Report of Assignment One: A Distributed Advection Solver Using MPI

Hong-Bo Zhang

College of Engineering and Computer Science, Australian National University,
Canberra, 2601, Australia
u6170245@anu.edu.au

UID	u6170245
Course	COMP8300 Parallel System
Assign No.	Assignment 1
Group No.	Individual
Lecturer	Peter Strazdins
Date	2017-April-21

ABSTRACT: This is a report of the course assignment 1 of parallel system. In this report, I answer all the questions, derive the performance model under some assumptons, show all the results of experiments on my parallel code and make a brief literature review on applying cache oblivious method to stencil problem. I also discuss some deficiencies of my work and give some feedback which may be useful.

[*] We declare that this work: upholds the principles of academic integrity, as defined in the ANU Policy: Code of Practice for Student Academic Integrity.

Contents

1. Overview

Hongbo Zhang, u6170245. All the work are accomplished by myself independantly.

The program is developed using git from beginning, so it is possible to trace all my codes in git. All the products, including the codes, computational results, shell/gnuplot scripts, figures, latex and so on, can be found in git. The source code is well commented and There is no any warnings when compiled with the `-Wall` option.

I also finished task 8. I choose to implement overlapping code for $P > 1$ as an optional task.

The report is organized as following. In next 8 sections, I answer the questions, derive the performance models, show experiments and so on. One section is for one task in the assignment specification. After that, there is a section about the deficiencies of the work, which is followed by a feedback.

2. Task 1 Deadlock Issues

According to experiment results, the code will fails due to deadlock with $M \geq 504$ for $P = 1$, and $M = N$. It will have deadlock since the all the processes are sending messages to their right neighbors and waiting for their returns. If the message is small, the message can be packed into the initial buffer, therefore it will behave as non-blocking. However, for larger sizes it fails. It will transit from a non-blocking send to a blocking send when the message becomes too large.

From above experiment results, we can infer that the size of initial buffer is about `1024*sizeof(int)`. Taking the halo data and some overheads into account, the M should be smaller than 504. However, for general P, Q, N and w, the size of M and N of triggering deadlock should satisfy

$$\max\left(\frac{M}{P}, \frac{N}{Q}\right) + w \approx 512$$

Without deploying non-blocking or buffered sends, we can change the order of sends/receives of processes. Odd processes will firstly send, then receive. Even processes will firstly receive, then send. The code can be found below.

```
if ( rank % 2 == 1 ) {
  MPI_Send(&V(u, 1, N_loc), M_loc, MPI_DOUBLE, rightProc, HALO_TAG,
      comm);
  MPI_Recv(&V(u, 1, 0), M_loc, MPI_DOUBLE, leftProc, HALO_TAG, comm,
       MPI_STATUS_IGNORE);
  MPI_Send(&V(u, 1, 1), M_loc, MPI_DOUBLE, leftProc, HALO_TAG, comm)
      ;
  MPI_Recv(&V(u, 1, N_loc+1), M_loc, MPI_DOUBLE, rightProc, HALO_TAG
      , comm, MPI_STATUS_IGNORE);
} else {
  MPI_Recv(&V(u, 1, 0), M_loc, MPI_DOUBLE, leftProc, HALO_TAG, comm,
       MPI_STATUS_IGNORE);
  MPI_Send(&V(u, 1, N_loc), M_loc, MPI_DOUBLE, rightProc, HALO_TAG,
      comm);
  MPI_Recv(&V(u, 1, N_loc+1), M_loc, MPI_DOUBLE, rightProc, HALO_TAG
      , comm, MPI_STATUS_IGNORE);
  MPI_Send(&V(u, 1, 1), M_loc, MPI_DOUBLE, leftProc, HALO_TAG, comm)
      ;
}
```

3. Task 2 The Effect of Non-Blocking Communication

I add compilation option $BLOCK$ to make a selection on which kind of communication to be used.

```
# ifdef BLOCK
   // blocking communication
# endif
# ifndef BLOCK
  // non-blocking communication
# endif
```

