

Latency hiding in GETM

Bjarne Büchmann and Jesper Baasch-Larsen, DaMSA

August 1, 2011

Report ID	Report_DaMSA_2011-02
Title	Latency hiding in GETM
Document name	Report_DaMSA_2011-02.pdf
Keywords	High Performance Computing, GETM, Latency Hiding
Report type	Technical report for internal and external use
Publishing date	August 1, 2011
Version no.	1.0
Department	Hydrography and Maritime Data (HMD)
Author	Bjarne Büchmann and Jesper Baasch-Larsen, DaMSA
Approval [contents]	Johan Mattsson
Class	Public
Approval [form]	Helena Brahe

Content

1	Summary	1
2	Introduction	2
3	Model problem	3
3.1	Physical setup	3
3.2	Numerical setup	4
4	Implementation	6
5	Initial timing of reference experiments	9
5.1	2D reference test	9
5.2	3D reference test	12
6	Latency hiding – 2D	14
6.1	Latency hiding principle	14
6.2	Sea level implementation	15
6.3	2D momentum implementation	15
6.4	2D results	15
7	Latency hiding – 3D	19
7.1	3D HALO exchange break-down	19
7.2	GETM 3D advection code structure	20
7.3	3D advection and the Arakawa C-grid	21
7.4	3D results	26
7.5	3D alternative approach	28
8	Conclusions	31
9	References	33
A.	Timing data	34

1 Summary

Danish Maritime Safety Administration publishes forecasts of hydrodynamic variables four times daily. For water elevation, current, salinity and temperature, the forecasts are based on the open-source circulation model GETM, <http://getm.eu/>. The forecasts are executed on DaMSA's production Linux cluster.

GETM is already parallelized using Message Passing Interface (MPI) and Open Multi-Processing (OpenMP). This study seeks to improve the MPI parallelization using latency hiding. Latency hiding is a method in which the inter-process communications are overlapped with computations with the goal of improving the performance. The model performance has been investigated in both 2D and 3D experiments.

It is found that the communications in 2D are mainly limited by initial net latency and not by throughput. The communication is shown to take up most of the time spent in subroutines where latency hiding can be applied. The result is that there are little computations available for hiding the latency. It is also found that the implementation of latency hiding itself incurs a computational penalty which is larger than the gains in the communications.

In 3D, the communications are mainly limited by net throughput and that approximately 25% of the total run time is spent on communications. For some of the subroutines the amount of computations available is found to be sufficient for latency hiding. Actual experiments with latency hiding for selected 3D related subroutines do show a significant improvement in the amount of time spent on communications. The improvement is however to a large degree mitigated by the computational penalty mentioned above. And it is argued that the situation will be even worse for faster networks. It is therefore concluded that latency hiding is not worth the extra code clutter it introduces into GETM. The study also underlines that an effective way to improve GETM performance is to use a fast network.

2 Introduction

The General Estuarine Transport Model¹ (GETM) is a finite difference 3-dimensional hydrodynamic model. The model is open source (GPL) code written in Fortran 90, and it consists of approximately 25,500 source lines of code (SLOC). It furthermore uses the General Ocean Turbulence Model² (GOTM), which consists of approximately 16,500 SLOC Fortran and 12,500 SLOC Python. The GETM model is parallelized using both MPI³ and OpenMP⁴.

In the present report, a possible improvement of the MPI parallelization of GETM is examined, using the method of so-called latency hiding. A priori, it has been a strict requirement that the changes introduced do not complicate the code unnecessarily and that the produced code has a quality which in time allows the changes to be committed to the GETM project.

The purpose of latency hiding is to interlace communications and computations⁵ in such a way, that there is work (computation) to be done, while the necessary – and often unavoidable – communications between processes take place. In a sense, the computations, which have to be done anyway, can “hide” the time it takes for communication to take place. The basic principle of latency hiding will be addressed in Section 6.1.

All computations are made on the Danish Maritime Safety Administration Linux development cluster. The cluster is a Beowulf type cluster consisting of a frontend with 15 nodes connected by switched Gigabit Ethernet LAN. The nodes feature Intel Core 2 Quad 2.4 GHz CPUs with 4 MB shared L2 cache and 2GB RAM with 4 cores per node.

Programs have been compiled with the Intel Fortran compiler⁶ version 11 and linked to MPICH2⁷ version 1.0.6p3 library for MPI.

¹ The General Estuarine Transport Model (GETM), <http://getm.eu/>

² General Ocean Turbulence Model (GOTM), <http://gotm.net/>

³ Message Passing Interface, see e.g. Gropp *et al.* (1999) or http://en.wikipedia.org/wiki/Message_Passing_Interface

⁴ Open Multi-Processing, see e.g. Chapman *et al.* (2008) or <http://en.wikipedia.org/wiki/OpenMP>

⁵ See e.g. Eijkhout *et al.* (2010) or MacDonald *et al.*

⁶ Intel Fortran Compiler, ifort, Version 11.1.059 (Build 20091012) <http://software.intel.com/en-us/articles/intel-compilers/>

⁷ Message Passing Interface Chameleon, MPICH2, <http://www.mcs.anl.gov/research/projects/mpich2/>

3 Model problem

In order to evaluate the effect of the implementation, a simplified model problem is constructed. The model problem will be solved in both 2D and 3D for both the unmodified (“plain vanilla”) GETM code, and with an updated code including latency hiding.

3.1 Physical setup

A test case is created to resemble the tidal propagation in the North Sea, at least in terms of order of magnitude for the physical parameters. The model domain is a simplified version of the North Sea with a horizontal extent of 480 by 480 km and a uniform depth of 60 m. The horizontal extent is chosen so that if a “nice” cell size (such as e.g. 1 km) is chosen, then the domain can be subdivided easily into $M \times N$ subdomains, where there are many possible choices for M and N . Thus, the setup makes it possible to examine a large number of different parallel settings. The horizontal resolution in most of the experiments is 2 by 2 km resulting in a horizontal grid size of 240 by 240. The limited problem size allows us to test very small subdomains, even though we will limit the present experiment to one subdomain per core on the 60 cores totally available for the computations. The setup has 60 vertical layers, discretized with sigma-type coordinates adapting to the (flat) bottom and the time varying free surface (air-water interface). The model has an open boundary to the north, where a spatially uniform harmonic sea level forcing with 0.5 m amplitude and 12 h period is applied. Due to the Coriolis force a Kelvin wave develops and progresses from the open boundary along the land to the right. The setup is illustrated in Figure 3-1, and the modeled sea level after 10 days of simulation is shown in Figure 3-2.

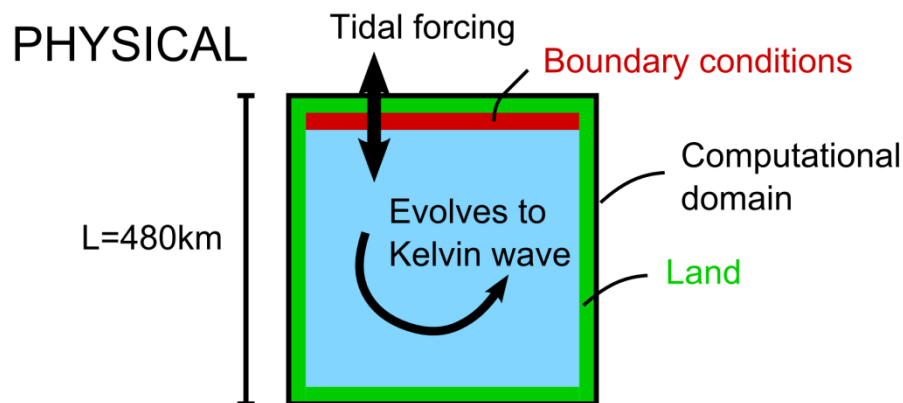


Figure 3-1: Model setup with tidal forcing and Kelvin wave propagation direction shown

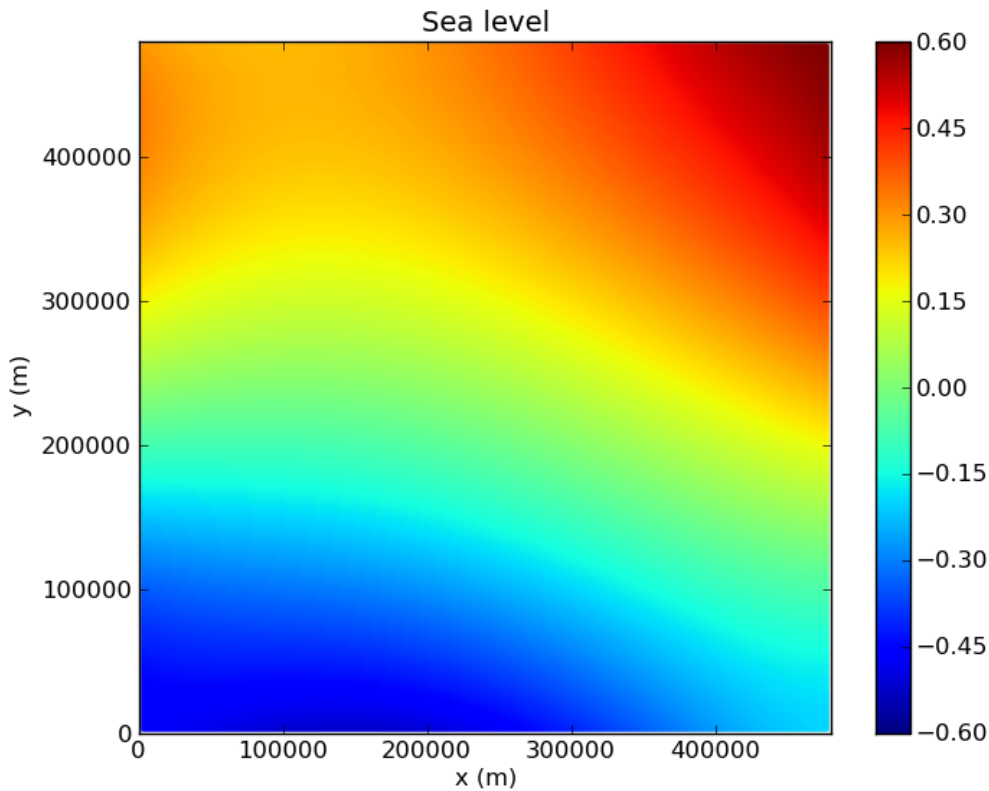


Figure 3-2: Model sea level after 10 days of simulation.

3.2 Numerical setup

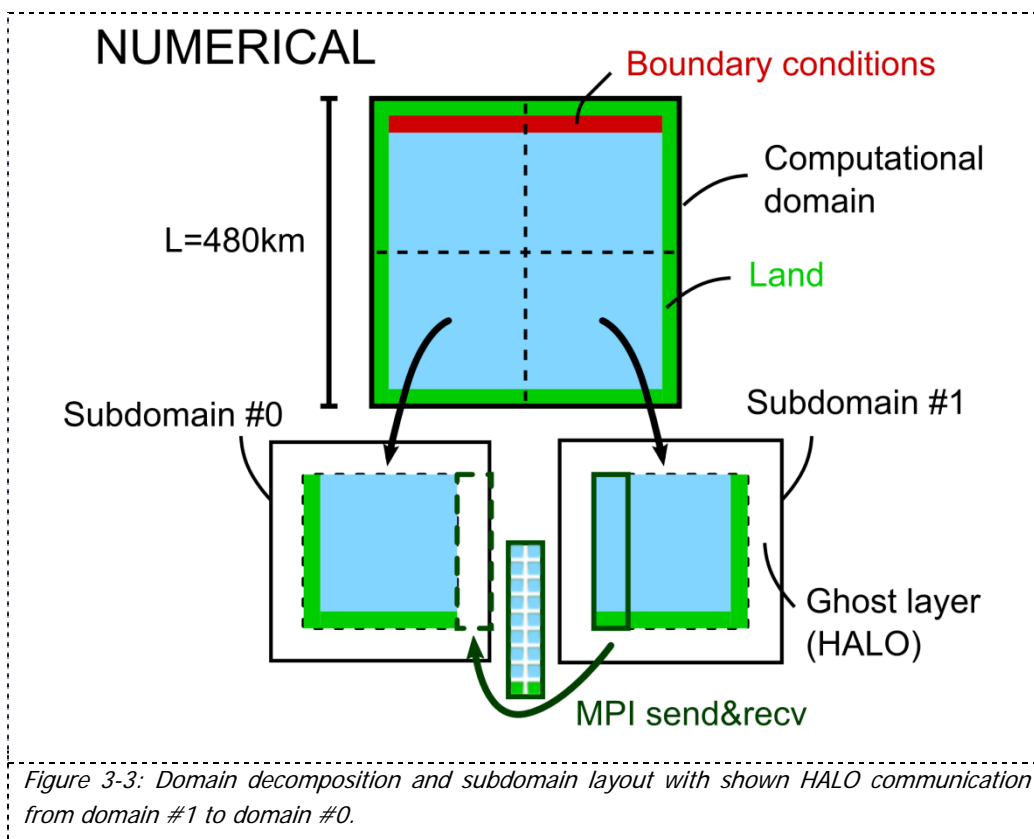
The model domain is split into rectangular subdomains as shown in Figure 3-3. The present version of GETM uses static allocation of many variables, as it has been shown to result in faster code (it also supports dynamic allocation). Thus, the subdomain size must be known at compile time. As the number of subdomains (and consequently the subdomain size) is a variable in our experiments we have created a script, which can compile multiple versions of the code facilitating easy examination of several different subdomain size layouts.

