any particular use or platform.

1.2 MPI

MPI [11] is a popular method for developing Single Program, Multiple Data applications (SPMD). From the point of view of this work, MPI utilizes a small server application (called smpd in the implementation of MPI used) which runs on a client machine waiting to receive a message to run a particular application (smpd is not supplied in all versions of MPI, but is part of MPI .NET [12] on which this work is based). When this message is received, the application that is executed via smpd is able to communicate with the other processes in the system using a global communicator mechanism. MPI-2 is the current standard outlined at www.mpi-forum.org – the home of the MPI standard. From the point of view of this paper, when discussing MPI, it is assumed that operations outlined in MPI-2 are meant.

Each process (actively running instance of the application) in the system is assigned a rank. These ranks are then used to send messages directly to a particular process from another process. As such, MPI adopts a coarser grained implementation of communication, implementing application instance to application instance communication rather than thread level communication. This is contrary to a CPA approach, where processes are of a finer grain, and channels enable more direct communication.

Although initially MPI applications belong to a collective group which can communicate using the global WORLD communicator, it is also possible for groups of processes to communicate together by creating group specific communicators. This allows small groups of processes to operate on problems independently.

MPI does have the advantage of being easier to set up and use in cluster computing scenarios, requiring only that the application to be run is available on all hosts. A command is then sent from a host machine to the others, initializing the MPI infrastructure, and executing the applications in parallel. CPA Networking does not have these features, and is focused on creating a communicating infrastructure.

2. Using MPI and CPA Networking

Although MPI and CPA Networking have been developed in isolation, they provide a number of similarities that can be exploited to provide similar operations. The following section examines how some fundamental constructs in both communication frameworks can be achieved by in the other. All code samples are in C#.

2.1 Choice

One core concept in all CPA frameworks is choice, where behavior is determined by the current system state and a number of guards. In CPA Networking, there is only really one type of guard (all other guards, such as timers, come from the base library), which is a networked input channel. Typically, we can select from a collection of input channels using the following code:

```
Alt alt = new Alt(inputs);
int index = alt.Select();
data = inputs[index].Read();
```

In MPI, the same sort of behavior can be achieved by using the probe command, which simply checks if any input is ready matching the given parameters, waiting until one is:

```
Status status = comm.Probe(Communicator.anySource, 1);
data = comm.Receive<Data>(status.Source, 1);
```

The probe command when used with the parameter anySource will return a status message which contains the actual source that sent the message.

2.2 Broadcast

MPI has a command that allows broadcasting of a single message to all other processes in the group (who must call broadcast at the same time). CPA Networking does not have an equivalent abstraction for broadcasting (the broadcast channel implemented in JCSP [13] would be a bit trickier to implement in a distributed manner). However, the same type of interaction can be implemented using parallel writing to multiple channels.

The disadvantage with using a parallel writer approach is that it does not scale well. Each output channel being broadcast to will require another process to service the channel. For lightweight platforms such as occam-π the overhead is not considered significant. However, for other libraries using operating system threads (JCSP, CSP for .NET, etc.) the overhead per thread can become quite significant.

Another approach that would gain some performance would be taken a collective communication technique as described in [9]. At present, this has not been further investigated as an implementation into CPA Networking.

2.3 Scatter-Gather

MPI also has a scatter-gather mechanism, which allows a single process to send an array of messages, with each process in the communicator receiving one of the messages based on its rank within the communicator. The results from the scatter can then be returned by calling the gather command, which causes the root process (the sender) to wait until all returned values are received.

Similarly to broadcast above, the scatter command can be achieved in CPA

Networking using parallel writes. The gathering can also be achieved using parallel reads. As such, the scatter-gather approach in CPA Networking also suffers from the resource implications for broadcasting, although as scattering and gathering are a sequence of events the resulting resource requirements aren't increased.

2.4 Barriers

MPI and CPA Networking both implement a barrier mechanism, allowing computation to be locked in step if required. However, CPA Networking allows the use of multiple, discrete barriers much like it has discrete channels. MPI, however, utilizes the communicator mechanism as a barrier, and although different communicators can be generated for different groups of processes, the use of barriers is a little more complicated than creating barriers in CPA Networking.

For example, with a CPA Networking barrier, it is possible to both enroll in (engage in an existing collective synchronization) and resign from (disengage from the collective synchronization) existing barriers. This enables the CPA Networking programmer to create a collection of required barriers at the start of the system, and modify as needed. MPI barriers work with the communicator mechanism, and therefore require maintenance of groups of nodes, rather than an individual process being able to enroll or resign at will.

2.5 Indexed Communications

The main goal of CPA Networking is to provide channel based semantics across a communication mechanism. It does this by utilizing a virtual channel indexing solution [1] within the event layer to create a software switch which routes messages to the appropriate channel end at the application layer.

