The handle http://hdl.handle.net/1887/18622 holds various files of this Leiden University dissertation.

**Author:** Vu, Van Thieu  
**Title:** Opportunities for performance optimization of applications through code generation  
**Issue Date:** 2012-03-22
Chapter 3

Overlapping communications with calculations

The HIRLAM forecast model code has been designed to run on a massively parallel machine. Parallelization is by straightforward rectangular domain decomposition. A time step consists of a communication step, in which border data are fetched from the neighboring processors, followed by a calculation step over the domain assigned to a processor.

In this chapter we report on experiments to start the calculations while the communications are still in progress. The aim is to reduce overall execution time by hiding the communications behind the calculations. This obvious advantage of overlapping communications with calculations has a counterweight in increased calculation time and, perhaps more importantly, in increased code complexity. However, code generating may take away the burden of complex codes.

We anticipate that overlapping will need additional calculation time, and that the balance between gains and losses may depend on the computer hardware. Therefore, we describe a range of strategies for overlapping, ranging from one that may be the fastest on vector type machines, to one that aims at maximum overlap, regardless of the additional computational costs of overlapping.

3.1 Data communication in HIRLAM

3.1.1 The original method

Parallelization in the explicit dynamics part of the time step of HIRLAM is achieved by domain decomposition. Each processor performs the calculations for a rectangular domain. In Figure 3.1(a) this domain is indicated as the “processor domain”. The processor domains collectively form the HIRLAM domain.
Chapter 3. Overlapping communications with calculations

(a) Data areas to be filled

(b) Data areas to be sent

Figure 3.1: The halo zone in case the West-East communication precedes the North-South one. The four corners of the halo come from four non-adjacent processors. Communications of the data in the four corners are done firstly to the West and East adjacent processors, and then to the North and South adjacent processors. Therefore, those corners are included in the North and South halo zones.

To solve the equations, information is needed from adjoining regions of the domains of neighboring processors. Those regions form the “halo zone” around the processor domain. The calculations at the corners of the processor domain need information from non-adjacent processors. Therefore, the halo zone includes also these corners. For convenience, we divide the halo zone into the four subareas, denoting as Wo, Eo, No, and So, which contain data received from the Ei, Wi, Si, and Ni regions (Figure 3.1 (b)) of the West, East, North, and South adjacent processors, respectively (and of the 4 neighboring corner processors).

To collect the halo information there are 4 communication steps: first for the West and East neighbors, then for the North and South neighbors (or the other way around). The four corners of the halo come from four non-adjacent processors, indirectly: first the data are communicated to the West and East adjacent processors, and then to the North and South adjacent processors. Hence, the North and South halos include those corner areas. The original communication method of the HIRLAM consists of the following steps:

1. Exchange data with the West and East processors
2. Wait for completion of the West and East communications
3. Exchange data with the North and South processors
4. Wait for completion of the North and South communications
5. Compute the processor domain
3.1. Data communication in HIRLAM

3.1.2 The overlap concept

In the original communication approach, the halo zone is completely collected before the processor starts its calculation. With non-blocking MPI calls [76], a sending call returns control immediately, without waiting for the completion of the data transfer. Therefore, during exchanging data, we can perform a calculation which does not use the information from that exchange. In other words, we can overlap calculations with communications.

In HIRLAM, the halo information is needed only for calculations close to the boundaries of the processor domain. By “folding” the halo zone inward, we divide the processor domain into 5 subdomains: M, W, E, N and S, as in Figure 3.2. The calculations in the middle domain M are independent of the information in the halo zone, and thus can be done while the information for the halo is being collected.

Overlapping has the obvious advantage of saving elapsed time by hiding communications behind calculations. There are disadvantages as well, though. We mention:

1. More complicated coding. The calculations over the processor domain cannot be executed anymore with straightforward double loops in the two horizontal directions. Instead, the calculations have to be executed separately for a series of subdomains. So the calculation code becomes more complicated. Furthermore, the communication code has to be interspersed with the calculation code, which also implies more complicated coding.

2. More time spent in calculations. The separate execution of loops over subdomains will probably result in longer calculation times. Furthermore, the W and E subdomains are very narrow. The loops over longitudes when doing the calculations for those subdomains become very short. These are the inner of the two horizontal loops. On vector machines short inner loops cost relatively much in terms of calculation time. On cache-based machines short inner loops may be relatively expensive as well. Also, the N and S subdomains are very slim.
implying short outer loops, but probably, this is less harmful for performance than the short inner loops on the W and E subdomains.

A priori, it is not obvious that the time saved by overlapping communications with calculations will exceed the calculation time lost due to less efficient execution of the shorter loops and the larger number of subdomains. And even if it does, it still has to be judged whether the advantages outweigh the disadvantage of more complicated coding.

3.2 The overlap strategies

We investigate several strategies to overlap communications and calculations. Table 3.1 gives an overview. In this Table, communication is indicated by a double arrow ($\leftrightarrow$), and calculation subdomains are indicated by italics. Sequential execution is indicated by the table rows, top-down. Overlap of communications and computations is indicated by a communication entry and computations in the same row. Of course, a computation subdomain must only be processed after communications of data in that subdomain have completed. In order to communicate the domain corners properly, a communication in one direction must have completed before the other one is allowed to start. For example in Figure 3.1, the North-South communications are started after the West-East communications have been finished.