In Figure 3-3, the communication between two of the subdomains is also shown. The data from the interior of a domain are copied to the ghost layer, the so-called HALO zone, of the neighboring domain. Such communication will be denoted “HALO-exchange” and it plays a major role in the present work. In fact, the principal result of the work is to examine if and how it is possible to speed up the computations by implementing latency hiding for the HALO exchanges in GETM. Each subdomain has up to eight different neighbors, four in the principal directions and four with which it shares a single corner. In practical coding, there are always exactly eight neighbors, but some of the neighbors may be defined with `MPI_PROC_NULL`, so the MPI library will take care of suppressing the communication to those (non-existent) neighbors.

It is possible to use MPI and OpenMP together. This combination of coarse grained and

fine grained parallelism has the potential to – in some situations – improve the performance further compared to pure MPI domain decomposed code: OpenMP can use the shared-memory system to take advantage of the fast communication within a node. Also, the use of OpenMP has the potential to limit the total number of subdomains, leading to larger subdomains and smaller relative communication overhead. In practice, however, it has turned out that in the current version of GETM, pure MPI is faster than the OpenMP/MPI combination.

GETM uses mode splitting to allow different time steps for updating 2D and 3D fields. The 2D update is computationally cheap per time step but a small time step is required. The 3D part is computationally expensive but can run with a longer time step. In the present work, both pure 2D simulations and full 3D simulations are considered.



4 Implementation

The MPI component in GETM is implemented in a modular design, such that the actual MPI-calls are hidden. As mentioned earlier it is a strict design criteria (and good software practice) that the code clutter from MPI related changes in the GETM source code is minimized except in the MPI module itself, where it can be abstracted away from developers, who are mainly interested in other aspects of the code.

All HALO-exchanges in GETM are already coded as non-blocking communications. On the abstraction-level, the call of HALO-communications in GETM are divided into two parts, see Codebox 4-1. The first part (`update_2d_halo` or `update_3d_halo`) starts the communication with eight neighbors, and the second part (`wait_halo`) waits for all eight communications to complete. In the present/public code (e.g. the development track, git master HEAD⁸), the HALO subroutine calls are always paired, so that an update, which initiates the communication, is followed directly by a wait for all outstanding communications. Thus, there are no latency hiding implemented, but the structure to implement it is in principle available. As we shall see later, several major code changes are necessary to actually implement latency hiding. However, the actual communication parts, i.e. the core of `update_2d_halo`, `update_3d_halo` and `wait_halo`, will not be altered as part of the present work.

```
...
call update_3d_halo(f,...) ! Start SENDRECV with 8 neighbors
call wait_halo(D_TAG)      ! Wait for all 8 communications
...
```

Codebox 4-1: HALO-communication calls in GETM. The `D_TAG` is not actually used by `wait_halo`, which simply waits for all communications. As a result only a single set of communications are allowed at once

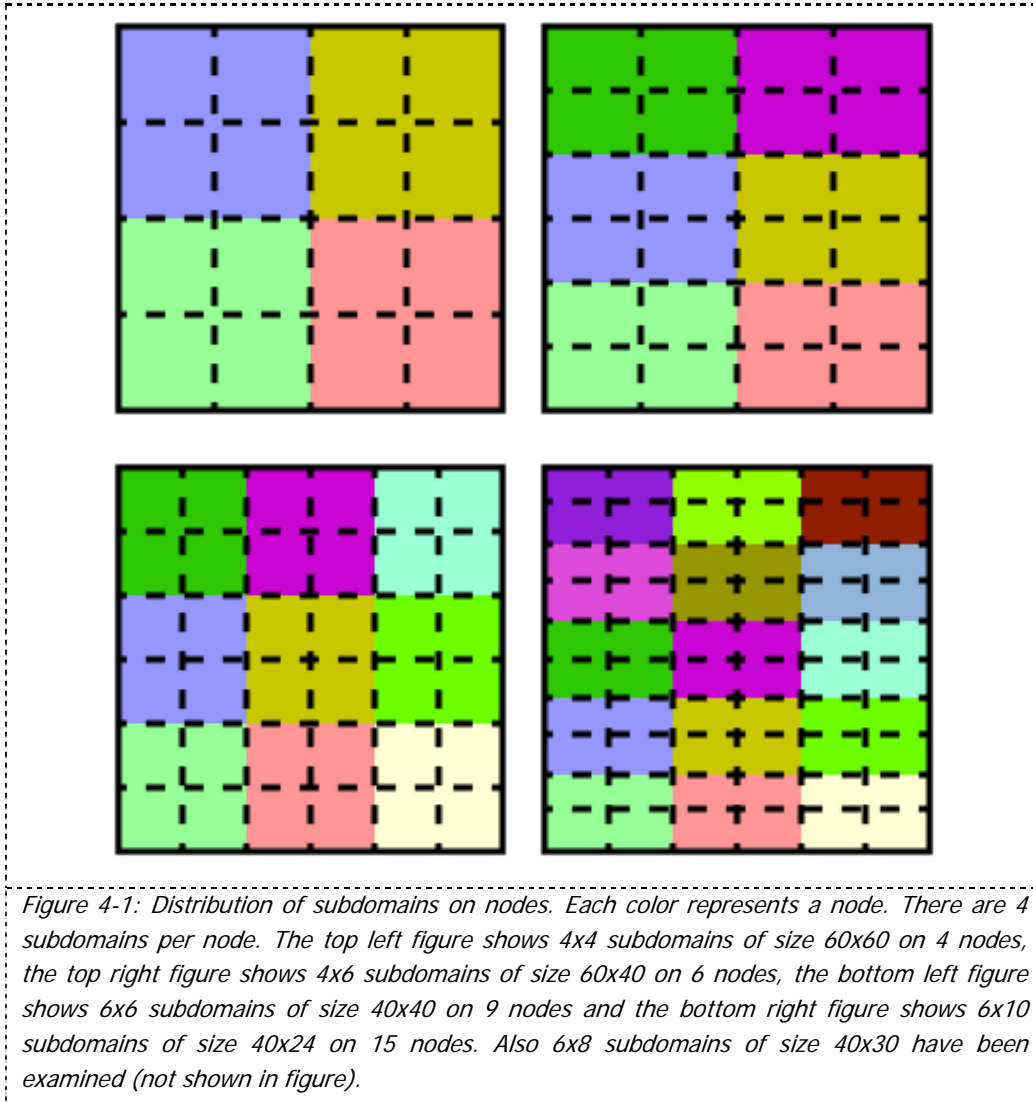
In GETM, calculations are only performed in “wet” points while land points are skipped. For practical applications this leads to a heterogeneous subdomain CPU cost. To make effective use of the computational resources an option could be to put more than 1 subdomain on each core. A reasonable load leveling is then found by a heuristic “subdomain division” tool⁹, which distributes the subdomains according to various (sometimes conflicting) criteria. Our test setup is, however, without land points except at the outer boundaries, so in this case the load leveling is trivial. Thus, the subdomains are distributed at exactly one subdomain (process) per CPU core, i.e. four subdomains per compute node.

In the present work, the subdomain size and the corresponding number of nodes are varied, keeping always 4 subdomains per node. The subdomain sizes used in the present work vary from 60x60 grid cells down to 40x24, i.e. roughly by a factor of four. The

⁸ GETM source code e.g. `git clone git://getm.git.sourceforge.net/gitroot/getm/getm getm-git`

⁹ Coded in Matlab. available as part of the “GETM-utils” package at Sourceforge:
<http://sf.net/p/getm-utils>

distributions of subdomains onto compute nodes are depicted in Figure 4-1. For comparison the operational setup features subdomains of sizes 60x60 (2D, 3nm model), 72x36 (3D, 1nm model) and 36x36 (3D, 600m model). When distributed to the nodes, the subdomains of the present test case are divided into (2x2) blocks of four, shown with colors in the figure.



GETM writes elevation and other output to files in a format called Network Common Data Form (NetCDF), with one netCDF file written for each subdomain. The `ncmerge`¹⁰ tool is then used to join the output files into a single NetCDF file. The resulting file can subsequently be compared with corresponding NetCDF files from other experiments – including serial execution of the code – to ensure the correctness of the implementation. To increase the validity of the timing experiments, NetCDF output is written only at end-of-run for these experiments, such that I/O is minimized during the computations.

As part of the present work, the NetCDF output was modified, such that it is possible, at

¹⁰ Coded in C, available as part of the “GETM-utils” package, <http://sf.net/p/getm-utils>

compile time, to select if output should be in single (4 byte real) or double (8 byte real) precision. Previously, only single precision output was supported. The increased output precision makes it easier to locate potential errors, especially relating to parallel implementation in comparison to serial runs.

Prior to the present work, GETM has been instrumented with a timer module, which uses the built-in `system_clock` Fortran subroutine to keep track of system (wall) time. The module provides reasonably accurate timers for the main parts of the code, as well as a number of test-timers, which can be used during development. The precision is achieved by using long integers for the counters. The present system and compiler reports a count rate of 1,000,000 clicks per second, corresponding to a timer granularity of around 1 μ s. The module provides subroutines for "tic" and "toc" of each timer, and each timer stores cumulative time as well as number of times it has been used. Status on each timer is written at end of simulation. The entire module can be disabled by a compilation flag if a user values speed over the timing information.

GETM runtime settings are controlled by Fortran namelists. For the present work, the execution of the experiments has been automated by modification of relevant namelist parameters on a scripting level. This approach allows easy modification and repetition of the experiments as necessary.

5 Initial timing of reference experiments

In order to assess the potential for speedup by latency hiding in GETM, a series of timing tests (both 2D and 3D) are conducted. In each test, the model problem is solved for a specified period: 10 days for 2D-only runs and 2 days for 3D.

To find how many calls actually being made, the HALO routines (`update_2d_halo` and `update_3d_halo`) have been instrumented with simple counters, and status written to log at end of simulation. The total number of calls found, were divided by, respectively, the number of 2D and 3D time steps. For the present scenario, in a setup, which resembles the operational environment at DaMSA, we found 3 2D HALO-updates per micro (2D) time step and 19 3D HALO-updates per macro (3D) time step. The 2D HALO updates are obviously for velocity (u,v) and sea surface elevation (H). The updates made in connection with the macro time steps (3D) are more complicated to find, and a break-down of those will be given later in the present work.

5.1 2D reference test

The main timing results for the 2D reference experiment are given in Table 5-1 (excerpt from Appendix A.) for five of the eight examined subdomain divisions. There are several general points to be made right away from these results. Initially, it can be noted that for subdomains of equal size, the ones with largest i -dimension (i.e. $i_{max} > j_{max}$) seem to run slightly faster than the ones with larger j -dimension ($j_{max} > i_{max}$). This is not very pronounced, but it is consistent for all the tests we have been running – both the present 2D reference tests and the remaining tests. As the difference is consistent (albeit small – around 1% or so), it is attributed to (slightly) better use of the cache for $i_{max} > j_{max}$. In practical scenarios, a developer should test several different subdomain divisions (and several distributions onto the nodes) to find one with good performance. In our experience, a tool such as Ganglia Monitoring System¹¹ for the cluster can be very informative during such a process. Already from watching the Ganglia page for the cluster, while the 2D reference case is running, it is evident, that the program does not fully exploit the CPU resources available (not shown here).

From Table 5-1, it is noted that the runtime (wall clock time) decreases with the number of subdomains, so there is a positive effect of increased parallelization. However, the runtime does not decrease linearly with the number of processes, see Figure 5-1, so there is a significant serial part of the code, which prohibits effective parallelization. If the total HALO time (about 35 seconds) is divided by the number of exchanges (43200 time steps and 3 exchanges per time step¹²), we get an exchange time on the order of 0.27ms. In comparison the ping-time between two nodes on the present system (gigabit Ethernet) is around 0.13ms, and this must be seen as an absolute minimum for the expected communication time. The communication time to all eight neighbors is only about twice the ping time, which indicates that the network time is primarily dominated by delay and less influenced by throughput. The amount of data transferred for a typical domain size

¹¹ Ganglia Monitoring System, <http://ganglia.sourceforge.net/>

¹² Without the so-called emergency brake option enabled, see discussion later

(say, 50x50) is roughly $2 \times 50 \times 4$ (HALO x width x #sides) reals, each of 8 bytes, i.e. about 3,200 bytes, or 25,600 bits. On a gigabit network, this amount of data would correlate to a throughput time of about 0.025ms, or about one fifth of the ping time. Even allowing for package size corrections, it is clear that the data amount to be transferred in 2D is so small that the HALO-exchange time in 2D are dominated by net delay on the present system. This is further underlined by the fact that the HALO time does not depend significantly on the subdomain size. The HALO time remains pretty constant even though the amount of data sent (per node or per subdomain) decreases with the subdomain size.