The advantage that this approach provides is that networked channels are generally lightweight, and that a process can happily create as many channels as it wishes to allow direct connection. From the point of view of a process, there is no operational difference between a local channel and a networked channel (although if required, a network channel can perform asynchronous communication).

MPI, on the other hand, has a global communicator mechanism which provides processes with a means of communicating with other processes directly. However, it is possible to provide a more selective mechanism by utilizing the tag system in communication. For example, to send a message to a process, the send command is used:

```
comm.Send<Data>(data, destination, tag);
```

The destination value allows a message to be sent to a particular process. The tag value allows a tag to be attached to the send. The equivalent receive command is as follows:

```
data = comm.Receive<Data>(source, tag);
```

The receive command requires the source to receive the communication from, and also a tag value. If both tag values match, the communication completes, otherwise the application will wait until a communication with the correct tag is received.

Using the tag values as a method to allow a more direct communication to a particular thread would enable an indexed style of communication. As such, it leads to the potential of MPI being used as a communication layer for CPA Networking, with CPA Networking providing a channel abstraction layer. Developing such a mechanism would allow simple

use of CPA Networking applications in cluster computing scenarios, and is examined further in future work. Tag values do require buffering, however, which could lead to further problems.

3. Approach

The aim of the work presented is to compare the messaging performance of CPA Networking and MPI. The goal is not to compare parallel performance of tasks when using the two approaches. In particular, MPI has a number of optimisations for particular tasks [4], which CPA Networking does not. Generally, the two message passing approaches have different goals:

- MPI has is a Single Program, Multiple Data (SPMD) based approach to parallel computing, allowing multiple instances of an application to execute on cluster-like architectures.
- CPA Networking has been developed to provide channel based semantics across a communication medium. Rather than a parallel computing solution, it provides usable abstractions to develop distributed CPA applications in a transparent manner.

Section 2 has illustrated how MPI style behaviour can be obtained in CPA Networking and vice-versa. Therefore, the focus of this work is to analyse the base communication performance, and allow application developers to make informed decisions about the approach to use based on the application goal.

For the approach, data sizes of base 10 are used rather than base 2. This is due to the fact that at the network layer buffers exist of size base 2. As both MPI and CPA Networking have a message overhead on top of the data packet, it was deemed appropriate to try and incorporate the message header size into a reasonable total packet size which would not enforce buffer fill.

3.1 Network Performance

For communication, there are two properties that we are interested in – *latency* and *throughput*. Latency allows us to determine the communication delay for a message using the medium and framework of choice. Throughput is a measure of the amount of data that can be transferred by the communication framework in comparison to the capability of the communication medium. Here, we wish to see that the communication framework does not reduce the throughput, particularly at large data sizes.

As MPI also has a broadcast mechanism, the performance of MPI broadcast will also be compared to a CPA approach to providing broadcast semantics (see Section 2.2). MPI should perform better at broadcast due to lower level optimisations which CPA Networking does not have.

3.2 Communication Stress

As both MPI and CPA Networking are designed to allow inter-process communication from many sources, a further experiment to measure how effective each framework coped with a stressed communication scenario was undertaken. This scenario involved a single server process having eight client processes request work to be undertaken. Each computer in the experiment would run two such client processes, meaning that four client machines were in operation. A fifth machine acted as the server. The architecture is illustrated in Figure 1.