Since at current stage, we restrict $P = 1$, so there are *only* left-right communications. In this case, The communication time t_{comm} and computation time t_{comp} should be

$$t_{\text{comm}} \propto 2 \times (t_s + M_{loc} \cdot t_w) \cdot r \propto M \cdot r$$
$$t_{\text{comp}} \propto N_{loc} \cdot M_{loc} \cdot r \propto \frac{M \cdot N}{Q} \cdot r$$

where r is the number of time steps in simulation, $N_{loc}, M_{loc}, M, N, Q$ have the same meaning as in the assignment specification, and we have use the fact $M_{loc} = M, N_{loc} = N/Q$. Obviously, communication time is independent of Q, since there is no top-bottom communication in this case. Consequently, the parameters to maximize the impact of communication should be:

- increase r.
- use large M.
- large Q to reduce the percentage of time used by computation. Furthermore, large Q will result in large $nproc$, so that there will be large mount of communication between different nodes, rather than between processors in a socket only. This is a hardware factor to increase communication time.
- use small N.

So I choose parameters $nprocs = 32, r = 100, N = 96$ and a varying $M = 2 \times 10^2, 2 \times 10^3, 2 \times 10^4, 2 \times 10^5$. The results can be found in Tab 1.

Table 1: The performance of blocking and non-blocking communication

M	2×10^2	2×10^3	2×10^4	2×10^5
blocking	1.45×10^{-2}s	2.26×10^{-2}s	9.96×10^{-2}s	1.32s
non-blocking	6.66×10^{-2}s	4.34×10^{-2}s	8.48×10^{-2}s	1.20s

From the results in Tab 1, for large M, non-blocking's performance is better than that of blocking. While, for small value of M, blocking communication is better. However, in a real fluid dynamics application, the number of grid usually is very large due to the multiple scale nature of the fluid problem. For example, in a fluid model with convection and diffusion, the number of grid should be larger than the $\frac{\text{simulation scale}}{\text{diffusion scale}}$. Consequently, we will adopt non-blocking communication in the following.

Therefore, I comment out the blocking communication code.

4. Task 3 Performance Modelling and Calibration

4.1 Outline

We will give a performance model under certain assumpton in subsection "Performance Model". Then we will design some experiments and compare the performance model with experiments.

The final results are $t_s = 18.3\mu$s, $t_w = 56.5$ns and $t_f = 8.69$ns.

4.2 Performance Model

We neglect the startup overhead, including reading command line parameters, setting simulation parameters and setting initial condition. In this case, this overall cost of this program can be divided into two terms, namely communication cost and computation cost. All the communications happen in `updateBoundary(w,*u,ldu)`, since there is no additional final collective communications to gather all the flow field to the "root" process. All the computation happens in `updateAdvectField(m,n,*u,ldu)`, since we have neglect the startup overhead.

We assume that $P = 1$, and leave the case $P \neq 1$ to next section. We also assume Q is larger than 1. If not, it will not be a parallel program any more, which is not our interest. Therefore, there are only two communications (left-right communication) per iteration. Since periodic boundary condition (wrap) is applied to the top-bottom direction, so some additional assignment operations will be introduced in each iteration. However, the cost of these operations is negligible comparing to the cost in Lax-Wendroff algorithm, so we will not consider them in the following analysis. We also only consider the Store and Forward (SF) routing mechanism.

The communication cost and computation cost read

$$\begin{aligned} t_{\text{comp}} &= M_{\text{loc}} \cdot N_{\text{loc}} \cdot t_f \\ t_{\text{comm}} &= 2\left[t_s + M_{\text{loc}} \cdot t_w\right] \end{aligned}$$

where t_f is the time per element computation time for the serial advection solver, which contains 23 FLOPS and several integer operations. t_w is the communication cost per *double precision number* (not word! if use word, there should be a multipler). t_s is the communication startup time. t_{comp} and t_{comm} are communication time and computation time for one simulation time step, respectively. $N_{loc} = N/Q$ and $M_{loc} = M/P = M$. In the left-right communications, only M_{loc} double precision numbers are sent and received, so there is no w dependence in t_{comm}.