#subdomains	16	24	36	48	60
Dimension ($i_{max} \times j_{max}$)	60x60	60x40	40x40	40x30	40x24
Total runtime [s]	107.10	83.96	72.81	64.31	59.58
Halo runtime [s]	32.50	33.78	37.07	36.28	36.01
update_2d_halo [s]	8.07	7.83	8.43	8.83	9.00
wait_halo [s]	24.43	25.95	28.64	27.45	27.01
Non-halo runtime [s]	74.60	50.19	35.73	28.03	23.57
Momentum - total [s]	40.97	35.65	33.86	31.23	29.86
Momentum - halo [s]	23.71	23.90	25.63	24.77	24.46
Sealevel [s]	10.50	10.93	11.94	11.78	11.70
Sealevel - halo [s]	9.07	10.14	11.38	11.34	11.33

Table 5-1: Selected wall-time results for 2D 10-day run of "plain vanilla" version of GETM without latency hiding. Time step=20s. More data are given in Appendix A.

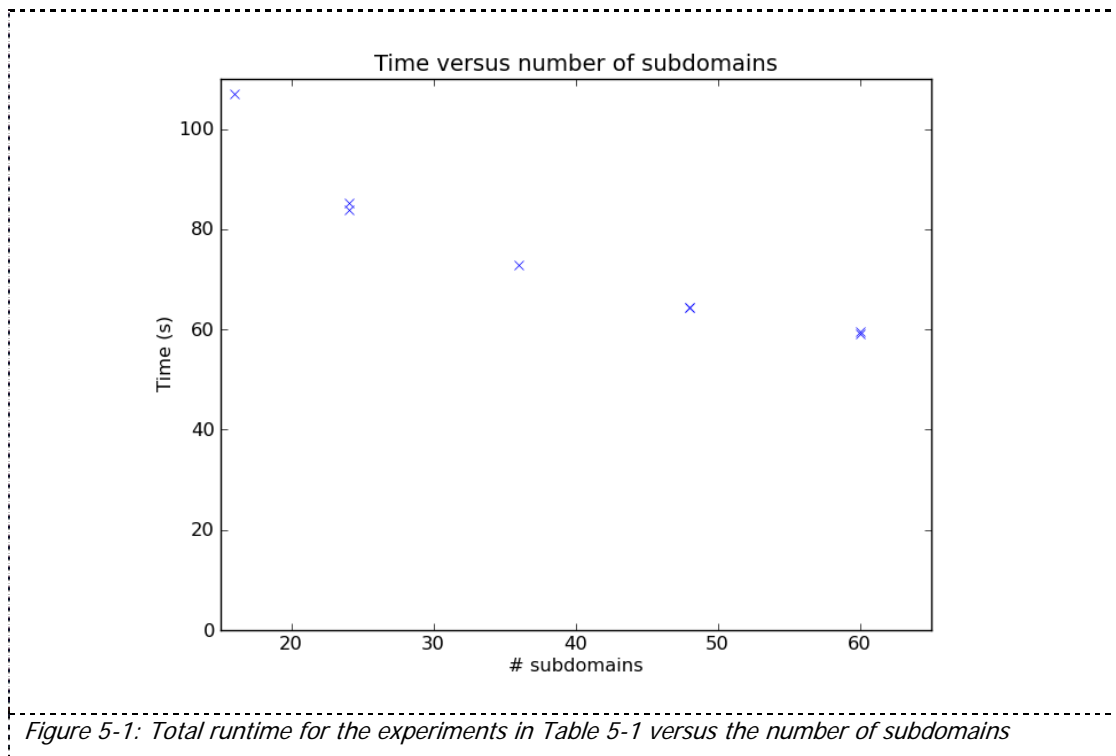


Figure 5-1: Total runtime for the experiments in Table 5-1 versus the number of subdomains

Out of the total HALO time, about one quarter is the initiation of communication (`update_2d_halo`), which we cannot hope to improve by latency hiding. But about 75% of the HALO time is wait, which we perceivably could hope to overlay by computations. However, if the HALO-part of the time is compared to the total time, then it can be seen that it increases from about 30% for size 60x60 subdomains to 60% for size 40x24. Even worse, the HALO-exchanges are parts of the sealevel and momentum subroutines, and only the computations of those routines can be used to hide the communication latency (since the updated fields are required right after these routines). For the smallest subdomains, the HALO part of momentum and sealevel routines amount to, respectively, 82% and 97% of the total subroutine time. Thus, there is very little compute time with which to hide the network latency in 2D. As a consequence, we do not expect to gain any significant positive effect from the implementation of latency hiding in 2D. Even so, the implementation will be done to verify these initial findings.

In Figure 5-2, the speedup of the 2D reference experiments from Table 1 is depicted. The experiment with 16 subdomains on 4 nodes is used as basis for calculating the speedup. The number of subdomains on the abscissa has therefore also been scaled with 1/16. Linear scaling will thus be a straight line with unity slope. For the present 2D model problem, the parallel fraction of the work is estimated to 60%, resulting in poor scaling even with relatively few subdomains.

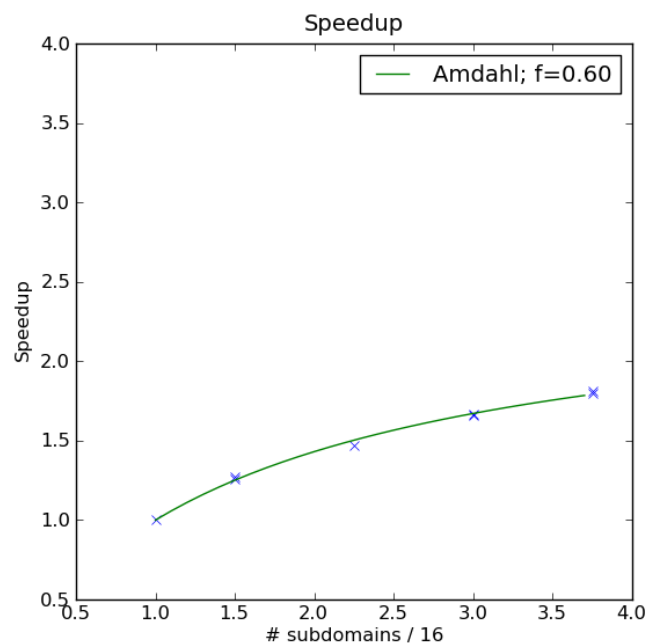


Figure 5-2: The blue crosses show the speedup of the 2D reference simulations compared to the base simulation with 16 subdomains. The number of subdomains on the x axis is therefore normalized with 16. The green line is a fit of Amdahls Law with an estimated parallel fraction of 0.60

5.2 3D reference test

The main timing results for the 3D reference experiment are given in Table 5-2 (excerpt from Appendix A2). Most importantly, it is noted that there seems to be a good speedup of the computations, when the number of processes (subdomains) is increased. This can be confirmed if the speedup – based on the 4-node case as base – is computed, see Figure 5-3. It is evident, that the speed-up in 3D is better than in 2D, but even so, the HALO-communication gets increasingly important and rises from 13% to 31% of the total run time, when the subdomain size is reduced. Even for the smallest subdomains, where the least amount of data are transferred per exchange, the time for `wait_halo` is about five times larger than the `update_halo` part¹³. It is noted that especially the time for `wait_halo` decreases with subdomain size, as the amount of data send per subdomain (and per node) decreases. That indicates that the data size in the HALO impacts the time, i.e. that the net throughput is important. If the amount of data send is computed, then the same conclusion may be reached. Data size is 60 times larger in 3D, as there are 60 layers in 3D compared to the single “layer” in the 2D computation. Consequently, the estimated time (based on throughput) is also 60 times larger, i.e. roughly 1.5 Mbit corresponding to 1.5ms at 1 Gbit/s¹⁴. This is more than 10 times the ping time, so throughput should be important. The HALO-time for a (temperature) 3D exchange is given in Table 5-2. If the temperature HALO-time (about 3 s) is divided by the number of 3D time steps (864), then an estimate of 3.4ms per 3D HALO-exchange can be found. Again, this is in agreement with the previous findings, that the throughput matters and that 3D data exchange take significantly longer than a simple ping.

#subdomains	16	24	36	48	60
Dimension (<code>i_{max} x j_{max}</code>)	60x60	60x40	40x40	40x30	40x24
Total runtime [s]	789.78	530.16	370.29	281.54	236.46
Halo runtime [s]	105.49	98.39	91.17	78.32	73.01
Non-halo runtime [s]	684.29	431.77	279.12	203.21	163.45
<code>update_*d_halo</code> [s]	20.07	17.11	14.73	13.15	12.17
<code>wait_halo</code> [s]	85.42	81.29	76.44	65.18	60.84
<code>do_temperature - total</code> [s]	156.67	104.61	72.25	55.52	44.90
<code>do_temperature - halo</code>	3.19	3.38	3.13	2.66	2.51

Table 5-2: Selected wall-time results for 3D 2-day run of “plain vanilla” version of GETM without latency hiding. Macro time step=200s. More data are given in Appendix A.

If the HALO-part of the runtime (Table 5-2) is compared to the total runtime, then it can be seen that it increases from about 13% for size 60x60 subdomains to 30% for size 40x24 (or 24x40, see Appendix). Thus, there might be enough computations available to

¹³ Note that the `update_halo` and `wait_halo` here sums results for 2D and 3D exchanges.

¹⁴ The actual time may be larger due to communication overhead, but also smaller, as some of the data are send internally on each node at a faster rate.

hide the net latency. If we look just at the temperature routine, then the communication uses only 5-6% - even for the smallest subroutines. Thus, it might be feasible to employ latency hiding - at least in the temperature routines. It might be problematic, however, that even though the temperature routine (for the smallest subroutines) use 27% of the non-HALO time, it is responsible for only one of 19 (5%) 3D HALO exchanges per time step. Thus, other 3D HALO exchanges are expected to have a smaller amount of computations available with which to hide the net latency.

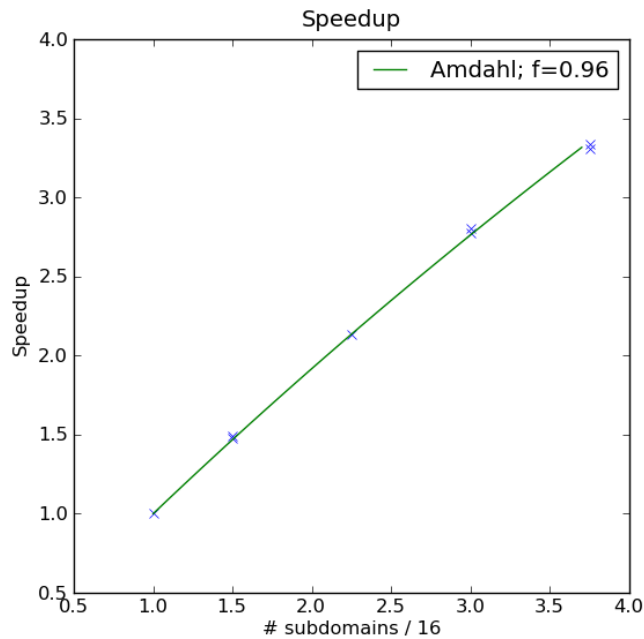


Figure 5-3: The blue crosses show the speedup of the 3D reference simulations compared to the base simulation with 16 subdomains. The number of subdomains on the x axis is therefore normalized with 16. The green line is a fit of Amdahls Law with an estimated parallel fraction of 0.96.

6 Latency hiding – 2D

The potential for speed-up by latency hiding in the 2D part of the code was determined in the discussion of the 2D reference tests above. The conclusion was that the computations that can potentially be used for hiding the latency are taking just a fraction of the time used for the communications. Therefore, it is not expected that latency hiding will impact the timings very much. Even so, latency hiding has been implemented for some of the algorithms in GETM.

6.1 Latency hiding principle

The current version of GETM does not employ latency hiding, but as part of the present work, latency hiding has been implemented for the sealevel (continuity) and velocity (momentum) equations. In GETM, the 2D communications are included in the subroutine, which also computes the updates to the arrays (`sealevel` and `momentum`). The latency hiding is implemented by moving the computational parts to separate “work” routines, to which start and stop indices are passed so that computations are performed only for parts of the subdomain. In general, four calls to the work routine are made to compute the outer part of the domain, see Figure 6-1. Thereafter, the communications are initiated and the inner part of the domain is computed. Finally, the code waits for the communications to finish. See Codebox 6-1 for an example.

To reduce the clutter from the code update, we introduced a preprocessor macro function. The main idea was to let the macro expand to four calls to an arbitrary subroutine for updating zones 1-4. The macro was intended to take the name of the subroutine, indexing information and a variable number of additional arguments for the subroutine.

Unfortunately, contrary to the CPP preprocessor, the Fortran Preprocessor (FPP) used by the Intel compiler cannot handle variable number of input variables. As a consequence, we have decided to not include preprocessor directives in an updated version of the code.