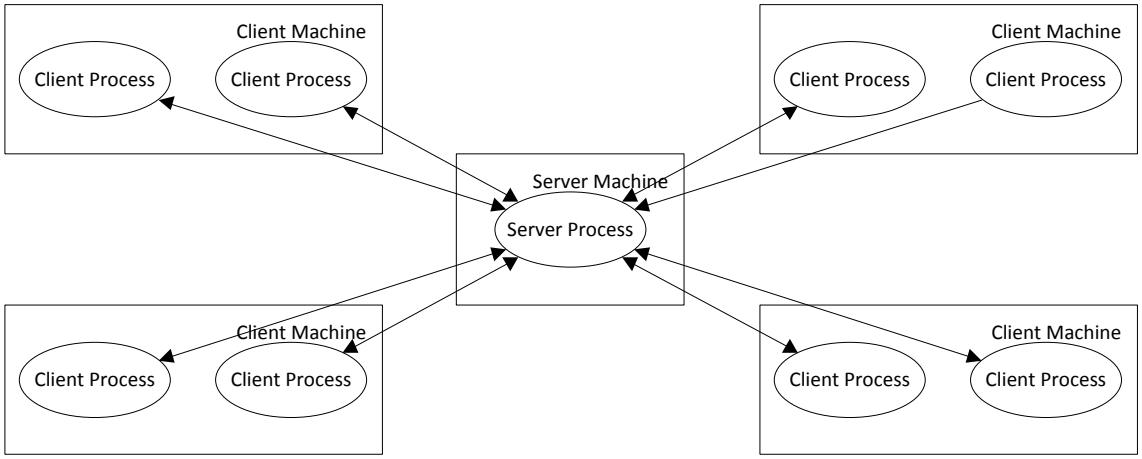

Figure 1: Communication Stress Architecture

The goal of the communication stress experiment is to determine how good the Server Machine is at coping with multiple request-respond communications. The clients request work from the server, which sends a work description to the client, which is processed and the result sent back to the server. The two stress scenarios are described below.

3.2.1 Monte Carlo Pi

Each client performs work to allow the calculation of π using the Monte Carlo method. The algorithm is as follows:

```
IN: NUM_ITERATIONS
COUNT := 0
FOR i in 0 to NUM_ITERATIONS - 1
  X := random 0.0 to 1.0
  Y := random 0.0 to 1.0
  DIST := √(X * X + Y * Y)
  IF DIST <= 1.0
    COUNT := COUNT + 1
OUT: 4.0 * (COUNT / NUM_ITERATIONS)
```

Monte Carlo Pi allows the calculation of π by generating random points in a square, and determining whether they are within a circle (or quarter of) within the square. The ratio can then be used to determine the fraction of the area of the square taken up by the circle. When multiplied by 4, this provides us with an estimate of π.

The algorithm allows the number of iterations to be modified, allowing the processing time to be adjusted. This allows the ratio of processing time to I/O time to be influenced. Initially the experiments will create work packets which have a longer I/O time than processing time, allowing the server to constantly be servicing work requests. The NUM_ITERATIONS value of the work will then be increased, until I/O time becomes marginal in the overall computation time. The work packets themselves will only be a few bytes in size, meaning that each work packet can be sent in a single TCP/IP packet. However, smaller NUM_ITERATIONS will lead to more packets being sent, thus increasing the overall system computation time.

3.2.2 Request-Respond versus Scatter-Gather

MPI encourages the use of a scatter-gather mechanism when developing parallel applications, whereas typically in CPA style applications a request-respond approach is taken, using alternation to select an incoming request, and servicing accordingly. Both MPI and CPA Networking can be made to operate in both manners (see Section 2), and therefore comparing both frameworks for both styles of operations will provide insight into how well each performs in a distributed work scenario.

3.3 Experimental Platform

The results presented were collected using machines of the following specification:

CPU – Intel Core Duo E8400 3.0 GHz (no hyper-threading)
Memory – 2 GB
Operating System – Microsoft Windows 7 32-bit
Software – .NET 4.0

The machines are each connected via a 100 Mbit/s switched Ethernet network. The machines are not set up using any special cluster tools, as this may provide further optimizations for MPI over CPA Networking. For MPI, each machine runs smpd and spawns processes when instructed to. For CPA Networking, two individual programs are run on each machine.

4. Results

In this section the results of the experiments outlined in Section 3 are presented. These results represent the communication performance and overhead of both MPI and CPA Networking. For these experiments, the .NET version of CPA Networking is used [3]. Work has already shown that JCSP and CSP for .NET are comparable in network communication time. The reason that CSP for .NET was chosen over JCSP is that a more recent version of MPI was available for .NET, MPI.NET [11][12], which runs using Microsoft's HPC Cluster Pack SDK. Also, MPI.NET is a wrapper for .NET around Microsoft MPI (MS-MPI), rather than a reimplementation in .NET. More recent Java implementations of MPI like platforms [15] have utilized Java RMI rather than wrapping an already optimized MPI implementation.

4.1 Network Latency

Network latency refers to the time taken to send a single packet using the communication mechanism of choice. The method used to measure latency is to perform a simple ping-pong test, giving the time to send a message back and forth between two hosts. The ping-pong time is then divided by two to provide the network latency. The results from this experiment are presented in Figure 2. The Network results represent the performance of using standard socket based communication for the experiment. 100 runs of the experiment where performed, and for each, the mean of 10,000 iterations was taken. The results presented are the mean of the 100 runs. The MPI results were collected using the standard Send paired with the standard Receive.

For Network, MPI and Async CSP results, the latency time is similar. Only the standard CSP results are different, and this is due to the synchronization that takes place in the CPA Networking layer.