It so happens that these strategies can be distinguished by the number of subdomains for computations. Hence, we code the strategies by that number followed by the letter "s", which denotes subdomain. The 1s strategy has 1 subdomain; it is the current HIRLAM strategy. The 6s strategy has 6 subdomains: the 5 as shown in Figure 3.2 plus one because we split the middle subdomain. The strategies, to be described below, are ordered by the number of subdomains for calculations. Hence, the order does not imply any general preference by us.

The 1s strategy is the one currently used in HIRLAM, i.e., no overlap at all. First, all data are communicated, and then computations are done over the entire domain. This strategy is expected to have the lowest calculation time, but the highest communication time since there is no overlap. However, its main advantage is its code simplicity.

The 3s strategy is a design that may be profitable if short inner loops are very expensive to execute. This strategy achieves long inner loops in all computations, at the penalty of not overlapping the W$\leftrightarrow$E exchange. It results in overlap of the N$\leftrightarrow$S communications only.

In the 5s strategy, we accept short vectors when processing the West and East subdomains. We assume that those computations are slower than those for the North and South subdomains, because the latter have long vectors. Thus, more calculation time will be available for overlap if first W$\leftrightarrow$E is communicated, than if the N$\leftrightarrow$S communications would have been the first. Hence, the
3.2. The overlap strategies

Table 3.1: The four investigated overlap strategies. Strategy 1s: the original HIRLAM strategy (non overlap). Strategy 3s: the N⇔S communications overlap with the WME calculations. Strategy 5s: the W⇔E communications overlap with the M calculation and the N⇔S overlap with the WE calculations. Strategy 6s: both the W⇔E and N⇔S communications overlap with the M calculation.

<table>
<thead>
<tr>
<th>Strategies</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s (HIRLAM)</td>
<td>N⇔S</td>
</tr>
<tr>
<td>simple code</td>
<td>W⇔E</td>
</tr>
<tr>
<td></td>
<td>NWMES</td>
</tr>
<tr>
<td>3s long vectors</td>
<td>W⇔E</td>
</tr>
<tr>
<td></td>
<td>N⇔S</td>
</tr>
<tr>
<td></td>
<td>WME</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td>5s undivided M</td>
<td>W⇔E</td>
</tr>
<tr>
<td></td>
<td>N⇔S</td>
</tr>
<tr>
<td></td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td>6s divide M</td>
<td>W⇔E</td>
</tr>
<tr>
<td></td>
<td>N⇔S</td>
</tr>
<tr>
<td></td>
<td>M/2</td>
</tr>
<tr>
<td></td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
</tbody>
</table>

W⇔E communications are done first, overlapped with the computations for the M subdomain. Next, the N⇔S communications are overlapped with the computations for the W and E subdomains. At the end of the next paragraph we will explain the possible advantage of this strategy.

In the 6s strategy, we try to overlap both communication steps with the calculations over the middle domain. The combination of Fortran and MPI does not simply permit to synchronize communications while calculating. This synchronization is required to ensure that the communications in one direction have completed before those in the other direction start. We resolve this by
Chapter 3. Overlapping communications with calculations

3.3 Message transfer protocols of MPI

MPI defines three protocols to transfer messages with different message sizes: short, eager, and rendezvous [50, 76]. The short and eager protocols are used to transfer short messages, and the rendezvous protocol is used for long messages. The aim of using different methods is to balance the latency and the bandwidth. For a small message the bandwidth is less important than the latency. If a message is long, the bandwidth is more important.

Figure 3.3 describes the procedure of these protocols. An MPI message consists of two parts: data to be sent/received and an envelope that contains information needed to route data. In the eager protocol, both the envelope and data are sent immediately to the destination and the sending buffer is immediately returned to the application. Since the message is sent without knowing if there is a matching receive waiting, the message may need to be stored in a buffer at the destination, till a matching receive is posted by the application. The transferring approach using the eager protocol is called non-blocking MPI. The short protocol is a special variation of the eager protocol where the envelope and data all fit in one packet, the smallest unit of data that is passed through the network. The maximum amount of data that can be sent in one packet is 120 bytes [86].

In the rendezvous protocol, when a send is posted, the envelope is sent to the receiver and eventually buffered there. When the receive is posted and the appropriate envelope has already been received, the receiver sends an acknowledgment to the sender. Data is sent only when the sender received this confirmation. This protocol requires the receiver to eventually buffer only the
envelopes. Because the receiver is synchronized, the rendezvous protocol can avoid an additional copy of the data. This transferring method is called blocking MPI.

As opposed to blocking MPI, where the sender is blocked until the receiver confirms that the message has been completely received, a send by non-blocking MPI returns control immediately to the application. Therefore, other tasks such as calculations can be done while data is transferring. Because of buffer requirement, the use of non-blocking MPI is limited by the fact that the message has to be smaller than the buffer size, which has a maximum of 64 KB [50]. MPI switches to blocking mode automatically if the message size is larger than 64 KB.