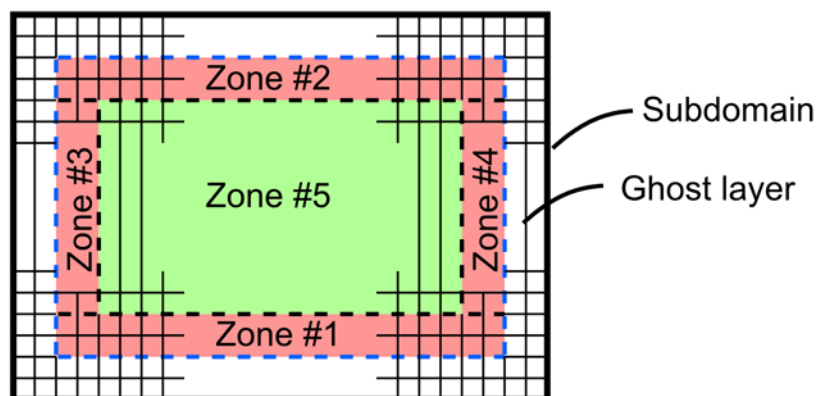


Figure 6-1: Latency hiding. Zones 1-4 are calculated prior to commencing communications while Zone 5 is calculated during the communications to hide the latency. The dashed blue rectangle denotes the border between the “inner” subdomain and the Halo zone (ghost layer). The principle for individual cells are shown in and near the two-cell wide Halo zone. The width of the halo zone in GETM is $HALO=2$.

```

subroutine sealevel(...)
...
call sealevel_work(imin      , imax      , jmin      , jmin+HALO-1) ! Zone 1
call sealevel_work(imin      , imax      , jmax-HALO+1, jmax      ) ! Zone 2
call sealevel_work(imin      , imin+HALO-1, jmin+HALO  , jmax-HALO  ) ! Zone 3
call sealevel_work(imax-HALO+1, imax      , jmin+HALO  , jmax-HALO  ) ! Zone 4
call update_3d_halo(z,...,z_TAG)
call sealevel_work(imin+HALO  , imax-HALO  , jmin+HALO  , jmax-HALO  ) ! Zone 5
call wait_halo(z_TAG)
...

```

Codebox 6-1: Coding example for 2D latency hiding. Note that the inner (non-halo) part of the subdomain (red and green parts in Figure B) are defined as $(imin:imax, jmin:jmax)$, and that $HALO=2$. The entire array is defined as shape $(imin-HALO:imax+HALO, jmin-HALO:jmax+HALO)$.

6.2 Sea level implementation

The sea level update in GETM is performed using the vertically integrated continuity equation. The sea level change is thus determined by the horizontal barotropic velocities and the freshwater flux through the sea surface. To avoid instabilities at shallow areas GETM also features a so-called “emergency brake” option, which iteratively sets barotropic currents to zero at the edges of grid cells where the updated water depth at a given time step is below a certain threshold. This means that the braking can propagate during subsequent iterations of the algorithm.

As a consequence of the emergency brake implementation, the latency hiding is not straightforward if braking is enabled. Simply put, it cannot be known if the sea level updates in zones 1-4 are finished before the update in zone 5 has been calculated. There are other optimizations that might be possible for the braking case. However, for the present work, only the case with no emergency brake will be considered. Thus, the emergency brake option is not employed in any of the test experiments. If the latency hiding for the non-brake code should prove efficient, then it is possible to revisit the “emergency brake” part of the code.

6.3 2D momentum implementation

The 2D momentum update is split into two parts, updating first for the y-direction and subsequently for the x-direction. In the next call to the 2D momentum update, the order is reversed and so on. The major portion of the update can be hidden without any problems. The update of the Coriolis force term is, however, not hidden since it is semi-implicit and requires the updated barotropic velocity components at the surrounding grid cell interfaces.

6.4 2D results

The results from the 2D latency hiding experiments are shown in Table A3 in the appendix. Comparing Tables A1 and A3 we see that the latency hiding does not improve the performance compared to the control experiments. In Figure 6-2, the timing of the 2D

latency hiding experiments as a function of the number of subdomains are shown. The corresponding speedup - again with 4 nodes as basis - is shown in Figure 6-3. The figures are almost identical to, respectively, Figures 5-1 and 5-2, meaning that there is no significant influence on the overall runtimes from the 2D latency hiding.

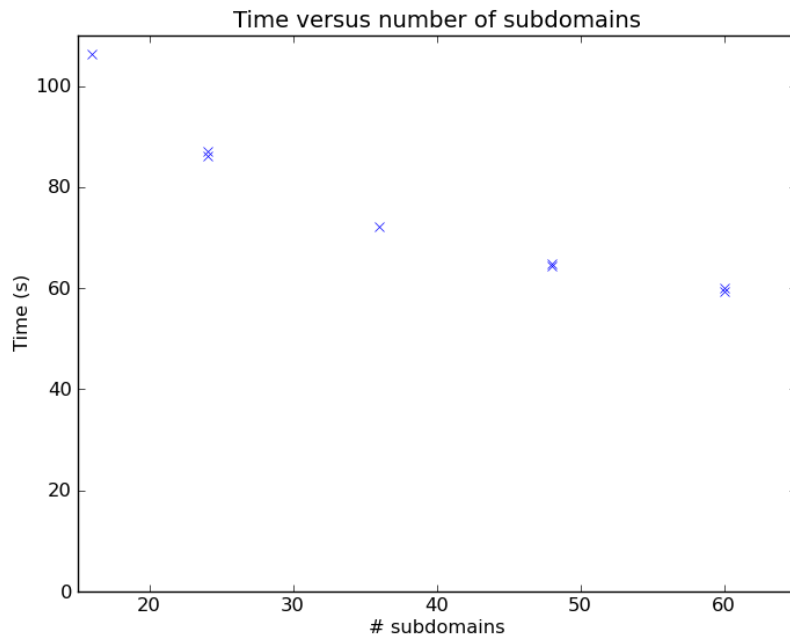


Figure 6-2: Total runtime for the 2D latency hiding experiments versus the number of subdomains

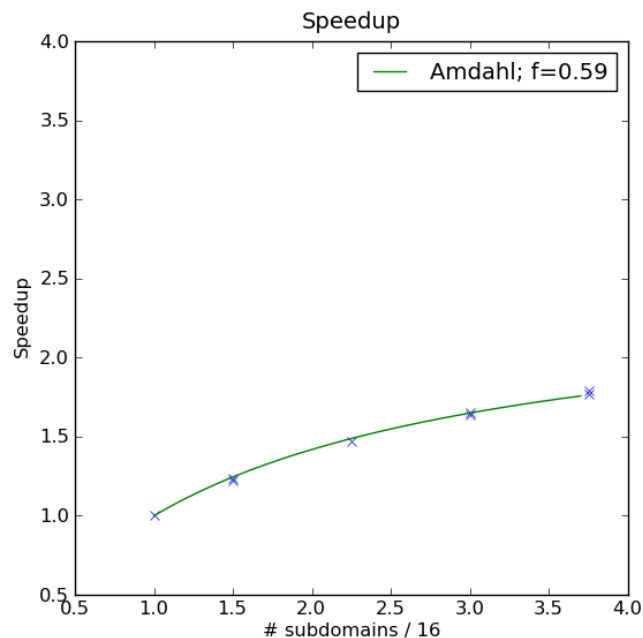
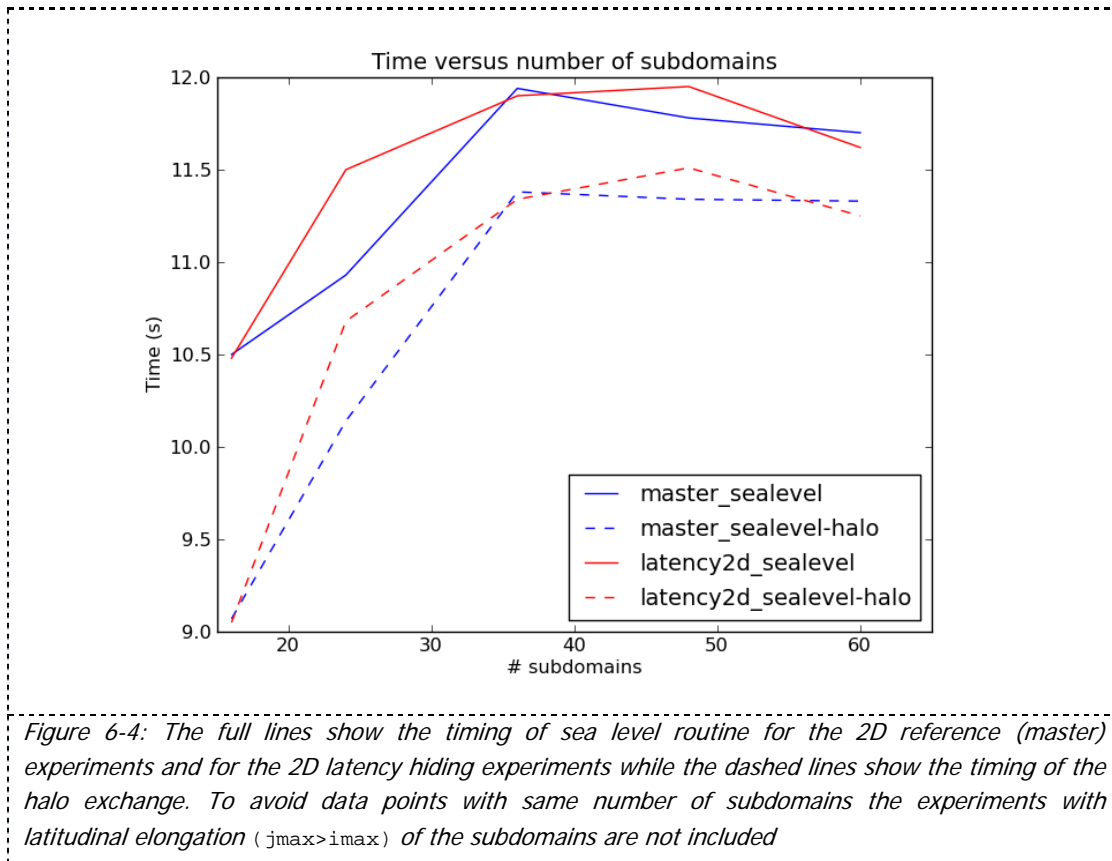


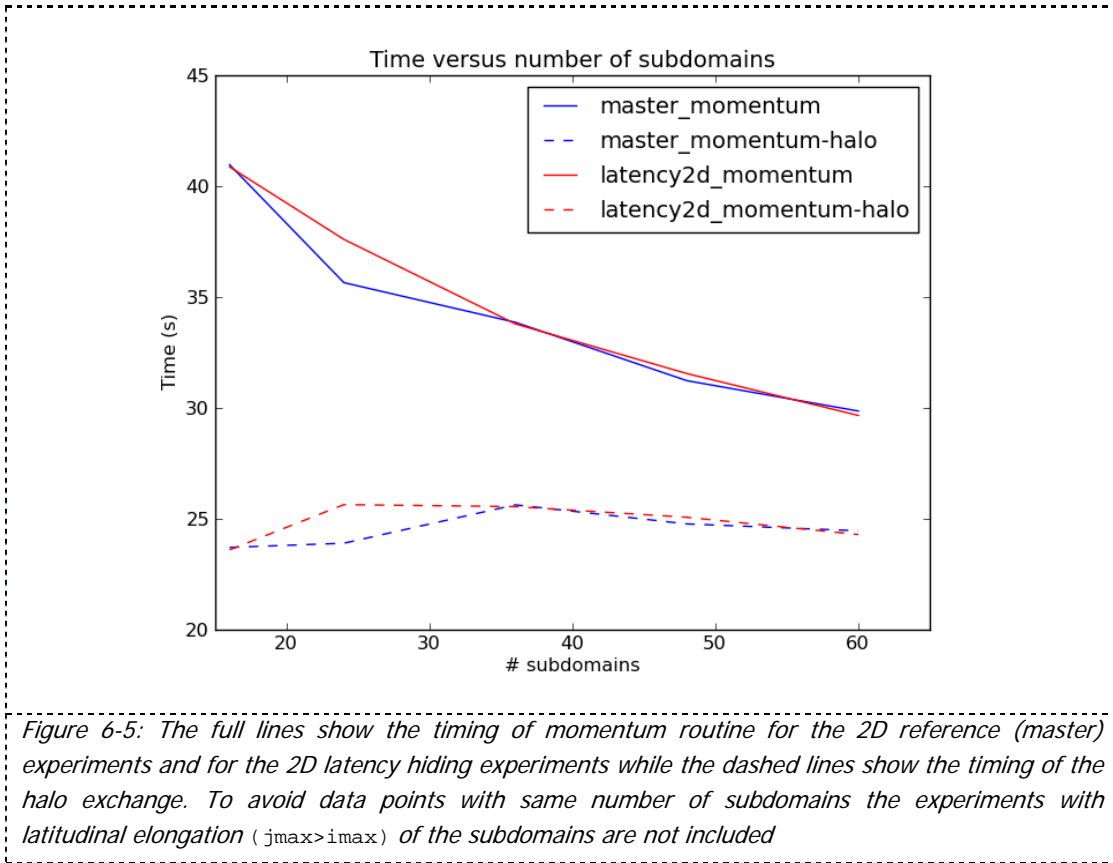
Figure 6-3: The blue crosses show the speedup of the 2D latency hiding simulations compared to the base simulation with 16 subdomains. The number of subdomains on the x axis is therefore normalized with 16. The green line is a fit of Amdahls Law with an estimated parallel fraction of 0.59



Timing results for the `sealevel` subroutine are given in Figure 6-4, for both the reference experiment and for the implementation including latency hiding implemented. It is noted that the latency hiding does not improve the performance. This is in agreement with the expectations described in Section 5.1. The poor results are attributed to the lack of “appropriate work” with which to overlay the communication, as well as to a worse cache performance in the updated code: the division of one (double) loop into five, as shown in Figure 6-1, increases the number of cache misses, and this limits the computational speed.