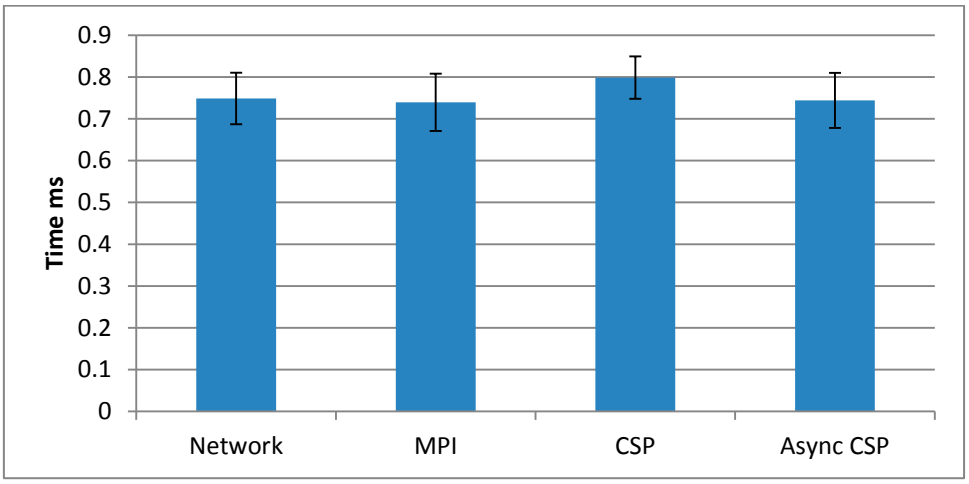

Figure 2: Ping-Pong Time

Of interest are the MPI results, as MPI also has a synchronization mechanism during send and receive communication. This should make MPI similar to the standard CSP results. MPI can have some network optimisations built in by the implementer. As CPA Networking is designed to allow communication across various mediums, there are no network level optimisations.

From the results, the network latency can be measured at 0.375 ms for the network itself, MPI communication and asynchronous CSP style communication. Synchronous CSP communication is measured at 0.4 ms.

4.2 Network Throughput

Network throughput is the measure of actual bandwidth achievable by the communication middleware. Ideally, the performance should reach the capabilities of the network (100 Mbit/s), although latency will have some impact.

For throughput, both standard point-to-point communication (client-server) and broadcast to eight machines (see Figure 1) will be measured. The Mbit/s for the broadcast test will reflect the total amount of data transferred upon the network. For these experiments, 100 tests were performed, each with the mean of 100 iterations gathered. The mean of the 100 tests are presented. Again, standard MPI Send and Receive operations where used.

4.2.1 Standard Communication

The results for standard point-to-point communication are presented in Figure 3. Again, Network represents a baseline gathered from standard socket based communication. The data size is the amount of bytes sent in a single communication.

MPI shows performance similar to the baseline, as do Async CSP results. Standard synchronous CSP results start off poorly in comparison, but are more comparable for larger data sizes. From the results gathered, it can be seen that MPI is a useful point-to-point communication mechanism when performance is important, with asynchronous CPA Networking communication also being useful. Synchronous CPA Networking communication is not good for performance, although the addition of synchronous communication has other benefits.

Figure 4 presents the results of point-to-point communication using a ping-pong style of interaction. Rather than measuring the time taken to send the data, the ping-pong interaction measures time taken to send and receive back a packet of the given size.

