PERFORMANCE OF NETWORK SUBSYSTEMS FOR A TECHNICAL SIMULATION ON LINUX CLUSTERS

Andreas Boklund, Christian Jiresjö, Stefan Mankefors-Christiernin, Nima Namaki, Linn Christiernin, Micael Ebbmar
Department of Technology, Mathematics and Computer Science
University of Trollhättan/Uddevalla
P.O. Box 957, SE-461 29 Trollhättan
Sweden
{andreas.boklund, christian.jiresjo, stefan.christiernin, linn.christiernin}@htu.se
nima.namaki@bredband.net, micael.ebbmar@student.htu.se

ABSTRACT
The focus of this paper is to evaluate the impact of the network infrastructure on the total execution time for non-linear industrial finite element simulations and to investigate the applicability of micro-benchmarks on this area of applications. For this purpose we selected an industrial strength simulation application which utilizes the MPI message passing library. Three network technologies (Gigabit Ethernet, SCI and InfiniBand) were evaluated from both a micro-benchmark and an application level perspective. The overall result is that although there are applications that benefit from being able to send vast amounts of small messages fast as well as other applications that need to move several Gigabytes of data thus utilizing either the low latency or high bandwidth of specialized network interconnects, non-linear FEM analysis is not one of them.

KEY WORDS
High Performance Computing and Networks, Cluster Computing, Network Performance

1. Introduction
The amount of Linux based Commodity Off The Shelf (COTS) clusters used for demanding technical applications has grown exponentially since their introduction in the early 1990’s [1, 2]. At that time the concept was state of the art, and only used in scientific environments. Today numerous companies and organizations develop products aimed for the Linux clustering platform, hence, performance and scalability has become a vital competitive factor. Thus performance optimization of an “application” towards the “Linux cluster” platform is an important and extremely challenging task since no two clusters are alike [3]. This makes, in some respect, application (solver) optimization a potential parallel nightmare.

The grand question with optimization is though; “optimised over what”? In a parallel Linux cluster environment there are a huge amount of different unknown parameters. Nevertheless, as pointed out by, among others, Sterling et. al. [1, 4] the inter process communication is an especially problematic area of major importance. It does not only apply to Beowulf clusters [1] but to all high performance computer (HPC) architectures and has been the subject of research over the last 40 years [5, 6, 7]. The major difference between Linux clusters and traditional HPC computers is that the hardware interconnect used in Linux Clusters differs from machine to machine while they stay virtually the same through the entire life span of a commercial HPC product line.

There are three major approaches to enhance the inter process communication performance; solver algorithm (distributed programming model), high level message passing api and network infrastructure. Implementing a new solver algorithm is often not an option, since most industrial software’s are closed source (as opposed to open source) and usually they use a generic algorithm that performs reasonably well on most architectures. The message passing environment is used as a contract [8] between the software and the Linux cluster hardware (Figure 1). There are several message passing environments/models that are widely used in both traditional HPC computers and Linux Clusters [9]. This diversity complicates the situation both for the software engineer (Figure 1: layers 4-5) and the cluster architect (Figure 1: layers 1-3 and sometimes 4). Nevertheless, optimal performance can be reached if the software and the cluster is developed in parallel. Although this might sound good in theory it is practically useless, mostly since the same software is used on hundreds or thousands of different installations and that the majority of all clusters are running several different applications. The approach left to influence for better performance is then the network infrastructure.
Figure 1: The five layer model used by most high performance software implementations

Hence, the main focus of this paper is to evaluate the impact of the network infrastructure on the total execution time for technical simulations and to perform a first study of the applicability of micro-benchmarks on this suite of applications. For this purpose we selected a non-linear finite element (FEM) solver implemented in an industrial strength application for traffic generation which utilizes the ScaMPI message passing library. The network technologies evaluated in this paper are Gigabit Ethernet, Scalable Coherent Interface (SCI) and InfiniBand, which are three of the frequently used network technologies for Beowulf style cluster systems.

In the following section the simulation application and the network technologies will be further presented in combination with the message passing library. Section 3 elaborates on the principles of evaluation, followed by a performance analysis in Section 4. Finally the paper is concluded with a discussion of the “Impact on Application performance”.