In Figure 6-5, timings for the `momentum` subroutine are shown, corresponding to the results for the `sealevel` subroutine given in Figure 6-4. Also for the `momentum` subroutine, it can be noted that the latency hiding does not result in any performance improvements. If anything, the updated code including latency hiding is slower overall than the “plain vanilla” version of GETM. Importantly, if Figures 6-4 and 6-5 are compared, then it can be seen that the scaling properties of the `sealevel` and `momentum` subroutines are very different. While the `momentum` subroutine gets faster, as the subdomains get smaller and the overall workload per subdomain is decreased, the `sealevel` subroutine runtime does not improve for smaller subdomains.

It is evident that a faster network between the nodes is necessary in order to achieve better scaling. The communication latency cannot be effectively hidden given the present latency of the network.



7 Latency hiding – 3D

A priori, the potential for speed-up by latency hiding in the 3D part of the code seems reasonable, at least on the present DaMSA hardware, see Section 5.2. However, it is not known, which parts of the code are most used and possible candidates for latency hiding implementation. In particular, it needs to be determined, how the 19 HALO-updates per time step occur in the code.

7.1 3D HALO exchange break-down

If the GETM code is examined¹⁵, then it can be found, that 2D-halo updates are coded in 52 places, while 3D-halo updates are coded in 44 places. Many are related to initial setup, such as sync after read of hotstart files, or sync before output. Such calls are only executed once or a few times per simulation, so there is not really a point in spending time on tuning those communications very much. For the 2D case, the three dominating HALO exchanges have already been identified: one for `sealevel`, and two for 2D `momentum` (emergency brake option not included). For the present model problem, it is also known that there are 19 3D HALO exchanges per time step, but a priori it is not known where in the source code, the 19 calls are located. In fact, prior to the present study, not even the exchange count was well known among the developers, and the sheer number of exchanges has seemed to surprise most of the developers.

To find the actual places, where the 3D HALO calls are made, the code has been compiled in debug mode (with `-g` flag) and run through the Intel Debugger. A break point was set in the `update_3d_halo` routine, and each time the break point was reached, a stack dump was made to get the call structure. The process was repeated for a few time steps to make sure consistent results were obtained. In this manner, the location of all 19 calls to 3D HALO update has been found, see Table 7-1. It is noted, that 10 of the 19 exchanges are due to the high-order advection method used for temperature and salinity. If more tracers are added, or if high-order advection is used for momentum, then the number of these exchanges will increase even more. Thus, it seems reasonable that if effective latency hiding is to be employed in GETM, then it must first be successful in the higher-order advection schemes.

There are several "1D directional split" advection schemes in GETM (P2PDM, TDV-Superbee, TDV-MUSCL, TVD-P2PDM). The implementations of the schemes are extremely similar, and only a small part of the core method differs among the schemes – the main loop structure is identical¹⁶. Of the several methods implemented, only the P2PDM method, which is in use operationally at DaMSA, is chosen for the present feasibility study. The 1D directional split method used takes $\frac{1}{2}$ time step in u , $\frac{1}{2}$ in v , a full in w , $\frac{1}{2}$ in v and $\frac{1}{2}$ in u . There are three different routines to do fractional-step (directional split) parts of the computation, one for each main direction (x,y,z) or (u,v,w) . The three subroutines

¹⁵ E.g. with `"grep -i update_2d_halo */**F90 | grep -i call | wc -l"`. Code from March 2011

¹⁶ The schemes are actually coded with a single loop structure with a deeply nested case-construct. The choice is not necessarily good for performance, but it keeps the amount of identical/redundant code down

are very similar in structure, so it should suffice to examine just a single of them for feasibility of latency hiding. Thus, we choose (arbitrarily) initially to examine the u-direction (only) of the TVD-P2PDM. If it turns out to be feasible, then implementation of v- and w-direction equations, and extension to the other implemented advection schemes, will follow almost trivially.

GOTM: Integration of turbulence variables (uuEx and vvEx)	2
Integration of salinity	1
Integration of temperature	1
Momentum equation (3D, uu,vv,ww)	3
Momentum advection (low-order method, uu,vv)	2
Salinity advection (1D directional split u/v/w/v/u)	5
Temperature advection (1D directional split u/v/w/v/u)	5
TOTAL	19

Table 7-1: 3D HALO exchanges in GETM with advection scheme "110" (upstream) for momentum and "661" (TVD-P2PDM) for salinity and temperature

7.2 GETM 3D advection code structure

For the 3D advection, the HALO-exchange is decoupled from update of the fields, as the overall routine (do_advection_3d) first calls subroutines to update the fields, and subsequently calls the HALO-exchange routine, see Codebox 7-1.

```

subroutine do_advection_3d(dt,f,uu,vv,ww,...)
...
select case (hor_adv)
  case (UPSTREAM) ! Low-order method
    ...
  case ((UPSTREAM_SPLIT),(P2),(Superbee),(MUSCL),(P2_PDM),(FCT))
    hi=ho
    select case (adv_split)
      case (0) ! Simple 1D split
        ...
      case (1) ! Higher-order 1D split
        call u_split_adv(dt,f,uu,...)
        call update_3d_halo(f,...,D_TAG)
        call wait_halo(D_TAG)
        call v_split_adv(dt,f,vv,...)
        call update_3d_halo(f,...,D_TAG)
        call wait_halo(D_TAG)
        ... ! Continue with w_split_adv, v_split_adv, u_split_adv(dt,f,...)
      case (2) ! 2D split
        ...
    end select
  ...
end select
...
end select
...

```

Codebox 7-1: Call structure for 3D advection in "plain vanilla" GETM, without latency hiding. The red code shows how the HALO-calls are separated from the advection computations. Timer routines are not shown

In order to implement latency hiding, the HALO calls must first be placed inside the routines, which actually computes the advection and updates the tracer arrays, in this case the routine `u_split_adv`. This update is easily made, and the changes to `do_advection_3d` are shown in Codebox 7-2. The necessary changes to `u_split_adv` will be discussed in the following sections.

```

subroutine do_advection_3d(dt,f,uu,vv,ww,...)
...
select case (hor_adv)
case (UPSTREAM) ! Low-order method
...
case ((UPSTREAM_SPLIT),(P2),(Superbee),(MUSCL),(P2_PDM),(FCT))
hi=ho
select case (adv_split)
case (0) ! Simple 1D split
...
case (1) ! Higher-order 1D split
call u_split_adv(dt,f,uu,...) ! Includes HALO calls
call v_split_adv(dt,f,vv,...)
call update_3d_halo(f,...,D_TAG)
call wait_halo(D_TAG)
... ! Continue with w_split_adv, v_split_adv
call u_split_adv(dt,f,uu,...) ! Includes HALO calls
case (2) ! 2D split
...
end select
...
end select
...

```

Codebox 7-2: Call structure for 3D advection in GETM with latency hiding for `u_split_adv`. Timer routines are not shown

7.3 3D advection and the Arakawa C-grid

In order to understand the advection scheme used in `u_split_adv`, it is necessary to introduce some information about the discretization used. In the horizontal, the GETM model is defined on a so-called staggered C-grid (Arakawa & Lamb, 1977), see Figure 7-1. Often, the grid is interpreted as a set of cells, with pressure (or tracer) points at the cell centers. At the centers of the cell faces velocity points (U-points and V-points) are defined. Thus, each velocity point always lies exactly in the middle of two pressure points, and the pressure/tracer points lie in the middle of four velocity points. This ordering of the grid is convenient for many of the finite difference and finite element evaluations, but it also complicates the bookkeeping somewhat.

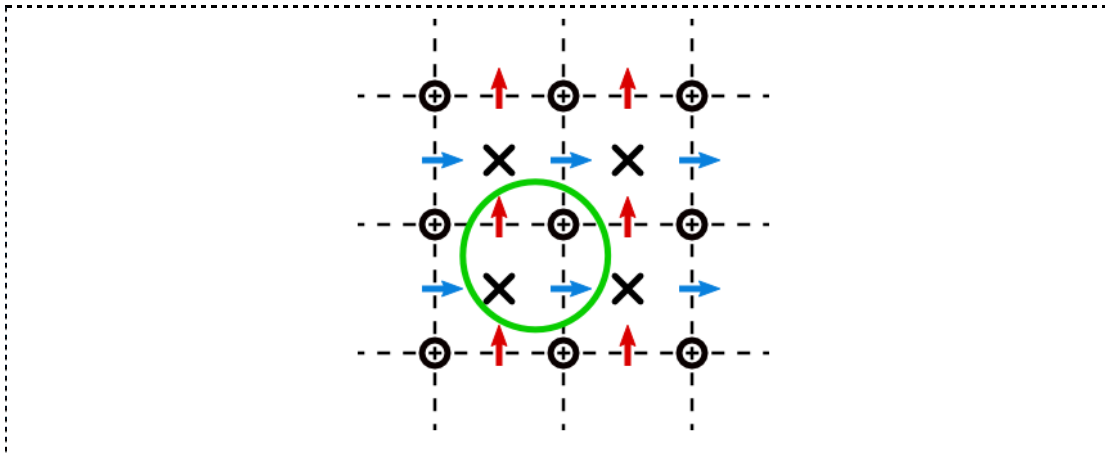


Figure 7-1: Staggered C-grid. The tracer/pressure points (T-points, denoted by X) are located in the centre of each "cell" (dashed lines). Velocity points are located at the centre of each cell face (u points: blue horizontal arrows; v-points: red vertical arrows). Corners of cells are denoted X-points (small black circles). The four points inside the green circle have identical indices (i, j)

In principle, the discretization of the u-directional fraction of the 3D advection step in GETM consists of the following three steps:

- A. Compute interface concentrations (CU) based on old/existing tracer values (T), with an (up to) four-point central scheme:

$$CU_{i,j,k} = \text{func1}(T_{i-1,j,k} , T_{i,j,k} , T_{i+1,j,k} , T_{i+2,j,k})$$
- B. Update tracer values (T) based on computed interface concentrations (CU) with a two-point central scheme:

$$T_{i,j,k} = \text{func2}(CU_{i-1,j,k} , CU_{i,j,k})$$
- C. Exchange tracer values with neighbor subdomains

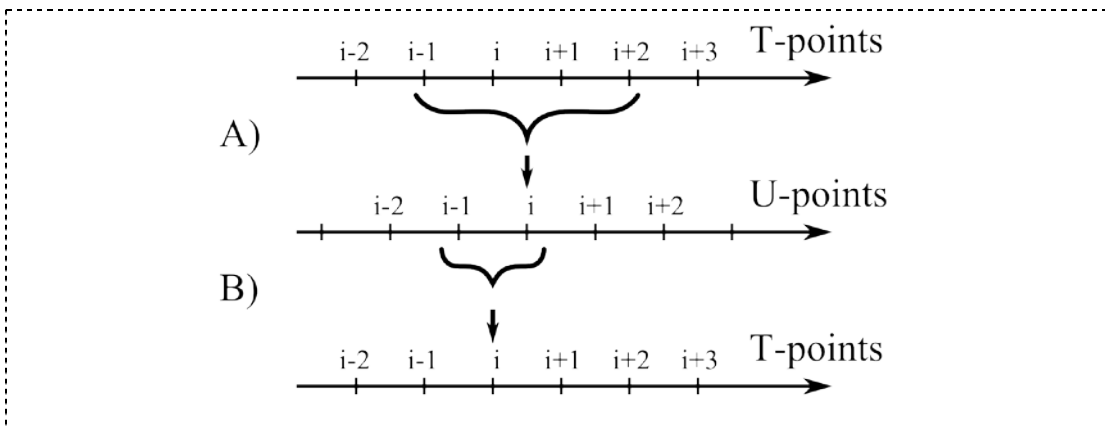


Figure 7-2: Stencil for the u-direction of the 1D directional split advection scheme. A: Computation of interfacial concentrations from tracer values. B: Update of tracer values from interfacial concentrations

The stencils used in steps A and B are shown in Figure 7-2. It should be noted, that due to the C-grid, the interface concentration (defined on U-points) are staggered half a grid cell from the tracer points (T-points). Latency hiding can be implemented for step B with zones

identical to what is used in 2D, see Figure 6-1. However, due to the larger stencil in step A and the staggered nature of the grid, the zones need to be changed for step A, if latency hiding is to be implemented in that case, see Figure 7-2.