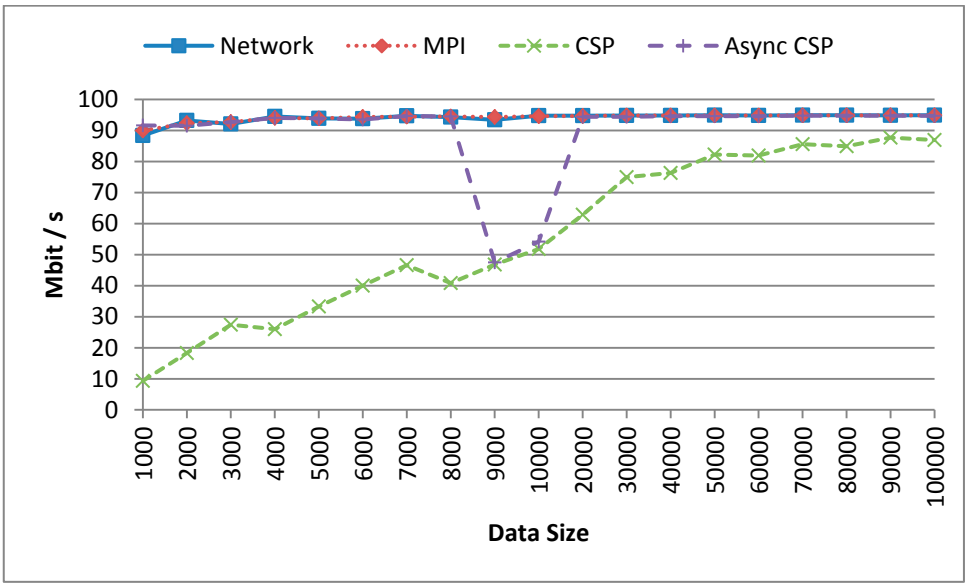

Figure 3: Point-to-Point Network Throughput

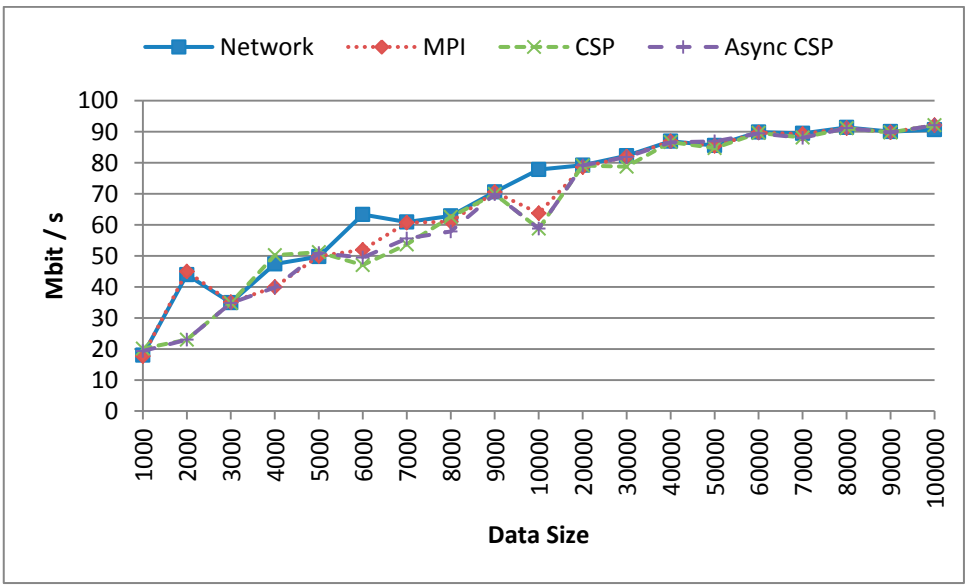

Figure 4: Point-to-Point Send-Receive Network Throughput

For small message sizes, the results show some initial disorder, due to the ping-pong style of interaction (where data is being copied in and out of buffers repeatedly). However, at large enough data sizes (>20000 bytes) we see that neither approach has any difference in overall performance. Synchronous CPA Networking gains as the acknowledgement packet sent back to the writer is followed by the returning data packet, negating any initial loss of performance.

From these results, we can gather that in request-respond interactions both approaches are comparable, particularly when large amounts of data are involved. This observation will have some impact on the stressed communication results presented in Section 4.3.

4.2.2 Broadcast Communication

Figure 5 presents the results gathered using a broadcast approach to communication. In this instance, a message of the data size indicated is broadcast to eight other machines. The Mbit/s value is therefore calculated based on sending the amount of data to eight machines (so actual data sent is 8 times the Data Size value).