### 3.4 Experiment configuration

#### 3.4.1 Hardware

The experiments were conducted on DAS-3 [67]. DAS-3 is a wide-area distributed system which consists of five clusters located at four universities in The Netherlands: Vrije Universiteit (VU), Leiden University (LU), University of Amsterdam (UvA-VLe and UvA-MN), and Delft University of Technology (TUD). The computational resources of DAS-3 are presented in Table 3.2.

Single cluster experiments were performed on the VU cluster, which has 85 dual compute nodes, so 170 processors in total, each dual core. The clock frequency of each processor is 2.4 GHz. Our experiments on two clusters involved the VU cluster and the LU cluster. The latter has 32 dual nodes single core processors, with a clock frequency of 2.6 GHz. In all experiments we always use both clusters in single node, single core mode. The physical distance between the two clusters is approximately 40 km. The two-cluster configuration is considered as a prototype for a grid [22].

<table>
<thead>
<tr>
<th>Cluster</th>
<th>#Nodes</th>
<th>Type</th>
<th>Speed</th>
<th>Memory</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU</td>
<td>85 dual</td>
<td>Dual core</td>
<td>2.4 GHz</td>
<td>4 GB</td>
<td>10 TB</td>
</tr>
<tr>
<td>LU</td>
<td>32 dual</td>
<td>Single core</td>
<td>2.6 GHz</td>
<td>4 GB</td>
<td>10 TB</td>
</tr>
<tr>
<td>TUD</td>
<td>68 dual</td>
<td>Single core</td>
<td>2.4 GHz</td>
<td>4 GB</td>
<td>5 TB</td>
</tr>
<tr>
<td>UvA-VLe</td>
<td>41 dual</td>
<td>Dual core</td>
<td>2.2 GHz</td>
<td>4 GB</td>
<td>5 TB</td>
</tr>
<tr>
<td>UvA-MN</td>
<td>46 dual</td>
<td>Single core</td>
<td>2.4 GHz</td>
<td>4 GB</td>
<td>3 TB</td>
</tr>
</tbody>
</table>
Chapter 3. Overlapping communications with calculations

Figure 3.4: The connections in DAS-3. DAS-3 consists of five clusters: VU (Vrije Universiteit); LU (Leiden University); UvA-VLe and UvA-MN (University of Amsterdam); and TUD (Delft University of Technology). Within a cluster, connection uses Myri-10G, except the TUD cluster that uses Ether-10G. Connection between clusters uses Myri-10G switch via an optical blackbone SURFnet6.

3.4.2 Network connections

The network connections in DAS-3 are shown in Figure 3.4. Within a cluster, the nodes are connected with a Myri-10G switch and with 1 Gbps Ethernet. Communication between clusters can be over common internet, to which the clusters are connected with 10Gbps Ethernet. The Myrinet hardware can also be used for communications between clusters, but then it does not support the Myrinet express protocol MX (Myrinet-MX). Instead, the protocol available for inter-cluster communications over the Myrinet cards is TCP (Myrinet-TCP). Between the clusters there are 8 fiber at 10 Gbps each, shared with other users of the clusters. The influence of the network connections on the communication times is investigated in Subsection 3.5.2.

Different from Ethernet and Myrinet-TCP, the Myrinet-MX communication primitives are fully asynchronous [51]. It means that the Myrinet-MX protocol supports the overlapping of communications with computations without a dependence on the message size. In Subsection 3.5.3, we will investigate how this feature of Myrinet-MX affects to the overlapping communication.

3.4.3 Software

As test case we used the DYN routine from the HIRLAM system. It codes a grid point, explicit Eulerian primitive equations solver. We chose to perform our
3.4. Experiment configuration

We experiment with the code generated by CTADEL [17] rather than the original hand coded routine. We made this choice for the following reasons:

1. Flexible area selection. The hand code assumes the South West corner of the domain to be the (1,1) point. This is awkward in case the DYN routine has to be applied on different subdomains. The code generator is straightforwardly adapted to generate code for which the South West corner can be adjusted.

2. The generated code is correct. This is not a very strong argument, as the hand code is sufficiently correct for our purposes.

3. Future: inlining, loop unrolling. The DYN routine is invoked over a number of subdomains. If subroutine calling overhead turns out to be significant, we may want to decide to inline the series of DYN invocations to eliminate that overhead. The code generator can be modified to do so. The West and East domains have very short loops in the longitude direction. In the hand code, these loops are the inner loops. A lot of loop overhead can be avoided by unrolling the inner loops. Again, the code generator can be modified to do so.

4. The generated code is faster. R. van Engelen et al. [17] showed that the code generated by CTADEL is faster than the hand-written code, with a speedup of 7.3%, 13%, 24%, and 44% on HP 9000/720, Convex C4, MasPar, and CRAY T3D machine, respectively. Our research aims at optimizing performance, therefore choosing the fastest code as basis for our assessment is meaningful.

We coded the invocation of DYN for each subdomain from a main routine by hand. So we invoke the generated DYN code from a hand-coded main routine.

Our main routine also organizes the communications by invocation of the MPI non-blocking communication routines MPI_ISEND and MPI_IRECV. Depending on the strategy, the order of invocation of these routines and DYN ensures that the communication is parallel or sequential to the calculation. Synchronization is by MPI_WAITALL.