In the present ("plain vanilla"/without latency hiding) GETM code, the `u_split_adv` sub-routine uses two nested case constructs to compute the interfacial concentrations (Step A in Figure 7-2), see Codebox 4-1.¹⁷ In the codebox, the important/critical parts of the code are shown with color. It should be noted, that steps A1 and A3 are common for the higher-order advection schemes, so only step A2 differs among the schemes. This is the reason for the inner-most select-case construct.

To include latency hiding in `u_split_adv`, the core work (steps A and B) have been parcelled out to work routines, where the loop limits are dummy variables, just as in 2D. In order to get rid of the inner-most select-case and still maintain reusable code, the common parts of step A (in particular steps A1 and A3, see Codebox 7-3) are moved to include files, which are then included from the work routine for the P2PDM. This will make it easier to create work-routines for the remaining advection schemes (MUSCL etc.). The implemented work routines are shown in Codebox 7-4. The changes to the `u_split_adv` itself are shown in Codebox 7-5. It should be noted, that the zone indices for parts A and B, see Figure 3-1, are not identical due to the staggered grid and the size of the discretization stencils.

¹⁷ The select/case deeply inside the triple-DO loop could be expensive, unless the compiler can reorder the loops and case constructs

```

subroutine u_split_adv(f,uu,...)
...
! Part A: Compute interfacial values CU
select case (method)
  case (UPSTREAM_SPLIT)
    ...
  case ((P2),(Superbee),(MUSCL),(P2_PDM))
    do k=1,kmax
      do j=jmin,jmax
        do i=imin-1,imax
          ! 1: Compute common scalar helper variables
          fu=... ! upstream
          fc=... ! central
          fd=... ! downstream
          c=...; r=... ; ...
          ! 2: Compute scheme-specific scalar helper variables
          select case (method)
            case ((P2),(P2_PDM))
              x = one6th*( _ONE_-_TWO_*c)
              Phi=( _HALF_+x)+( _HALF_-x)*r
              ...
            case (Superbee)
              ... ! Different Phi
            case (MUSCL)
              ... ! Yet another Phi
          end select
          ! 3: Compute interface concentrations
          cu(i,j,k)=... Phi ...
          ...
        end do
      end do
    end do
  end select

! Part B: Compute advection (common for all schemes):
do k=1,kmax ! Doing the u-advection step
  do j=jmin,jmax
    do i=imin,imax
      if (az(i,j) .eq. 1) then ! (Not on land)
        hio(i,j,k)=hi(i,j,k)
        hi(i,j,k)=hio(i,j,k) - ...
        f(i,j,k)=(... cu(i,j,k) ... cu(i-1,j,k)...)/hi(i,j,k)
      end if
    end do
  end do
end do
...

```

Codebox 7-3: u_split_adv in GETM without latency hiding. Timer routines are not shown.

```

subroutine u_split_adv_work_P2_PDM(f,uu,...,i1,i2,j1,j2)
...
  do k=1,kmax
    do j=j1,j2
      do i=i1,i2
        if (au(i,j) .gt. 0) then
#include "u_split_adv_1.h" ! setup common scalar
          x = one6th*( _ONE_-_TWO_*c)
          Phi=( _HALF_+x)+( _HALF_-x)*r
          limit=max( _ZERO_,min(Phi,_TWO_/(_ONE_-c), &
            _TWO_*r/(c+1.d-10)))
#include "u_split_adv_2.h" ! Do common interfacial values
          else
            cu(i,j,k) = _ZERO_
          end if
        end do
      end do
    end do
  end subroutine u_split_adv_work_P2_PDM

subroutine u_split_adv_work_advstep(f,uu,...,i1,i2,j1,j2)
...
  do k=1,kmax ! Doing the u-advection step
    do j=j1,j2
      do i=i1,i2
        if (az(i,j) .eq. 1) then ! (Not on land)
          hio(i,j,k)=hi(i,j,k)
          hi(i,j,k)=hio(i,j,k) - ...
          f(i,j,k)=(... cu(i,j,k) ... cu(i-1,j,k)...)/hi(i,j,k)
        end if
      end do
    end do
  end do
end subroutine u_split_adv_work_advstep

```

Codebox 7-4: Work routines for u_split_adv in GETM with latency hiding. Timer routines are not shown.

```

subroutine u_split_adv(f,uu,...)
...
select case (method)
case (UPSTREAM_SPLIT)
...
case (P2_PDM)
! Interfacial values (Part A, Red zones)
call u_split_adv_work_P2_PDM(f,uu,..., imin-1, imax, jmin, jmin+HALO-1)!Z1
call u_split_adv_work_P2_PDM(f,uu,..., imin-1, imax, jmax-HALO+1, jmax )!Z2
call u_split_adv_work_P2_PDM(f,uu,..., imin-1, imin+HALO+0, jmin+HALO, jmax-HALO )!Z3
call u_split_adv_work_P2_PDM(f,uu,..., imax-HALO-1, imax, jmin+HALO, jmax-HALO )!Z4
! Advection step (Part B, Red zones)
call u_split_adv_work_advstep(f,uu,..., imin, imax, jmin, jmin+HALO-1)!Z1
call u_split_adv_work_advstep(f,uu,..., imin, imax, jmax-HALO+1, jmax )!Z2
call u_split_adv_work_advstep(f,uu,..., imin, imin+HALO-1, jmin+HALO, jmax-HALO )!Z3
call u_split_adv_work_advstep(f,uu,..., imax-HALO+1, imax, jmin+HALO, jmax-HALO )!Z4
! Start SENDRECV
call update_3d_halo(f,...,D_TAG)
! Interfacial values (Part A, Green zone)
call u_split_adv_work_P2_PDM(f,uu,..., imin+HALO+1, imax-HALO-2, jmin+HALO, jmax-HALO )!Z5
! Advection step (Part B, Green zone)
call u_split_adv_work_advstep(f,uu,..., imin+HALO, imax-HALO, jmin+HALO, jmax-HALO )!Z5
! Wait for SENDRECV
call wait_halo(D_TAG)
case (Superbee) ! Other schemes...
...
end select
...

```

Codebox 7-5: u_split_adv in GETM with latency hiding. Timer routines are not shown.

7.4 3D results

Test-timers have been introduced in `u_split_adv` in the “plain vanilla” version of GETM, to be able to check how the new implementation compares to the old one. Timings for the 3D reference test (no latency hiding) are shown in Table 7-2. It is noted that a significant part of the HALO time is “wait”, which we may hope to reduce by latency hiding. The computational work (non-HALO) is divided into a part, which is executed before `update_3d_halo` is called, and a part executed between `update_3d_halo` and `wait_halo`. These parts are denoted, respectively, “red zone” and “green zone” work. For the plain version examined in Table 7-2, the “green zone” work is zero, as all the computations are done before `update_3d_halo` is called (no latency hiding). Further, the computational work is divided into computation of interfacial concentrations (Part A in Figure 14) and subsequent computation of the advection step (Part B).

In Table 7-2 it is noted that parts A and B are approximately equally expensive for “large” subdomains. As the subdomain size is reduced by a factor of 3.75 (in area and computational work), the computational time of Part A is reduced equivalently - by roughly a factor 3.5. Thus, there is a reasonably good scaling for Part A. However, for Part B, the computational time is reduced by around a factor 7, i.e. there is super-linear scaling for Part B. If the actual code is examined, see Codebox 7-3, then it is noted that while Part B has just a few computations within each loop, Part A includes computation of a number of scalars. Also, Part A has a deeply nested case-construct, and many of the scalar-computations are conditional on e.g. velocity sign (not shown). Thus, it is reasonable to believe, that there is a better usage of the cache in Part B, as the subdomain size is lowered. With a decreasing number of cache misses, the computations of Part B have a

great cache advantage, while the same speedup is not gained for the computations of Part A.

#subdomains	16	24	36	48	60
Dimension (imax x jmax)	60x60	60x40	40x40	40x30	40x24
u_split_adv – total [s]	100.25	67.12	46.83	35.71	29.21
u_split_adv – update-halo [s]	4.22	3.73	3.03	2.53	2.25
u_split_adv – wait-halo [s]	14.16	14.03	12.73	10.61	9.98
u_split_adv – “red” zones [s]	81.85	49.35	31.06	22.55	16.98
u_split_adv – “green” zones [s]	0.00	0.00	0.00	0.00	0.00
u_split_adv – Part A [s]	41.09	28.22	19.19	14.33	11.65
u_split_adv – Part B [s]	40.73	21.10	11.85	8.20	5.31

Table 7-2: Wall-time results for test timers in `u_split_adv` for 3D 2-day run of “plain vanilla” version of GETM without latency hiding

#subdomains	16	24	36	48	60
Dimension (imax x jmax)	60x60	60x40	40x40	40x30	40x24
u_split_adv – total [s]	109.55	70.66	50.90	37.15	28.46
u_split_adv – update-halo [s]	3.49	3.14	2.61	2.22	2.01
u_split_adv – wait-halo [s]	5.31	5.17	4.62	4.38	4.30
u_split_adv – “red” zones [s]	30.54	20.91	18.11	13.02	9.87
u_split_adv – “green” zones [s]	70.19	41.42	25.53	17.52	12.26
u_split_adv – Part A [s]	55.29	38.12	28.21	20.19	15.55
u_split_adv – Part B [s]	45.40	24.17	15.39	10.31	6.56

Table 7-3: Wall-time results for test timers in `u_split_adv` for 3D 2-day run of GETM with latency hiding.

Timings of `u_split_adv` for the modified version of GETM with latency hiding implemented are shown in Table 7-3. First, it is noticed that the HALO-wait is reduced by roughly a factor 2 to 3 compared to the “plain” version (Table 7-2). A significant amount of computational time has also been shifted to the “green zone”, i.e. between `update_3d_halo` and `wait_halo`. Thus, the computations actually do hide some of the communication time. However, the total run time for the subroutine is not reduced for any of the subdomain sizes. It can be seen that this is because the computational time increases for both Parts A and B. It is conjectured that the increase in computational time is due to the “zoning”, and the fact that it reduces the effectiveness of the cache. The data for especially zones 3 and 4 are not consecutively located in memory, and thus there will

be an increase in the number of cache misses. For the computations of Part A, this leads to an increase of the computational time of around 30%. For Part B, the slowdown seems to depend on the subdomain size, but for the smaller subdomains it is in the same order of magnitude as for Part A.

7.5 3D alternative approach

An alternative latency hiding approach has been implemented, where only the computations of “Part B” (Figure 7-2) are used for latency hiding. For Part A, the entire array is precomputed to reduce the computational overhead of the “zoning”. The new work-routine for the P2PDM scheme is retained, but only called once, with the full range for loop limits. The resulting code is shown in Codebox 7-6, with changes from the previous implementation shown in red.

```

subroutine u_split_adv(f,uu,...)
...
select case (method)
case (UPSTREAM_SPLIT)
...
case (P2_PDM)
! Interfacial values (Part A, All zones: no latency hiding)
call u_split_adv_work_P2_PDM(f,uu,..., imin-1, imax, jmin, jmax )!Z5
! Advection step (Part B, Red zones)
call u_split_adv_work_advstep(f,uu,..., imin, imax, jmin, jmin+HALO-1)!Z1
call u_split_adv_work_advstep(f,uu,..., imin, imax, jmax-HALO+1, jmax )!Z2
call u_split_adv_work_advstep(f,uu,..., imin, imin+HALO-1, jmin+HALO, jmax-HALO )!Z3
call u_split_adv_work_advstep(f,uu,..., imax-HALO+1, imax, jmin+HALO, jmax-HALO )!Z4
! Start SENDRECV
call update_3d_halo(f,...,D_TAG)
! No computation of interfacial values/Part A here
! Advection step (Part B, Green zone)
call u_split_adv_work_advstep(f,uu,..., imin+HALO, imax-HALO, jmin+HALO, jmax-HALO )!Z5
! Wait for SENDRECV
call wait_halo(D_TAG)
case (Superbee) ! Other schemes...
...
end select
...

```

Codebox 7-6: u_split_adv in GETM with alternative latency hiding, where “Part A” is not used for latency hiding

Timing results for `u_split_adv` with the modified latency hiding code is given in Table 7-4. First, it is noticed that the timings for Part A correspond closely to the results for the unmodified code, see Tabel 7-2. The added computational time for Part A has thus been eliminated as planned. It also means that the updated code without a deeply nested case-construct is not faster than the original code. Thus, it is concluded that the deeply nested case-constructs in the advection routines are not important for the performance – at least for the present compiler, flags and hardware.