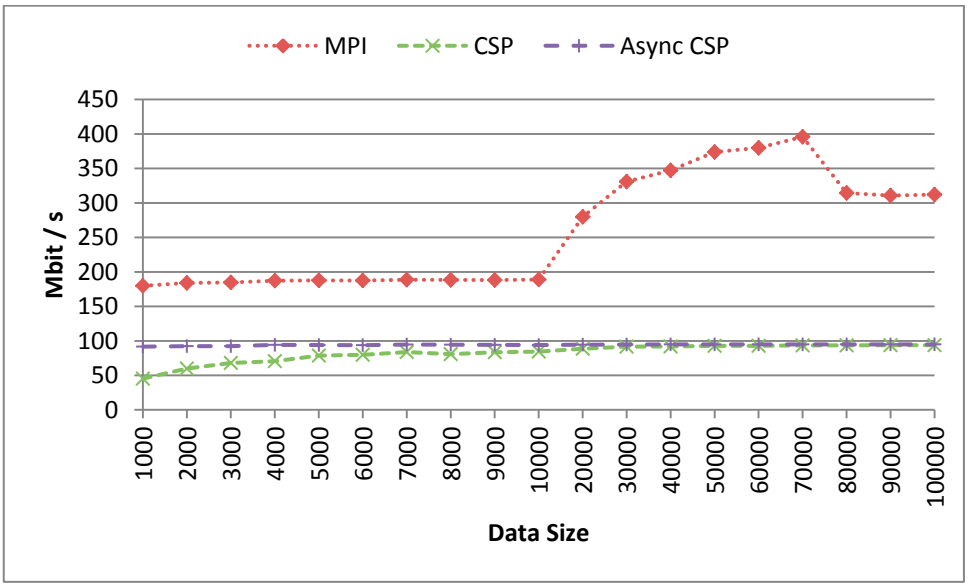

Figure 5: Broadcast Network Throughput

MPI performs far better in broadcast tests that CPA Networking, performing better than the base network performance (100 Mbit/s). This result illustrates that MPI is using lower level optimizations to allow better performance. MPI has performance that at one point comes close to 4 times the underlying capability, showing that the performance is aggregated (e.g. we gain the performance of all four machine connections). CPA Networking cannot break the 100 Mbit/s value, and the results are similar to standard point-to-point communication seen in Figure 3. Asynchronous CPA Networking should be able to perform at any level MPI is capable of if there is no network optimization.

From these results, we can see that if the application style utilizes a broadcast mechanism, then MPI is a better approach. Although CPA Networking can simulate the effect, it does not perform as well.

4.3 Stressed Communication

The stressed communication tests how well a single machine deals with distributing work to others. The goal is to test how well MPI and CPA Networking cope when dealing with a high communication load in comparison to computation load.

4.3.1 Optimal and Sub-optimal Time

Depending on how well the communication mechanism manages stressed communication, the value will fall within a certain range defined by the optimal and sub-optimal performance. For optimal performance, the total communication time for the test is divided evenly amongst the number of worker processes (eight). For sub-optimal, the total communication time is not divided amongst the worker processes.

The calculation for sub-optimal performance is as follows:

$$\frac{computation\ time}{number\ of\ processes} + communication\ time$$

For optimal performance, the calculation is:

$$\frac{computation\ time + communication\ time}{number\ of\ processes}$$

From the results presented in Figure 2, we know that a small ping-pong interaction takes approximately 0.75ms. The packets used in the tests are small (less than 20 bytes), and therefore this is a suitable number for working out the communication time. The total communication time will therefore be:

$$communication\ time = 0.75ms \times number\ of\ packets$$

For computation time, we determine how long it will approximately take for the eight processes to perform 1 billion iterations of Monte Carlo π. A benchmark of the machines indicates that they can perform approximately 4.85 million Monte Carlo π iterations per second. This allows us to calculate the total computation time as:

$$computation\ time = \frac{\left(\frac{1 \times 10^9}{4.85 \times 10^6}s\right)}{8} = 25773ms$$

The optimal and sub-optimal times for the tests are presented in Table 1.

Table 1: Optimal and Sub-Optimal Times

Iterations Per Packet	Number of Packets	Communication Time (ms)	Computation Time (ms)	Optimal Time (ms)	Sub-Optimal Time (ms)
1×10^3	1×10^6	750000	25773	119523	775773
1×10^4	1×10^5	75000	25773	35148	100773
1×10^5	1×10^4	7500	25773	26711	33273
1×10^6	1×10^3	750	25773	25867	26523
1×10^7	1×10^2	75	25773	25782	25848

The values chosen allow a direct comparison between approaches which do not distribute communication optimally, against others that do. When the number of iterations per packet is low, the communication time dominates both optimal and sub-optimal times, although sub-optimal with a clear disadvantage. For packets with a large number of iterations, the communication time has little influence on the total time for optimal and sub-optimal.

4.3.2 Request-Respond Approach

Figure 6 presents the results for using a request-respond approach to distributing the work, utilizing alt in CPA Networking and probe in MPI.

For request-respond style interactions, both MPI and asynchronous CPA Networking perform near optimal. MPI actually performs better than optimal for larger number of iterations, although the optimal value is an approximation. Synchronous CPA Networking performs close to sub-optimal, particularly at the small iteration value.

4.3.3 Scatter-Gather Approach

Figure 7 presents the results for using a scatter-gather approach to distributing the work. Only synchronous CPA Networking results are shown as asynchronous are similar.

Interestingly, MPI performs less optimally when using scatter-gather, whereas synchronous CPA Networking performs more optimally, and initially better than MPI. Asynchronous CPA Networking performs less optimal as well, but not as bad as MPI.