As compiler we used mpif77 version 1.2.7. The communication routines for our single cluster runs were from MPICH version 1.2.7, and for the multi-cluster runs from OpenMPI version 1.2.1, because MPICH turned out not to work well between clusters. The performance difference between the two MPI implementations is small.

Timings were measured with the MPI wall clock routine MPI_WTIME.

3.4.4 Reproducing results

In parallelization studies it is always crucial to avoid the usage of data from other processors before they have been communicated. We verified the correctness of our different strategies, and the correct implementation in our experiments, by checking that the tendencies of surface pressure, humidity, temperature, and wind components, which are calculated with DYN, were exactly the same as
those obtained with a single processor run over the total domain. Hence, the meteorological results of our experiments are reproducible.

However, the main objectives of this study are not the tendencies, but the elapsed times it takes to obtain them. To get an impression of the reproducibility of the timings, we ran each experiment 5 times. It turns out that the pure calculation times are very well reproducible: they show variations of the order of 1%. This low variation is a consequence of the processor allocation strategy on DAS-3: there is no time sharing. The communication times, however, were found to vary by much more. On a single cluster, variations by 50% were observed, and on two clusters even by a factor of 2. These variations arise because the networks were not dedicated in our experiments. Some variation within a cluster arises because the topology of the allocated processors is not always optimally matching the domain decomposition. The timings that we report in this study are therefore chosen to be the medians out of 5 runs instead of the average of 5 runs.

3.5 Results

This section presents the experimental results of the overlap strategies on a single and multiple clusters of DAS-3. We also investigated the scaling of the calculation times with the number of processors and the effect of the system buffer size on overlapping.

3.5.1 Scaling of the calculation times

Our first investigation is to look into the scalability of the calculation time, without looking at communication times yet. We ran the domain of 120 by 120 grid points in the horizontal, and 16 levels in the vertical, on $2 \times 2$, $4 \times 4$, $6 \times 6$ and $8 \times 8$ processors, on a single cluster. The calculation times are shown in Figure 3.5. The line “lin” is the linear extrapolation of the 1s strategy data point on $2 \times 2$ processors.

The 1s strategy scales almost linearly upward from the $2 \times 2$ processor run. Scalability decreases with the number of subdomains, which is consistent with the increased number of subroutine calls. This is because when splitting the processor domain into the subdomains, the loops when doing calculations on the subdomains are shorter than those on the processor domain, which results in the longer calculation times. The Ethernet environment is less scalable than Myrinet. This indicates that the calculations with the Ethernet environment are more expensive than with the Myrinet environment. In other word, Ethernet requires more CPU activity than Myrinet.

On $2 \times 2$ processors, the other strategies are less than 5% slower than 1s, and Ethernet is about 1% slower than Myrinet. With more processors, these
3.5. Results

![Graphs showing calculation times for Ethernet and Myrinet-MX networks.](image)

**Figure 3.5:** Inverse calculation times as a function of the number of processors. The dashed line without measurement marks follows the linear speed up from the $2 \times 2$ processors, 1s strategy measurements.

Differences grow to almost 20% on $8 \times 8$ processors. This 20% reduction in scalability by domain decomposition does not deter us much, in particular because the number of processors is usually not increased to complete a certain problem faster, but to increase the problem size, i.e., the number of grid points.

3.5.2 Influence of the network connections on the communication times

To assess the influence of the network connections on the communication times, we performed experiments with the domain of $120 \times 120 \times 16$ and $240 \times 240 \times 16$ grid points on $4 \times 4$ processors. Figure 3.6 shows the communication times with Ethernet, Myrinet-TCP, and Myrinet-MX on a single and two clusters.

We observe that the Myrinet-MX protocol is faster than the Myrinet-TCP and the Myrinet-TCP protocol is faster than the Ethernet. On a single cluster, the Myrinet-MX is 6 times faster than the Ethernet protocol. The Ethernet protocol is 1.5 to 2 times slower than the Myrinet-TCP protocol. On two clusters the Ethernet protocol is 1.2 to 1.6 times slower than the Myrinet-TCP protocol.
3.5.3 Effect of the system buffer size on overlapping

As mentioned in Section 3.3, if the size of a message exceeds a certain system buffer size, the non-blocking MPI routines switch to blocking mode, thus inhibiting overlap of communications with calculations. In this Subsection we examine the effect of the system buffer size to the performance of overlapping in our application.

We investigated two buffer sizes, namely 32 KB and 64 KB. With each buffer size we performed two experiments in which the message size is smaller than the buffer size, and two others in which the message size is larger than the buffer size. We used the configuration of 4 × 4 processors on the Vrije Universiteit (VU) cluster, and three connection protocols on DAS-3: Myrinet-MX, Myrinet-TCP, and Ethernet.

To assess the performance of overlapping, we compared the communication times of the overlapping communication method, strategy 6s, with the original non-overlapping approach of the HIRLAM, strategy 1s (see Table 3.1). The results are shown in Figure 3.7. The communication time is the total cost of sending (MPI_ISEND) and waiting (MPI_WAIT) calls. With the overlapping strategy 6s, a part of the communication times has been hidden behind the calculation times. Therefore the communication times that we showed in Figure 3.7 are the non-overlapping part of the communication times. If there is overlap, the non-overlapping communication times of strategy 6s are much smaller than the communication times of strategy 1s. If overlap does not happen, the non-overlapping communication times of strategy 6s and the communication times of strategy 1s are similar.