The time used on wait-HALO is increased compared to the initial latency hiding implementation (Tabel 7-3), but it is still lower than for the unmodified code (Tabel 7-2), so a significant amount of latency hiding is still retained. However, as the computational time for Part B is still higher than for the unmodified code, the impact on the total time for the routine is not impressive. For square subdomains or when $j_{\max} > i_{\max}$, see Appendix A., the latency hiding provides no advantage. When $i_{\max} > j_{\max}$, there is a small

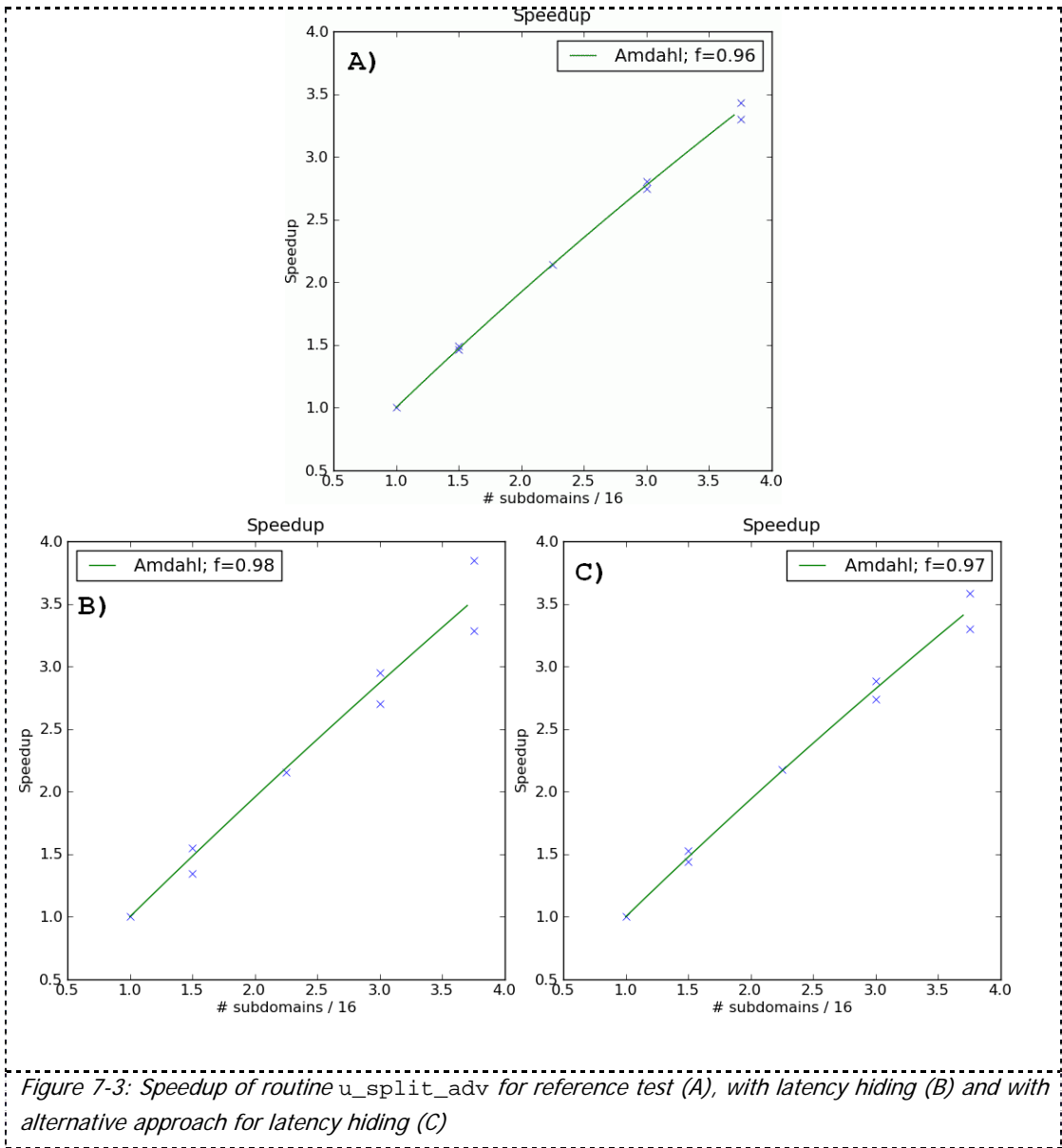
advantage. But even though the communication in the 40x24 size case takes more than 40% of the routine run time in the original code, the latency hiding makes the routine only around 4% faster. If the cluster had been equipped with 100Mbit/s Ethernet network, then we might have had a significant effect, but with Gigabit Ethernet the effect is marginal at best. If the cluster had a faster low-latency network, such as Infiniband or Myri10G, then the net latency would be much smaller, but the complexity due to the zoning would remain, leading without doubt to a disadvantage to the modified code compared to the unmodified code without latency hiding.

#subdomains	16	24	36	48	60
Dimension (i_{max} x j_{max})	60x60	60x40	40x40	40x30	40x24
<code>u_split_adv</code> – total [s]	100.35	65.57	46.14	34.76	27.97
<code>u_split_adv</code> – update-halo [s]	4.25	3.68	2.81	2.34	2.09
<code>u_split_adv</code> – wait-halo [s]	7.27	7.52	7.58	6.90	6.97
<code>u_split_adv</code> – “red” zones [s]	52.65	35.57	25.11	18.52	14.56
<code>u_split_adv</code> – “green” zones [s]	36.16	18.78	10.63	6.99	4.34
<code>u_split_adv</code> – Part A [s]	41.48	28.40	19.45	14.56	11.72
<code>u_split_adv</code> – Part B [s]	47.29	25.91	16.26	10.92	7.16

Table 7-4: Wall-time results for test timers in `u_split_adv` for 3D 2-day run of GETM with latency hiding for the work of Part B only

The speedup of the `u_split_adv` routine has been computed for both the reference test and the two versions with latency hiding, see Figure 7-3. It can be seen that the latency hiding gives a larger spread in the results, i.e. that the “skewness” of the subdomains have a larger impact on the results with latency hiding than in the reference case. Thus, for the “zoning” it is increasingly important to choose subdomains with $i_{max} > j_{max}$. Such subdomains have smaller Zones 3 and 4, see e.g. Figure 6-1, and these zones are expected to have the largest number of cache misses. So decreasing the size of these zones, there might be a better utilization of the cache, resulting in faster execution of the code.

From Figure 7-3, it is further noted that if latency hiding is introduced, then the parallel fraction increases. However, at least for the initial implementation of latency hiding (Figure 7-3B), the apparent increase in parallel fraction hides the earlier mentioned decrease in speed in general: all runs relating to Figure 7-3B are slower than the corresponding runs in Figure 7-3A.



8 Conclusions

Latency hiding has been implemented in GETM to evaluate the possible computational speedup of the method. The code has been examined in both modified and unmodified versions on a Gigabit Ethernet-based development cluster at DaMSA. It has been shown that for a setup, which is very similar to operational setups at DaMSA, network communication takes up a large time of the wall clock time of GETM. For the most important 3D setups, it seems to be around 25% of the execution time, which is related to communication.

Initial reference tests have shown that not much can be gained by 2D latency hiding, and subsequent experiments including latency hiding confirm this conclusion. The routines that update the 2D fields, which must be communicated to other processes, are dominated by communications, while computations take only a small fraction of the total execution time. Furthermore, the communications are dominated by initial delay and not throughput. It is conjectured that the "data zoning" used in latency hiding increases the number of cache misses for the computations, and consequently results in a slower computational part of the code. It is concluded, that latency hiding is not effective for the 2D mode in GETM.

For the 3D communication, it has been found that there are 19 data exchanges per time step in GETM for the operational setup in use at DaMSA. The locations of all 19 have been identified, and it has been proved that 10 of the 19 relate to higher-order advection of tracers (salinity and temperature). As a consequence, the higher-order advection scheme has been selected to test latency hiding for 3D communication.

In the 3D latency hiding experiments the results are more interesting, but only slightly more encouraging, than the 2D ones. Implementation of latency hiding by "data zoning" can reduce the communication latency of the higher-order advection scheme significantly. However, the computational "data zoning" results in computational blocks, which are less able to take advantage of the cache. Especially two of the computational zones will lead to many cache misses per computation. As a consequence, implementation of latency hiding results in a code, where execution time is increasingly dependent on the shape of the subdomain. In essence, "fat" subdomains are faster than "tall" ones, even though the number of computations (and communications) is identical for the two cases. In the initial latency hiding implementation, the increase in cache misses turned out to slow down the code more than what was gained on the communication side. Therefore, an alternative implementation has been carried out, in which only a part of the computations are used for latency hiding. The result is a version of GETM, which is slightly faster than the unmodified code. However, the substantial changes to the code - and especially increased maintenance complexity - does weight against updating "main" GETM to include latency hiding. In addition, it is speculated that on a fast network, such as Infiniband, the advantages of latency hiding will be reduced, while the computational disadvantages of "zoning" remain. Thus, GETM users running on newer clusters or super-computers would not gain an advantage with latency hiding. Thus, if latency hiding is implemented, it should allow a compile-time method to disable it and regain higher computational performance.

During the testing of 3D results it was found that a particular select-case construct, which is found deep inside a triple-DO loop, does not seem to adversely affect the computation time.

It is strongly advised that the coming cluster at DaMSA implements a fast network, such as Infiniband, in a cluster-update, which is planned for early 2012. The present Gigabit Ethernet network limits the scaling, which is possible on the hardware.

There are ways to decrease the network communications, which have not been examined in the present work. For instance, the initial delay might be decreased if persistent communications are implemented. This can be tested with little work in 2D, while the 3D implementation could require a larger effort.

Alternatively, the present "one size fits all" approach to communication in GETM, where all communications are for a full HALO width (2 cells) and in all direction (8 neighbors) could be changed. That would require looking at each communication in turn and updating just enough to keep the given array synchronized. For the higher-order advection schemes, which require five subsequent exchanges, it might be sufficient that the last one is a "full" exchange, which results in a fully synchronized data set.

Finally, for some exchanges it might be possible, to postpone the call to the wait-part of the data exchange until the data are actually needed for further computations. This approach would need dedicated communicators (which could be persistent) for each array, which is to be communicated in such a way. For some computations, this kind of speedup is not feasible, because the updated array is used for further computations immediately after it is updated and synchronized. Such challenges could form the basis of future student projects.

9 References

Arakawa, A., Lamb, V. R. (1977): *Methods of computational physics*, volume 17, pp 174-265. Academic Press.

Chapman, B., Jost, G. van der Pas, R. (2008): "Using OpenMP", ISBN-13: 978-0-262-53302-7. The MIT Press, Cambridge, Massachusetts, USA.

Eijkhout, V., Chow, E., van de Geijn, R., (2010): "Introduction to High-Performance Scientific Computing", 1st edition, The University of Texas at Austin, Texas, USA. Free on-line version available at: <http://www.lulu.com/product/file-download/introduction-to-high-performance-scientific-computing/15735386>

Ganglia Monitoring System, <http://ganglia.sourceforge.net/>

General Estuarine Transport Model (GETM), <http://getm.eu/>

General Ocean Turbulence Model (GOTM), <http://gotm.net/>

GETM utilities package, <http://sf.net/p/getm-utils>

Gropp, W., Lusk, E., Thakur, R. (1999): "Using MPI-2", ISBN-10 0-262-57133-1. The MIT Press, Cambridge, Massachusetts, USA.

Intel Fortran Compiler, ifort, Version 11.1.059 (Build 20091012)
<http://software.intel.com/en-us/articles/intel-compilers/>

MacDonald, N., Minty, E., Harding, T., Brown, S.: Writing Message-Passing Parallel Programs with MPI. Course notes, version 1.8.2, The University of Edinburgh, UK.
http://www2.epcc.ed.ac.uk/computing/training/document_archive/mpi-course/mpi-course.pdf

Message Passing Interface, Wikipedia article,
http://en.wikipedia.org/wiki/Message_Passing_Interface

Message Passing Interface Chameleon, MPICH2,
<http://www.mcs.anl.gov/research/projects/mpich2/>

Open Multi-Processing, OpenMP, Wikipedia article, <http://en.wikipedia.org/wiki/OpenMP>

A. Timing data

The GETM code is instrumented with counters, the status/contents of which are written at end of simulation. The present appendix contains data from the timers for all the simulations made. A perl script (parse_results.pl) has been programmed to average the timers over all subdomains of each simulation. The content of selected timers is included in this appendix, along with some derived variables such as wall time times and number of processes.

For the 3D runs, the routine `u_split_adv` has been instrumented with additional test timers to specifically examine the local effect of latency hiding in that particular routine.