2. Technical Foundation

For the evaluation we choose a non-linear FEM simulation application [10] a program that is representative for an industrial strength, demanding technical application. The software at hand is one of the leading non-linear FEM software packages and is widely used in the aerospace and automobile industries for simulation and technical calculations [11]. Areas of applications include but are not limited to, welding, stretching, heat transfer, etc. It is being utilized by a large amount of companies around the world within areas such as car-, airplane-, space-, tools- and manufacturing of industrial equipment industry. The simulation execution times vary between a few minutes to several days or weeks.

Two reference models were chosen as foundation for the evaluation. The first one, Simulation1 (S1) is a basic theoretical model which was made with good scalability in mind and the second simulation (S2) is a real world welding simulation model used by a large aerospace corporation.

2.1. Message Passing Environment

The most common message passing environment in the technical simulation industry is MPI [12]. MPI is a specification of a message passing library that is maintained by the MPI Forum [13] and it is available for most HPC computers. The ScaMPI library [14] used in our evaluation is an implementation of the MPI standard and started out as MPI for SCI but has over the years evolved to support SCI, Ethernet Myrinet and InfiniBand. It can either run directly on the wire or over protocols like TCP/IP. ScaMPI will therefore, in theory, work with any interconnects that supports TCP/IP. The main advantage in this evaluation is that the ScaMPI [15] implementation runs on top of them without any TCP/IP support, thus, reducing the overhead.

2.2. Network Technologies

Ethernet is an early and well known standard for Local Area Network (LAN) communication and is specified in IEEE standard 802.3. Ethernet has a low cost due to its large install base and it is easy to upgrade to new Ethernet versions since the frame format and size have been constant over versions. However, since it was developed for generic network communication tasks its main drawback is the relatively high latency [16].

Scalable Coherent Interface (SCI) is a modern (ANSI/IEEE standard 1596-1992) CPU-Memory-I/O bus supporting distributed multiprocessor systems, designed to achieve high bandwidth in combination with very low latency (less than 1 µs). SCI uses a bus-like service with point-to-point unidirectional links, instead of a traditional parallel bus. Three different topologies are supported: ring, star or mesh (2-d or 3-d torus). To efficiently utilize the underlying hardware a packet transmission protocol is defined by SCI. It uses small packets to transport data and commands between nodes with very low latency. SCI has been evaluated in a number of studies [17, 18] and a comparison between SCI and Ethernet can be found in [19].

InfiniBand Trade Association consists of over 180 member companies (Intel, Sun, etc.) InfiniBand is a network for inter process communication (IPC), processing and I/O nodes communication. In contrast to other HPC architectures, InfiniBand was not designed for a specific application but for a broad set of applications and hence it can have various uses: storage interconnect, high performance cluster networking, shared memory etc. Furthermore, InfiniBand consists of high-speed point-to-point links, using switched fabric as interconnects between I/O or processing nodes. In contrast to standard buses which send bursts of bits in parallel, InfiniBand specifies a serial bus.
The main advantage is fewer electrical connections which saves manufacturing costs and improves reliability. InfiniBand supports link rates of 2.5, 10 and 30 Gbps. For further information please refer to [20, 21, 22]. Of the three architectures discussed, Gigabit Ethernet is the most commonly used, especially for heterogeneous Linux clusters mainly due to its low price and ease of use. The remaining two technologies, SCI and InfiniBand both support Gigabit speeds, shared memory and a variety of topologies. Table 1 provides a short theoretical comparison of the network technology specifications.

<table>
<thead>
<tr>
<th>Table 1: Theoretical Comparison of the Three Network Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ethernet</strong></td>
</tr>
<tr>
<td><strong>Media</strong></td>
</tr>
<tr>
<td><strong>Topology</strong></td>
</tr>
<tr>
<td><strong>Bandwidth</strong></td>
</tr>
<tr>
<td><strong>Latency (µs)</strong></td>
</tr>
<tr>
<td><strong>MTU</strong></td>
</tr>
<tr>
<td><strong>Cost NIC</strong></td>
</tr>
</tbody>
</table>

3. Evaluation Principles and Setup

The aim of the evaluation is to determine the impact of the performance of the network subsystem based on its traffic pattern and to see which network technology yields the best overall performance and thus is the preferred choice. It should be noted that layers 2-5, in Figure 1, are constant throughout the entire evaluation.