First of all, we observe that with the Myrinet-MX protocol, the communica-
3.5. Results

![Diagrams showing communication time (ms) vs message size (KB) for different network and buffer configurations.]

Figure 3.7: Effect of the system buffer size on the performance of overlapping. The overlapping is recognized by the reduction of the communication time of the overlap strategy, 6s.
tion times of strategy 6s are always smaller than those of strategy 1s. It means that we always have overlap, even in case that the message size exceeds the system buffer. This is because the MX communication primitives fully support asynchronous communication [51], which allows overlapping calculation with communication without the dependence on the system buffer size.

With Myrinet-TCP and Ethernet environments, the overlap occurs only if the message sizes are smaller than the buffer, specifically, 29 KB and 31 KB in case of 32 KB buffer; and 57 KB and 61 KB for a 64 KB buffer. With larger messages, the overlap does not happen because the system buffer can not fully contain those large messages, hence, the sending call is blocked until the communication is completed. As a result, the communication switches from non-blocking to blocking mode. In the next subsections, we simply circumvent this limitation by ensuring that our messages are smaller than the possible largest system buffer size, which, in our case, is 64 KB. We reach this message size by using a fairly small domain: 120 $\times$ 120 points in the horizontal, and 16 vertical levels; and by using a sufficiently large number of processors: for this domain we need at least 4 $\times$ 4 processors. In the context of the limiting message size, we profit from the fact that DYN does not need a halo that is wider than 2 points. With the 120 $\times$ 120 $\times$ 16 domain, the message size transferred by each processor depends on the number of processors as can be seen in Table 3.3.

Table 3.3: The size of message transferred by each processor. The shading denotes the configuration of which the message is smaller than the 64 KB system buffer.

<table>
<thead>
<tr>
<th>Processor configuration</th>
<th>Message size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 $\times$ 2</td>
<td>122</td>
</tr>
<tr>
<td>4 $\times$ 4</td>
<td>61</td>
</tr>
<tr>
<td>6 $\times$ 6</td>
<td>30.5</td>
</tr>
<tr>
<td>8 $\times$ 8</td>
<td>15.25</td>
</tr>
</tbody>
</table>

3.5.4 Overlap on a single cluster

The experiments on a single cluster were performed on the VU cluster of DAS-3. The system buffer was set to be 64 KB [50], the maximum value of the system buffer. We used the domain of 120 $\times$ 120 $\times$ 16 gridpoints. The processor configurations are 2 $\times$ 2, 4 $\times$ 4, 6 $\times$ 6, and 8 $\times$ 8 processors. Figures 3.8 and 3.9 show the overlap between communications and calculations.

The 1s strategy does not have any overlap. The 3s, 5s and 6s strategy have at least one of the communication phases overlapped with calculations. This
3.5. Results

![Graphs demonstrating time differences for different configurations.](image)

Figure 3.8: Calculation time, and non-overlapping parts of the communication times on $2 \times 2$ and $4 \times 4$ processors. Single cluster results.
Figure 3.9: Calculation time, and non-overlapping parts of the communication times on $6 \times 6$ and $8 \times 8$ processors. Single cluster results.
is seen in Figures 3.8 and 3.9 by the reduction of communication time spent in that phase. To be precise, the communication time is still there, but it is hidden behind the calculation time.

With the $2 \times 2$ processor configuration, because the message size exceeds the system buffer, the overlap does not happen with the Myrinet-TCP and Ethernet, and occurs only with the Myrinet-MX protocol. This result confirms our explanation in Subsection 3.5.3.

We observe in Figures 3.8 and 3.9 that communication is hidden behind calculations, to a large extent, as far as the strategies permit: in 1s there is no overlap; in 3s the N⇔S communications are overlapped; in 5s the W⇔E exchange is overlapped, but the calculation time for the W and E domains is far too small to hide the N⇔S communications; in 6s communication times almost disappear.

The increase in calculation time with the number of subdomains is clearly demonstrated by Figures 3.8 and 3.9. The calculation time is almost linear in number of subdomains; e.g., with Myrinet-MX, on $4 \times 4$ processors, the correlation between calculation time and number of subdomains is higher than 0.999. This, by the way, confirms the reproducibility of the calculation times. Communication times are less reproducible. For example the W⇔E communication times on $8 \times 8$ processors with Ethernet of the strategies 1s and 3s are different, although there is no overlap in either of the strategies 1s and 3s.

With Myrinet-TCP and Ethernet, the communication is so slow that each strategy to overlap communications and calculations is profitable. With Myrinet-MX the communication overhead is very low. Hence, the decreased communication time hardly compensates the increased calculation time arising from splitting the processor domain into subdomains (discussed in Subsection 3.5.1). The non-overlapped communication time, for example in the case of strategy 6s, $8 \times 8$ processor, with Myrinet-TCP and Ethernet environment, indicates that Myrinet-TCP and Ethernet also requires some CPU activity.