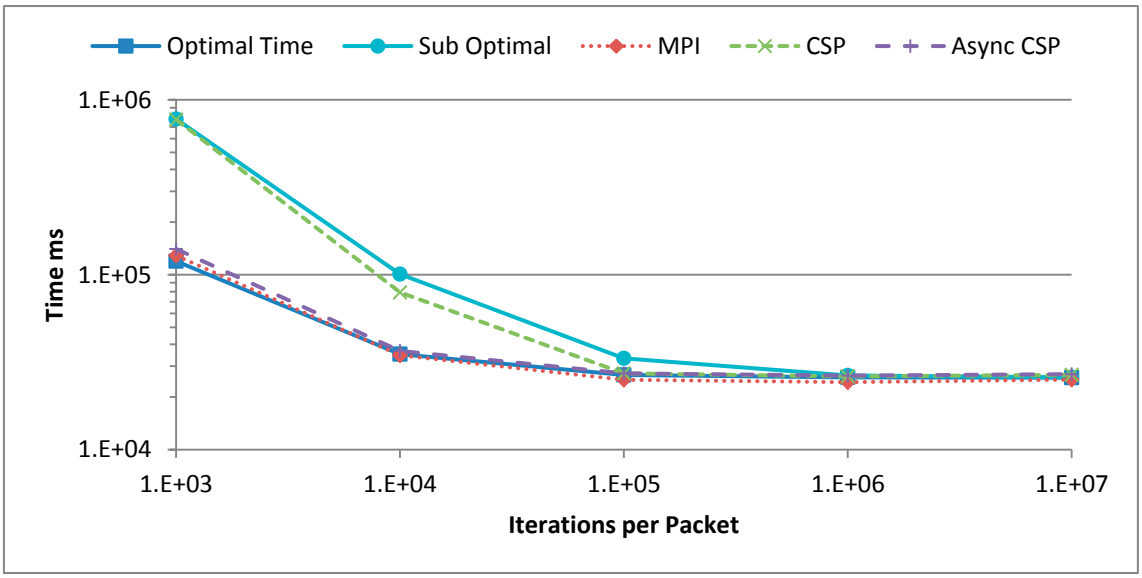

Figure 6: Request-Respond Monte Carlo Pi

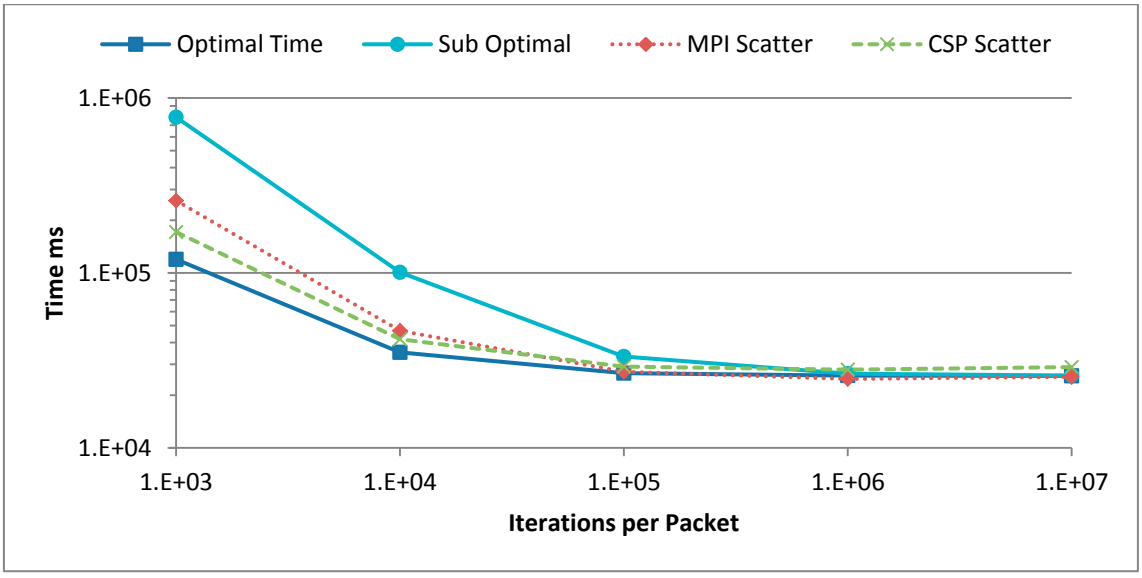

Figure 7: Scatter-Gather Monte Carlo Pi

The improvement in synchronous CPA Networking results is expected, as the system now performs eight write operations at once, rather than one at a time (where the synchronization causes an issue). At larger iteration values performance is less optimal, but this will be the result of the parallel readers and writers in use for the scatter-gather operations.

Considering the MPI results, less optimal values are harder to explain. In the first instance, it may simply be because with scatter-gather the operation can only operate as fast as the slowest processor. However, it would be expected that the same results would be seen in the CPA Networking results, and all machines in the experimental framework were of the same specification. Some of the lower level optimizations may cause the slowest member problem to be more apparent in MPI.

5. Conclusions and Future Work

This paper has compared the base communication of CPA Networking and MPI, and has found that in many circumstances the performance is comparable, particularly when asynchronous CPA Networking communication is taken into account.