The evaluation was conducted on a typical Beowulf cluster [2] consisting of 16 identical 2.4 GHz Intel Xeon compute nodes equipped with 2 GB RAM. The communication hardware specifications are listed in Table 2, below. The network interfaces specified in Table 2 are all of different bandwidth and latency as earlier noted (see Section 2).

<table>
<thead>
<tr>
<th>Table 2: Technical characteristics of computer cluster communication hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
</tr>
<tr>
<td>PC1 bus</td>
</tr>
<tr>
<td>Ethernet</td>
</tr>
<tr>
<td>SCI</td>
</tr>
<tr>
<td>InfiniBand</td>
</tr>
</tbody>
</table>

The application dependant part of the evaluation was performed over multiple sets of nodes ranging from one single computer to all sixteen. Simulation S1 was executed on [1,2,4,6,8,10,12,14,16] nodes while S2 were executed on [1,2,3,4,5,6] nodes. When utilizing more then 6 nodes the S2 models execution time did not decrease application uses a well defined set of MPI calls, which can be found in Table 3. Kurmann et. al. [24] makes the following definition:

“The primary aim of a micro-benchmark is to analyze an isolated (communication) primitive to gain architectural insight into the bottlenecks.”

Kurmann et. al. [24]

The vast majority of the function calls are generated by MPI_Get_count and blocking MPI_Send/MPI_Recv calls. A micro-benchmark is then suitable to use to measure the bandwidth and latency sustained by the “MPI_Send and MPI_Recv” operations over different interconnects and message sizes to determine how these (network subsystem) characteristics affect the performance of the simulation.

There are several MPI benchmark suits [25] that are widely used to benchmark MPI-performance [26, 27, 28]. In this paper we choose to evaluate the calls by using a simple ping-pong benchmark which means that one message is sent from node A to node B and then returned from node B to node A. During this exercise the “round trip time” is measured on node A.

According to Vaughan [29] benchmarking tools suffer from inadequate timing functionality. The traditional solution to this problem is to perform a high number of repetitions and use the average result. This approach obviously removes the detailed information, especially when the execution time is as fine grained that it can be counted in clock cycles. A good statistical approach was taken by Tabe et. al. [30]. However, this approach is not a viable option when applied to a 3 GHz system and small messages.

To ensure high quality results from the micro-benchmark the transmission of 1000 repetitions were timed and the time per packet calculated, as shown in Figure 2. Hence, minimizing the error due to the above explained measurement problem. This was then repeated 100 times for each packet size and a mean value and a standard deviation was calculated based on these results.

```c
for ( i = 0 ; i < 100 ; i++ ) {
    T_start = time( )
    Send 1000 packages of size S
    T_n = ( time( ) - T_start ) / 1000
}
```

Figure 2: Micro-benchmark setup; send 1000 packages of size S hundred times. T_n equals the time needed to send 1000 packets divided by 1000. The values T_1 to T_n are then used for the calculation of the mean and standard deviation values.
further, it even increased slightly, thus these values are not interesting to include in the study.

During the simulations two sets of data were collected: The number of executed MPI function calls and their execution time. The amount of function calls were collected using the “strace” program while the sustained bandwidth and latency of the MPI_Send and MPI_Recv were written to a log file from ScaMPI verbose mode. Unfortunately the potential measurement errors of the execution time of the MPI_Send and MPI_Recv function calls can not be minimized in the same way as with the micro benchmarks (Section 3, paragraphs 6-8). The very large statistical foundation (almost 4 million calls) should reduce the relative discrepancy, however. This is so since the central limit theorem guarantees a Gaussian like distribution for extensive enough statistics for uncorrelated events, something that in turn ensures a $1/\sqrt{n}$ behaviour for the relative statistical uncertainty. Hence the timing results obtained from the MPI_Send and MPI_Recv function calls should model, quite accurately, how well the simulations utilize the different network architectures.

4. **Performance Analysis**

Turning to the results from the evaluation we start with Figure 3 which contains the sustained (measured) bandwidth for packet sizes $0 < S \leq 256$MB and Figure 4 that contains the latency for the same interval. InfiniBand has the, by far, highest sustained bandwidth with a 730 000 Kbps, while SCI levels out at approximately 117 000 Kbps and Ethernet at 115 000 Kbps.

All tests show an acceptable standard deviation they are generally between 0.04% and 4%. As one might expect the difference is highly correlated with the packet size, hence, the size of the deviation decreases as the packet size increases, this is because the measurement error in microseconds is constant while the total time increases. This might have been avoided if the number of iterations between measurement points had been higher. However, the statistical error is small enough to disregard.