3.5.5 Overlap on two clusters

On two clusters, the picture changes on some points. We implement the system on two clusters by assigning the Northern half of the total HIRLAM domain to one cluster, and the Southern half to the other. Figure 3.10 shows the timings on the Leiden University (LU) cluster, which has faster processors. Figures 3.8 and 3.9 contain timings on a single cluster, hence at the Vrije Universiteit (VU) cluster, with slower processors.

The first point to note in Figure 3.10 is that the calculation times have decreased by approximately 8%. This is a consequence of our choice to present the timings on the LU cluster with the faster processors. On the VU cluster, the calculation times are similar to those in Figures 3.8 and 3.9. However, that cluster delays the faster cluster during communications. Indeed, on the faster
Chapter 3. Overlapping communications with calculations

![Graphs and bars showing calculation times and non-overlapping parts of communication times for different network configurations and processor counts.]

(a) Ethernet, 2 × 2 processors
(b) Myrinet-TCP, 2 × 2 processors
(c) Ethernet, 4 × 4 processors
(d) Myrinet-TCP, 4 × 4 processors
(e) Ethernet, 6 × 6 processors
(f) Myrinet-TCP, 6 × 6 processors
(g) Ethernet, 8 × 8 processors
(h) Myrinet-TCP, 8 × 8 processors

Figure 3.10: Calculation time, and non-overlapping parts of the communication times. Two cluster results.
cluster (LU) the N ⇔ S communication is less overlapped than on the slower cluster (VU). The difference in overlap is equal in size, opposite in sign, to the difference in calculation times.

The second point we note is that the Myrinet-TCP protocol is not consistently faster than Ethernet, even though Ethernet is just over common internet.

The third point we mention is that the N ⇔ S communication takes more time than on a single cluster, even after correction for the delays caused by the slower processors on the other cluster. However, given the large physical distance between the two clusters (40 km) the additional delays are relatively small. Yet, by using two clusters, the total communication time has grown more than the total calculation time, in particular on 8 × 8 processors, e.g. with Ethernet, from 39.8 ms to 43.6 ms. Hence, it cannot be hidden fully anymore, not even in strategy 6s.

3.6 Discussion

If a fast interconnect is available, nearest neighbor communication is fast with straightforward domain decomposition. In this case it will be difficult to reduce the total execution time by overlapping communications with calculations. Even if the little time needed for communication can be eliminated, it is doubtful whether the gain balances the additional calculation time. Knowing this, we started this investigation with designing a range of strategies to play with the balance of gaining by overlap and losing by additional calculations. Our experiments with the Myrinet-MX protocol confirmed this: little time is spent in communications if not overlapped at all (strategy 1s), and minute savings in total execution times are possible. On 8 × 8 processors, we found that strategy 3s is the fastest, but in total, when using MX, the gains do really not warrant the code complications needed for overlap.

On the other hand, if the interconnect is not very fast, considerable savings are possible by overlapping. On our hardware, the additional computational costs of processing several subdomains, some with very short vectors, are limited. On two clusters we even found that calculations are too fast to enable full overlap. In that case all our experiments show that strategy 6s is preferable.

After inspection of the 6s strategy, we realized that a refinement should be possible, to hide more calculation time behind the N ⇔ S communication. Table 3.4 and Figure 3.11 illustrate this refinement: the middle subdomain M is divided into two parts, not necessarily equal in size, and the calculations over W and E are also done parallel to the N ⇔ S communication. To maximize vector length, we join the second part of M and the matching parts of W and E to form the (1-α)(WME) area (see Figure 3.11). Parameter α can be tuned to achieve maximum overlap. We introduce the notation 6s(α) for this strategy, and note that 6s(0) and 6s(1) become our old strategies 3s and 5s, respectively.
Figure 3.11: The processor subdomains for strategy $6s(\alpha)$. The $N \leftrightarrow S$ communication overlaps the calculations in the grey area.

After elimination of the redundant code for the empty subdomains. On our hardware the calculations over $W$ and $E$ do not take much time; hence $6s(1/2)$ is not much better than $6s$; we found an optimum at $\alpha = 0.3$, saving 2% of total execution time over $6s$.

<table>
<thead>
<tr>
<th>Table 3.4: Strategy $6s(\alpha)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$6s(\alpha)$</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Once we have overlap, we observe that there is not much advantage of having a state-of-the-art interconnect. With a relatively cheap 1 Gbps Ethernet the total execution times are comparable to those obtained with Myrinet-MX. For example the total execution times of strategy $6s$ in Figures 3.8b and 3.8d, Figures 3.9a and 3.9c, and Figures 3.10c and 3.10d are comparable.

Also, we observe not more than a slight performance decay when we went to two clusters, even if the two clusters are connected over the internet. Part of the performance loss on our two clusters is due to the difference in processor speeds, of approximately 8%. When we used $4 \times 4$ processors, we assigned a domain of $30 \times 30$ grid points to each processor. We can reduce the impact of the
different processor speed by redistributing grid points. Specifically, we let the faster processors calculate 31 lines, and the slower processors calculate 29 lines of latitudes. The speed difference is then almost fully compensated, and HIRLAM needs 3% less execution time. On $8 \times 8$ processors, it is hardly worth assigning 14 lines of latitudes to the slower, and 16 to the faster processors, because then the faster processors get a too heavy load: this distribution overcompensates the speed difference. We note that the speed difference can be made useful by increasing the HIRLAM domain size from $120 \times 120$ to $120 \times 124$, where the 4 additional lines of latitudes are assigned one to each of the faster processors. In this way we get an increase of domain size for free.