A1: 2D basis run

2D RUN

10 days, DT(2D)=20.0, 43200 time steps

CASES:

```
RUN_sub04x04_master_2000m_M00 RUN_sub04x06_master_2000m_M00
RUN_sub06x04_master_2000m_M00 RUN_sub06x06_master_2000m_M00
RUN_sub06x08_master_2000m_M00 RUN_sub08x06_master_2000m_M00
RUN_sub06x10_master_2000m_M00 RUN_sub10x06_master_2000m_M00
```

#subdomains	16	24	24	36	48	48	60	60
Dimension	60x60	60x40	40x60	40x40	40x30	30x40	40x24	24x40
TIMERS SUM	107.10	83.96	85.19	72.81	64.31	64.49	59.58	59.18
sum halo	32.50	33.78	33.97	37.07	36.28	36.19	36.01	35.37
sum rest	74.60	50.19	51.23	35.73	28.03	28.30	23.57	23.82
sum total x np	1713.55	2015.12	2044.60	2621.07	3086.80	3095.42	3574.73	3551.01
sum rest x np	1193.54	1204.50	1229.41	1286.45	1345.45	1358.49	1414.17	1428.98
sum start_halo	8.07	7.83	7.95	8.43	8.83	8.63	9.00	8.86
sum wait_halo	24.43	25.95	26.02	28.64	27.45	27.56	27.01	26.50
momentum	40.97	35.65	35.82	33.86	31.23	31.42	29.86	29.38
momentum-halo	23.71	23.90	24.05	25.63	24.77	24.93	24.46	23.96
sealevel	10.50	10.93	10.97	11.94	11.78	11.80	11.70	11.50
sealevel-halo	9.07	10.14	10.18	11.38	11.34	11.35	11.33	11.13
do_temperature	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
temperature-halo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sum do_advection_3d	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
do_advection_3d halo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

sealevel and momentum called once per time step.

A2: 3D basis run

3D RUN, 60 layers
 2 days, DT(2D)=20.0, 8640 time steps
 M=10, DT(3D)=200.0; 864 macro time steps

CASES:

RUN_sub04x04_master_2000m_M10 RUN_sub04x06_master_2000m_M10
 RUN_sub06x04_master_2000m_M10 RUN_sub06x06_master_2000m_M10
 RUN_sub06x08_master_2000m_M10 RUN_sub08x06_master_2000m_M10
 RUN_sub06x10_master_2000m_M10 RUN_sub10x06_master_2000m_M10

#subdomains	16	24	24	36	48	48	60	60
Dimension	60x60	60x40	40x60	40x40	40x30	30x40	40x24	24x40
TIMERS SUM	789.78	530.16	534.84	370.29	281.54	284.60	236.46	238.84
sum halo	105.49	98.39	95.76	91.17	78.32	79.08	73.01	74.40
sum rest	684.29	431.77	439.08	279.12	203.21	205.52	163.45	164.43
sum total x np	12636.54	12723.96	12836.05	13330.52	13513.78	13660.76	14187.51	14330.29
sum rest x np	10948.69	10362.48	10537.88	10048.38	9754.23	9865.14	9806.71	9865.99
sum start_halo	20.07	17.11	18.19	14.73	13.15	13.91	12.17	12.74
sum wait_halo	85.42	81.29	77.57	76.44	65.18	65.17	60.84	61.66
momentum	13.49	11.82	10.54	10.56	9.38	9.45	9.28	9.25
momentum-halo	9.82	9.31	8.07	8.84	8.07	8.13	8.18	8.14
sealevel	2.77	2.67	2.56	2.87	2.85	2.87	3.01	2.99
sealevel-halo	2.45	2.49	2.39	2.75	2.76	2.77	2.93	2.91
do_temperature	156.67	104.61	106.29	72.25	55.52	56.23	44.90	46.34
temperature-halo	3.19	3.38	3.22	3.13	2.66	2.75	2.51	2.58
sum do_advection_3d	316.75	208.51	213.83	139.60	103.81	106.02	82.88	86.03
do_advection_3d halo	44.50	42.52	41.61	38.59	32.63	33.17	30.04	30.69
test-00	100.25	67.12	68.55	46.83	35.71	36.51	29.21	30.40
test-01	4.22	3.73	4.03	3.03	2.53	2.69	2.25	2.44
test-02	14.16	14.03	13.38	12.73	10.61	10.67	9.98	10.06
test-03	81.85	49.35	51.13	31.06	22.55	23.13	16.98	17.87
test-04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
test-05	41.09	28.22	28.44	19.19	14.33	14.54	11.65	11.91
test-06	40.73	21.10	22.66	11.85	8.20	8.57	5.31	5.94

Explanation test timers:

test-00: u_split_adv total
 test-01: u_split_adv update_3d_halo (start send)
 test-02: u_split_adv wait_halo (block for data sendrecv)
 test-03: u_split_adv computations before update_3d_halo
 test-04: u_split_adv computations after update_3d_halo
 test-05: u_split_adv computations of interface concentrations
 test-06: u_split_adv computations of advection (given interface concentrations)

The test-00 timer has been called 3456 during the 864 time steps, i.e. 4 times per time step.
 That is two for salinity and two for temperature.

A3: 2D latency hide run

2D RUN

10 days, DT(2D)=20.0, 43200 time steps

CASES:

RUN_sub04x04_latency2d_2000m_M00 RUN_sub04x06_latency2d_2000m_M00
RUN_sub06x04_latency2d_2000m_M00 RUN_sub06x06_latency2d_2000m_M00
RUN_sub06x08_latency2d_2000m_M00 RUN_sub08x06_latency2d_2000m_M00
RUN_sub06x10_latency2d_2000m_M00 RUN_sub10x06_latency2d_2000m_M00

#subdomains	16	24	24	36	48	48	60	60
Dimension	60x60	60x40	40x60	40x40	40x30	30x40	40x24	24x40
TIMERS SUM	106.27	87.14	86.08	72.24	64.85	64.32	59.27	60.04
sum halo	32.38	36.25	35.07	36.64	36.78	35.96	35.88	36.03
sum rest	73.89	50.89	51.01	35.60	28.06	28.36	23.39	24.02
sum total x np	1700.26	2091.32	2065.89	2600.47	3112.68	3087.50	3556.01	3602.63
sum rest x np	1182.18	1221.30	1224.26	1281.51	1347.03	1361.29	1403.33	1440.97
sum start_halo	8.11	8.15	7.87	8.14	8.97	8.68	9.07	8.92
sum wait_halo	24.27	28.10	27.20	28.50	27.81	27.28	26.81	27.11
momentum	40.87	37.60	36.77	33.79	31.55	31.16	29.66	30.02
momentum-halo	23.60	25.64	24.93	25.55	25.07	24.68	24.29	24.56
sealevel	10.48	11.50	11.21	11.90	11.95	11.71	11.62	11.68
sealevel-halo	9.05	10.68	10.40	11.34	11.51	11.27	11.25	11.30
do_temperature	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
temperature-halo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sum do_advection_3d	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
do_advection_3d halo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

sealevel and momentum called once per time step.

A4: 3D latency hide run – version A

3D RUN, 60 layers
 2 days, DT(2D)=20.0, 8640 time steps
 M=10, DT(3D)=200.0; 864 macro time steps

CASES:

RUN_sub04x04_latency3d_2000m_M10 RUN_sub04x06_latency3d_2000m_M10
 RUN_sub06x04_latency3d_2000m_M10 RUN_sub06x06_latency3d_2000m_M10
 RUN_sub06x08_latency3d_2000m_M10 RUN_sub08x06_latency3d_2000m_M10
 RUN_sub06x10_latency3d_2000m_M10 RUN_sub10x06_latency3d_2000m_M10

#subdomains	16	24	24	36	48	48	60	60
Dimension	60x60	60x40	40x60	40x40	40x30	30x40	40x24	24x40
TIMERS SUM	809.05	546.69	555.79	380.41	289.73	300.69	237.51	250.01
sum halo	112.72	103.89	97.87	90.00	77.41	79.26	69.36	73.61
sum rest	696.34	442.80	457.92	290.41	212.31	221.43	168.14	176.40
sum total x np	12944.86	13120.60	13338.92	13694.74	13906.93	14432.88	14250.31	15000.60
sum rest x np	11141.41	10627.20	10990.04	10454.68	10191.04	10628.62	10088.44	10584.02
sum start_halo	18.78	15.98	16.61	13.73	12.49	13.15	11.48	12.39
sum wait_halo	93.93	87.91	81.26	76.27	64.92	66.11	57.89	61.22
momentum	21.70	14.10	11.19	10.42	9.58	10.00	9.20	9.83
momentum-halo	17.97	11.58	8.68	8.70	8.26	8.67	8.10	8.71
sealevel	2.76	3.21	2.69	2.81	2.89	3.02	2.89	3.11
sealevel-halo	2.44	3.04	2.52	2.69	2.80	2.92	2.81	3.03
do_temperature	163.47	109.07	114.76	75.81	57.18	59.65	45.09	48.38
temperature-halo	4.96	5.18	4.62	4.33	3.47	3.54	3.04	3.21
sum do_advection_3d	328.03	216.93	229.59	147.19	107.91	112.68	83.86	90.83
do_advection_3d halo	42.70	41.65	39.98	35.67	29.63	30.67	26.46	27.79
test-00	109.55	70.66	81.23	50.90	37.15	40.55	28.46	33.34
test-01	3.49	3.14	3.27	2.61	2.22	2.42	2.01	2.22
test-02	5.31	5.17	5.42	4.62	4.38	4.71	4.30	4.79
test-03	30.54	20.91	29.51	18.11	13.02	15.43	9.87	13.41
test-04	70.19	41.42	43.02	25.53	17.52	17.97	12.26	12.90
test-05	55.29	38.12	44.60	28.21	20.19	21.24	15.55	17.47
test-06	45.40	24.17	27.89	15.39	10.31	12.13	6.56	8.81

Explanation test timers:

test-00: u_split_adv total
 test-01: u_split_adv update_3d_halo (start send)
 test-02: u_split_adv wait_halo (block for data sendrecv)
 test-03: u_split_adv computations before update_3d_halo
 test-04: u_split_adv computations after update_3d_halo
 test-05: u_split_adv computations of interface concentrations
 test-06: u_split_adv computations of advection (given interface concentrations)

The test-00 timer has been called 3456 during the 864 time steps, i.e. 4 times per time step. That is two for salinity and two for temperature.

A5: 3D latency hide run – version B

3D RUN, 60 layers
 2 days, DT(2D)=20.0, 8640 time steps
 M=10, DT(3D)=200.0; 864 macro time steps

CASES:

RUN_sub04x04_latency3dB_2000m_M10 RUN_sub04x06_latency3dB_2000m_M10
 RUN_sub06x04_latency3dB_2000m_M10 RUN_sub06x06_latency3dB_2000m_M10
 RUN_sub06x08_latency3dB_2000m_M10 RUN_sub08x06_latency3dB_2000m_M10
 RUN_sub06x10_latency3dB_2000m_M10 RUN_sub10x06_latency3dB_2000m_M10

#subdomains	16	24	24	36	48	48	60	60
Dimension	60x60	60x40	40x60	40x40	40x30	30x40	40x24	24x40
TIMERS SUM	788.64	531.26	538.75	376.05	283.15	286.37	231.64	242.50
sum halo	105.16	95.36	95.21	92.26	77.64	77.00	69.83	71.75
sum rest	683.48	435.90	443.54	283.79	205.51	209.37	161.81	170.75
sum total x np	12618.32	12750.17	12929.97	13537.91	13591.25	13745.89	13898.66	14549.72
sum rest x np	10935.73	10461.50	10644.87	10216.55	9864.38	10049.97	9708.73	10245.01
sum start_halo	19.66	16.79	17.30	14.18	12.64	13.51	12.01	12.51
sum wait_halo	85.51	78.57	77.92	78.08	65.00	63.48	57.82	59.24
momentum	13.43	11.66	10.38	12.52	9.80	9.47	8.87	9.14
momentum-halo	9.68	9.13	7.89	10.82	8.47	8.14	7.78	8.03
sealevel	2.86	2.63	2.55	2.91	2.95	2.83	2.82	2.90
sealevel-halo	2.53	2.45	2.38	2.79	2.85	2.74	2.74	2.82
do_temperature	157.81	105.01	107.51	72.79	55.27	56.45	44.25	46.33
temperature-halo	4.24	3.98	3.80	3.51	2.85	2.93	2.59	2.66
sum do_advection_3d	317.17	208.37	215.34	140.41	103.90	107.09	82.01	86.46
do_advection_3d halo	43.37	39.57	39.80	36.04	30.30	30.71	27.57	28.21
test-00	100.35	65.57	69.57	46.14	34.76	36.69	27.97	30.43
test-01	4.25	3.68	3.82	2.81	2.34	2.56	2.09	2.29
test-02	7.27	7.52	7.57	7.58	6.90	6.82	6.97	6.96
test-03	52.65	35.57	38.15	25.11	18.52	19.92	14.56	16.20
test-04	36.16	18.78	20.01	10.63	6.99	7.37	4.34	4.97
test-05	41.48	28.40	28.86	19.45	14.56	14.66	11.72	12.04
test-06	47.29	25.91	29.26	16.26	10.92	12.61	7.16	9.11

Explanation test timers:

test-00: u_split_adv total
 test-01: u_split_adv update_3d_halo (start send)
 test-02: u_split_adv wait_halo (block for data sendrecv)
 test-03: u_split_adv computations before update_3d_halo
 test-04: u_split_adv computations after update_3d_halo
 test-05: u_split_adv computations of interface concentrations
 test-06: u_split_adv computations of advection (given interface concentrations)

The test-00 timer has been called 3456 during the 864 time steps, i.e. 4 times per time step. That is two for salinity and two for temperature.