As seen in Figure 4; InfiniBand provided the lowest latency of the three network technologies. There is a large difference in sustained latency between the evaluated networking technologies. Nonetheless, these numbers, as shown later, does not correlate with the sustained throughput of the real world simulations.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>S1</th>
<th>Nodes</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>458</td>
<td>2</td>
<td>58076</td>
</tr>
<tr>
<td>4</td>
<td>1 752</td>
<td>3</td>
<td>138 384</td>
</tr>
<tr>
<td>6</td>
<td>3 725</td>
<td>4</td>
<td>273 852</td>
</tr>
<tr>
<td>8</td>
<td>5 370</td>
<td>5</td>
<td>341 252</td>
</tr>
<tr>
<td>10</td>
<td>7 723</td>
<td>6</td>
<td>452 299</td>
</tr>
<tr>
<td>12</td>
<td>10 518</td>
<td>14</td>
<td>11 157</td>
</tr>
<tr>
<td>16</td>
<td>13 295</td>
<td>16</td>
<td>13 295</td>
</tr>
</tbody>
</table>

Turning to the real world simulation software, Table 3 contains the number of executed MPI_Send calls for different amount of nodes. It should be noted that each MPI_Send operation has an MPI_Get_count and an MPI_Recv counterpart. Figure 5 and 6 are based on the average execution times from all the executed calls for each collection of nodes respectively. As can be seen the execution time of the MPI_Send calls are lower for Infiniband then for Ethernet and SCI which correlates well with the findings in Section 4. However, the execution time, of the independent MPI_Send calls does not seem to be a function of the amount of nodes used.
Thus, the simulations are far from utilizing the maximum bandwidth; this is true even when reaching the 16 node limit of the cluster. On the contrary, the send times for the SCI network decreases as the amount of nodes increase. Hence, the more complex topology of SCI is fully utilized.

It should be noted that although the Infiniband was ~4.4 (64byte package) and ~4.2 (512byte package) times faster then the other network technologies in the micro benchmark, it is only a factor of 2.11 times faster then Gigabit Ethernet at 10 nodes when utilized by the real industrial application. Hence, the impact of the lower latency and higher bandwidth does not affect the application as much as theoretically possible. The execution time of the MPI_Recv function increases almost linearly as a function of the amount of nodes, although it peeks at 3 and 5 nodes for SCI and InfiniBand technologies respectively.

The low numbers in Figures 5 and 6 indicate that relatively small messages (packages) are being sent, thus, MPI is utilizing buffered transmissions. Hence the transmission occurs in 3 steps: MPI_Send first copies the message to a buffer, then MPI transmits the buffer to the receiver and finally on the receiving computer MPI_Recv serves the message to the application. For larger packages the MPI_Recv operation would start reading before the entire package has been transmitted over the network.

### 5. Impact on Application Performance

Based on the findings from the evaluation in Section 4 we can conclude that the network impact on applications for technical simulation micro benchmarks and real world application measurements correlate quite well. The network infrastructures impact, \( T_I \), on the overall application performance is now trivial to calculate since we have determined the values of all unknown variables (amount and execution time for the MPI calls). By combining the executing time of the MPI_Send and its corresponding MPI_Recv we get the total execution time of one blocking MPI call, \( T_{MPI} \). When \( T_{MPI} \) is determined we can multiply it with the number of calls made during the simulation, \( N_{calls} \). Thus, we get the following expression:

\[
T_I = T_{MPI} \cdot N_{calls}
\]