The calculations over the W and E subdomains have inner loops of length 2. This calls for elimination of those loops, by unrolling them. A further optimization will then be reached by merging the codes for the W and E subdomains. Actions of this kind are best performed by the CTADEL code generator [17]. On our hardware we will not profit much from these modifications, because the additional computation time needed to achieve overlapping is small. But on hardware where short inner loops are expensive, e.g., on vector machines, these optimizations may be essential to achieve advantages of overlapping. Similar loop eliminations and code merging for the outer of the horizontal loops over the N and S subdomains will be profitable, though less than those for the inner loops on W and E.

Our study seems to suggest that possibly HIRLAM can be modified, to achieve reasonable performance also on clusters with slower (cheaper) interconnects, or on a wide area computer. However, first we will have to resolve the issues raised by the global communications, by the semi-Lagrangian, and by the limited MPI buffer size.

On the issue of global communications we make the following remarks. The nature of the calculations that need global communications inhibits implementing overlap straightforwardly. Hence, algorithms can be designed that permit starting calculations before all data needed to complete the calculations have been received, but we doubt that the time taken by those calculations is significant as compared to the communication times. Even if some overlap can be achieved, results will lose the bitwise reproducibility, because the order of calculations will depend on the domain decomposition. Fortunately, global communications happen infrequently as compared to the communications needed by the semi-Lagrangian HIRLAM implementation, so their optimization is not very urgent. It seems wise to give priority to resolving the issues raised by the semi-Lagrangian scheme.

The first of those is its implementation in HIRLAM, with a large number of communication steps during each time step. Hand coding for overlap is very cumbersome. Although the essentials of the semi-Lagrangian scheme can be generated by CTADEL [44] the complete scheme has never been generated. A
full, generated, implementation will result in a very small number of communication steps, probably one per semi-Lagrangian iteration step, and from then on, it would be straightforward to use the work we have done on the Eulerian scheme to experiment with overlapping in the semi-Lagrangian scheme.

A second issue for the semi-Lagrangian scheme is the width of the halo, usually in the order of 10 grid points. Because this is much wider than in the Eulerian scheme, the middle subdomain becomes smaller, and communications more costly. There will be less calculation time to hide the larger communications behind. This is partly compensated by the higher costs for semi-Lagrangian than for Eulerian time stepping. More serious is the fact that the wider halo will result in too large messages to fit into the MPI buffers. Our experiments were configured to fit the MPI (‘eager-limit’) buffer size requirements to allow overlap. For example, our experiments used 16 levels whereas the real application uses 60 levels. Before more levels can be used, or before the halo width can be extended (as needed by the semi-Lagrangian scheme), or before more parameters like trace gases can be added, we will have to find ways around this limit. We can configure the buffer size during installation of MPI; we can use buffered communication MPI; and we can reduce the message size by message splitting. The last of these options will only work if the middle domain is split as well, resulting in more calculation overhead. An interesting interaction will occur if we also implement the “halo on demand” technique [44]. In this technique, advection wind speed and direction determine which part of the halo has to be communicated. It reduces communication. If combined with overlapping, the division into subdomains becomes flow dependent. We can also use the MX protocol to eliminate the dependence of asynchronous communication on the message size. However, this protocol is only available within a single cluster of DAS-3.

In the course of time, the number of grid points in a processor domain has the tendency to decrease [13]. Within two decades from now, it will have shrunk to less than 400 grid points. In a semi-Lagrangian scheme the halo zones have width of 10 grid points. Hence, there will be nothing to overlap if these 400 points are chosen in a $20 \times 20$ configuration. But in, e.g., a $10 \times 40$ configuration, N⇔S communication can still be overlapped with the calculations in the W and E areas as in the strategies 3s and 5s, even though the middle subdomain M has disappeared.

3.7 Related work

The overlap between communication and computation using non-blocking MPI implementation has been studied for a long time. Several papers assessed the ability to overlap communication with computation of non-blocking MPI. Sohn et al. [71] identified the capability of overlapping computation with communica-
3.7. Related work

The experiments have been performed on two distributed-memory multiprocessors: the EM-X and IBM SP-2. The results indicated that both multiprocessors would yield up to 30% to 40% overlap of communication time when the message size is approximately 1K integers. EM-X is found to be message size insensitive yielding high overlap for various message sizes, while SP-2 was effective for a window of message size from 512 to 2K integers. In [62], the authors assessed the possibility to overlap computation with communication based in InfiniBand, Myrinet-10G, and 10-Gigabit Ethernet, and concluded that transferring small messages makes an acceptable level of independent progress \(^1\). On the other hand, in most cases, transferring large messages does not make progress independently, decreasing the chances of overlap in applications. Brightwell et al. [9, 10] analyzed the impact of overlap, independent progress, and offload to the performance of an application. The experiments showed that the overall execution time of an application applying these techniques may reduce up to 20%.