In both instances, it was found that MPI and asynchronous CPA Networking communication performed close to the base network performance for latency and standard network throughput. Synchronous CPA Networking communication performs poorly in comparison, until large enough data packets are sent.

MPI does have an advantage when dealing with broadcast messages, and therefore if the application requires broadcast semantics it is recommended that MPI is used over CPA Networking. Also, in scatter-gather scenarios, MPI is probably a better choice, although using a CPA approach (request-respond) does appear to be a better method. Asynchronous CPA Networking does compare favorably in these situations, but the simplicity in doing MPI applications in this manner has an advantage. For scatter-gather, CPA Networking also has a higher thread requirement.

5.1 MPI or CPA Networking?

In Section 3 the two different aims of MPI and CPA Networking where provided:

- MPI has is a Single Program, Multiple Data (SPMD) based approach to parallel computing, allowing multiple instances of an application to execute on cluster-like architectures
- CPA Networking has been developed to provide channel based semantics across a communication medium. Rather than a parallel computing solution, it provides usable abstractions to develop distributed CPA applications in a transparent manner

From the results presented, these two statements still stand. If the application domain requires SPMD parallel computing, then MPI will provide a better approach. The setup of an MPI style application is easier in these circumstances, and the communication infrastructure is created automatically. However, if the application domain requires more complexity than SPMD provides, then CPA Networking can provide an infrastructure which has comparable performance, particularly if asynchronous communication is used carefully.

5.2 MPI Links for CPA Networking

At present, CPA Networking has been built with the belief that communication streams would be the underlying mechanism of choice. However, the observations made in this work indicate that it may be possible to utilize MPI as a communication mechanism for CPA Networking, allowing CPA Networking to be utilized more in cluster computing scenarios. Making this change would require some reengineering of CPA Networking, but there may be advantages to doing so.

Another possible advantage in using MPI as a base layer in CPA Networking would be in utilizing the mobile process and channel mechanisms. It would be fairly trivial with an MPI layer to implement a mobile agent type system on a cluster with little overhead. Mobility is a definite direction of interest for an MPI controlled CPA Networking application.

References

[1] K. Chalmers, "Investigating Communicating Sequential Processes for Java to Support Ubiquitous Computing.," PhD Thesis, Edinburgh Napier University, 2009.
[2] P. H. Welch, "Process oriented design for Java: concurrency for all," in *International Conference on Parallel and Distributed Processing Techniques*, 2000, pp. 51–57.
[3] K. Chalmers, "Performance of the Distributed CPA Protocol and Architecture on Traditional Networks," in *Communicating Process Architectures 2011*, 2011, pp. 227–242.
[4] MPI Forum, "MPI," *MPI Forum*. [Online]. Available: http://www.mpi-forum.org/.
[5] D. May, R. Shepherd, and P. Thompson, "The T9000 transputer," in *Computer Design: VLSI in Computers and Processors, 1992. ICCD '92. Proceedings., IEEE 1992 International Conference on*, 1992, pp. 209 –212.
[6] P. Welch, J. Aldous, and J. Foster, "CSP Networking for Java (JCSP.net)," in *Computational Science — ICCS 2002*, vol. 2330, P. Sloot, A. Hoekstra, C. Tan, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2002, pp. 695–708.
[7] K. Chalmers, J. Kerridge, and I. Romdhani, "A Critique of JCSP Networking," in *Communicating Process Architectures 2008*, 2008, pp. 271–291.
[8] M. Schweigler and A. T. Sampson, "pony - The occam-pi Network Environment," in *Communicating Process Architectures 2006*, 2006, pp. 77–108.
[9] J. M. Bjørndalen and A. T. Sampson, "Process-Oriented Collective Operations," in *Communicating Process Architectures 2008*, 2008, pp. 309–328.
[10] N. C. C. Brown, "C++CSP Networked," in *Communicating Process Architectures 2004*, 2004, pp. 185–200.
[11] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 3241, D. Kranzlmüller, P. Kacsuk, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2004, pp. 353–377.
[12] J. Willcock, A. Lumsdaine, and A. Robison, "Using MPI with C# and the Common Language Infrastructure," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 7–8, pp. 895–917, 2005.
[13] P. H. Welch, N. C. C. Brown, J. Moores, K. Chalmers, and B. H. C. Sputh, "Integrating and Extending JCSP," in *Communicating Process Architectures 2007*, 2007, pp. 349–369.
[14] D. Gregor and A. Lumsdaine, "Design and implementation of a high-performance MPI for C# and the common language infrastructure," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, New York, NY, USA, 2008, pp. 133–142.
[15] A. Nelisse, J. Maassen, T. Kielmann, and H. E. Bal, "CCJ: object-based message passing and collective communication in Java," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 3–5, pp. 341–369, 2003.