Utilizing Equation 1 we can now calculate the difference in execution time (and the preferable network) for different network infrastructures and amount of nodes. The result can be found in Tables 4 and 5. Hence, the most optimal network for these simulations is SCI whilst the second most optimal network is InfiniBand leaving Gigabit Ethernet, quite far behind, in relative terms. Relatively, SCI is as much as 2.3% faster then InfiniBand and 8% faster then Gigabit Ethernet.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Ethernet</th>
<th>SCI</th>
<th>InfiniBand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9.15</td>
<td>8.48</td>
<td>8.67</td>
</tr>
<tr>
<td>4</td>
<td>35.02</td>
<td>32.43</td>
<td>33.16</td>
</tr>
<tr>
<td>6</td>
<td>74.45</td>
<td>68.95</td>
<td>70.51</td>
</tr>
<tr>
<td>8</td>
<td>107.33</td>
<td>99.40</td>
<td>101.65</td>
</tr>
<tr>
<td>10</td>
<td>154.35</td>
<td>142.95</td>
<td>146.19</td>
</tr>
<tr>
<td>12</td>
<td>210.22</td>
<td>194.69</td>
<td>199.10</td>
</tr>
<tr>
<td>14</td>
<td>222.99</td>
<td>206.52</td>
<td>211.19</td>
</tr>
<tr>
<td>16</td>
<td>265.72</td>
<td>246.09</td>
<td>251.66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Ethernet</th>
<th>SCI</th>
<th>InfiniBand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1160.72</td>
<td>1075.00</td>
<td>1099.33</td>
</tr>
<tr>
<td>3</td>
<td>2765.78</td>
<td>2561.52</td>
<td>2619.48</td>
</tr>
<tr>
<td>4</td>
<td>5473.27</td>
<td>5069.06</td>
<td>5183.77</td>
</tr>
<tr>
<td>5</td>
<td>6820.35</td>
<td>6316.64</td>
<td>6459.59</td>
</tr>
<tr>
<td>6</td>
<td>9039.76</td>
<td>8372.14</td>
<td>8561.60</td>
</tr>
</tbody>
</table>

This oversimplified conclusion is for the slightly insightful only superseded by its own insignificance; hence 8% faster network communication during a 4 node Simulation2 job equals ~404 ms and the total execution time of the simulation is 114 minutes and 10 seconds.
However, putting this result in perspective it addresses a very fundamental issue in parallel computing: the perceived importance of latency.

The very concept behind parallelization is to divide the workload of a serial implementation on a single computer over a number of computer nodes running a parallelized version of the same application. Hence a successful implementation means reasonable processor load on each node. But in any modern parallel computing resource, the available RAM memory and computing power of each node corresponds typically at least to 256 MB and a Pentium 4 processor. In order for such a computer node to be both reasonably well used and dependent on latencies, i.e. be waiting on the arrival of certain packages, the implementation has to be close to pathological.

The existence of very special physical and mathematical problems with small, but still extremely vital amounts of information being communicated should of course be acknowledged. But in many other physical applications based on e.g. spectral methods, it is rather the communication of extremely large blocks of information (mega- or gigabyte sizes) and hence the bandwidth that limits the execution time. In the case of industrial technical simulations this is definitively the case most of the times. Still, even for the most special and demanding physical problem, it is only if the information transferred between the nodes is small enough so the communication time is dominated by latency and not bandwidth, and the rate of sending/receiving of these very small pieces of information is so extreme that the accumulated lost time for high latency becomes comparable to the execution time on the compute nodes and it is not possible to increase the time between communication in the program module being place on the compute nodes latency actually will affect the total execution time to any degree. It stands to reason that the occurrence of such problems that inherently prohibit any other kind of implementation should be extremely low. Our particular investigation furthermore shows that our real-life industrial software at hand certainly is not of this category.

6. Conclusion

The results presented in Section 5 are interesting in that sense that they show the danger in using micro-benchmarks or evaluating a subpart at a time without having a proper model to map the findings into. This is since the micro-benchmark showed a performance difference of approximately 400% while the subpart evaluation of the MPI function calls from within the industrial application was in the order of 200%. Furthermore the impact on the execution time of the MPI routines was in the order of 10%, thus if one does not put these numbers into perspective one might come to believe them to be vital. Bottom line is that in reality the performance increase, in wall clock time, was ~400 ms on a total runtime of 1 hour 54 minutes and 10 seconds.

Consequently the overall conclusion of this paper is that although there are applications that benefit from being able to send vast amounts of small messages fast as well as applications that need to move several Gigabytes of data thus utilizing either the low latency or high bandwidth of specialized network interconnects, non-linear FEM analysis is not one of them. Please refer to Section 5 for further details and a full discussion on latency vs. bandwidth and the impact on the current application domain.

Acknowledgements

We would like to thank Mikael Ericsson and Per Nordlund for the simulation models. We would also like to thank MSC.Software Sweden AB, Scali A/S and Dolphinics for lending us their equipment.

References


[8] Sommerville, I., Software engineering, 6th ed. (Reading, USA: Addison-Wesley, 2001)