In the context of algorithms, there are some papers that evaluated the benefit of overlapping computation with communication in parallel computing. In [30], Torsten Hoeffer et al. studied standard non-blocking collective operations. They implemented parallel multi-dimensional Fast Fourier Transformations (FFTs) and performed experiments on the Cray XT4 and Coyote InfiniBand-based cluster system. The results showed the performance advantages of non-blocking collectives. Up to 99% of the communication could be overlapped with computation. The application of a pipelined computation/communication scheme to a FFTs problem showed performance gains for a 128 process job of up to 14.2% on a Cray XT4 and 13.7% on an InfiniBand-based cluster system. Majumder et al. [41] proposed an algorithm that uses event-driven communication threads to make message transfer progress in the library separately from the application thread, thus decoupling communication progress from library invocations by the application. The asynchronous event-thread allows messages to be sent and received concurrently with application execution. Micro benchmark results showed that the time spent for waiting for non-blocking receives to complete can be significantly reduced or even eliminated entirely. In [39], Gary Liu et al. evaluated a compiler transformation for parallel programs on Network-of-Workstation (NOW) shared memory multiprocessors. The transformation overlaps the communication time resulting from non-local memory accesses with the computation time in parallel loops to effectively hide the latency of the remote accesses. Experimental evaluation of the transformation on a NOW multiprocessor indicated that it is generally effective in improving parallel execution time.

\(^1\)MPI implementations use a progress engine to manage the progress of communication after a non-blocking call return. Independent progress means that a pending non-blocking communication request makes progress without requiring any future library calls.
Our research is different from the related studies mentioned above, since it is application driven, in contradiction with the studies described above, which are specific studies based on ideal situations, or stand-alone algorithms. In our research selected choices can interfere with other parts of the application code. Therefore these choices should be taken into account.

The behavior and performance of non-blocking MPI strongly depend on the system buffer size. This problem has been addressed in [50, 76]. These papers describe three send/receive protocols of non-blocking MPI and the dependence of these protocols on the buffer size. In practice, the impact of the buffer size on the performance of overlapping computation with communication is studied in some papers. In [64] Saif et al. investigated the behavior of non-blocking communication with different message sizes. The results showed that when the message size exceeds the system buffer, the communication switches from asynchronous to synchronous mode. In [3], the authors examined the send/receive mechanism of non-blocking MPI. Experiments with two different parallel algorithms for sparse Gaussian elimination, a multifrontal solver (MUMPS) and a supernodal one (SuperLU), showed that the performance of the non-blocking MPI communication primitives are very sensitive to the system buffer. If the messages are smaller than the system buffer, the communication time using non-blocking MPI is up to 40% smaller than those using blocking MPI. If the messages are bigger than the system buffer, the communication times using non-blocking and blocking MPI are similar.

In this chapter we also investigated the effect of the buffer size on the performance of overlapping computation with communication. We experimented with the overlapping and non-overlapping communication methods of the HIRLAM weather forecast model on DAS-3. Our aim is to study the impact of the buffer and environment on the overlap behavior of the new communication strategies and to find the size of the problem under which the overlap occurs. We found that the limited system buffer has impact on the problem size. Indeed, in experiments we limited our problem to a small domain of $120 \times 120 \times 16$ grid points. We discussed several solutions for this issue, such as splitting the message or configuring the buffer size during installation of MPI.

### 3.8 Conclusions

We have demonstrated that depending on the available hardware little to considerable savings on total execution times can be achieved by overlapping communications with calculations, for the explicit Eulerian HIRLAM scheme. On our hardware, with our compiler, the savings are so substantial that there is no need for an expensive interconnecting network, and even running on a wide area Grid hardly incurs any delay. However, if a fast (hence expensive) interconnecting network is available, the attainable savings are negligible, and they
surely do not warrant the associated code complications.

We base our conclusions on experiments with the CTADEL generated code for the HIRLAM routine DYN. We chose this generated code mainly because initially we anticipated that we would need several optimizations, like loop unrolling and subroutine inlining, to reduce additional computational overhead to a level that the time savings by overlapping would exceed that overhead. However, we found that on our hardware, the overhead is small. Similar results would therefore have been obtained with the original HIRLAM hand code. In the configuration of $8 \times 8$ processors over 2 clusters, we found that pure calculation time of the generated code is not sufficient to fully cover the communication times. In that configuration the hand code, being computationally more expensive, would even have shown bigger profits of overlapping than the generated code.

We investigated several strategies for the overlap. We almost universally found that strategy 6s with 6 subdomains was the most efficient, although it incurs the largest computational overhead. The reason is that on our hardware the increased calculation time arising from splitting the processor domain was small. We anticipate that on hardware that favors long inner loops, e.g., on vector machines, strategy 3s, which was designed to avoid short inner loops, would be preferable.

It will take considerable efforts to implement our findings in a production code, in particular because that would require a fair amount of rewriting of the semi-Lagrangian scheme. We would also have to resolve the problem posed by buffer size limitations. Furthermore, the savings by the overlapping scheme would be reduced by the anticipated challenges and limitations to apply it to the global communications required for the implicit parts of the time stepping schemes. But perhaps the biggest challenge is how to cope with the required code complexities.