Consequently, we can write down the performance model

$$\begin{aligned} t_{\text{total}} &= r \cdot (t_{\text{comp}} + t_{\text{comm}}) && (4.1) \\ &= r \cdot \left[2t_s + 2Mt_w + \frac{M \cdot N}{Q} t_f\right] && (4.2) \end{aligned}$$

4.3 Calibration

Before proceeding to next step, I would like to emphasize one point. If one wants to fix the value of t_s, t_w and t_f from experiments by statstics method, he must realise that these three terms are highly correlated. Since the only observerable in the experiments is t_{total}, t_s, t_w and t_f are dengerated.

From Eq 4.2, we can see that the prefactor of t_s is a constant, the prefactor of t_w is M and the prefactor of t_f is MN/Q. So we can run two sets of experiments to determine t_s, t_w and t_f.

The first set of experiments we vary Q only, and keep r, M and N fixed. And we can plot a $t_{\text{total}} \sim \frac{1}{Q}$ figure. We fit this plot by a line. The slope of this line is $rMNt_f$. So we can obtain t_f. The setup is that $r = 100$, $M = N = 1000$ and $Q = 2, 4, 8, 16, 32, 64$. Since $P = 1$, Q is the just *nproc*. The results of the experiments is listed in the Tab 2 and is plotted in Fig 1 as well. The plot shows very good linearity, and the slop is 0.869, therefore $t_f = 0.869/r/N/M = 8.69$ns.

Table 2: Result of first set of experiments to determine t_f. Varying Q and keeping r, M, N fixed.

nproc = Q	2	4	8	16	32	64
t_{total} (s)	4.61×10^{-1}	2.42×10^{-1}	1.30×10^{-1}	6.62×10^{-2}	3.97×10^{-2}	6.26×10^{-2}

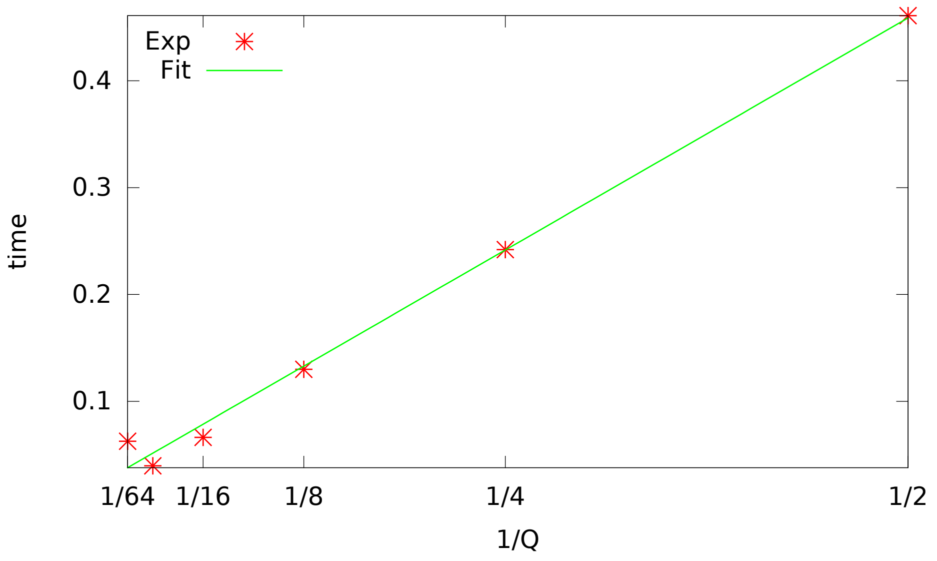

Figure 1: Plot of first set of experiments. The slope is $rMNt_f$, hence $t_f = 8.69$ns.

The second set of experiments we vary M only, and keep r, N and Q fixed. And we can plot a $t_{\text{total}} \sim M$ figure. We also fit this plot by a line. The intercept of this line is $2rt_s$. The slope of this line is $r(2t_w + \frac{N}{Q}t_f)$. From the intercept and slope, we can get t_s and t_w. The setup is that $r = 100$, $N = 1000$, $Q = 32$ and $M = 10^3, 2 \times 10^3, 3 \times 10^3, 4 \times 10^3, 5 \times 10^3$. The results of the experiments is listed in the Tab 3 and is plotted in Fig 2 as well. The plot shows very good linearity, and the intercept is 3.66×10^{-3}, therefore, $t_s = 3.66 \times 10^{-3}/2/100 = 1.83 \times 10^{-5}$s. The slope is 3.82×10^{-5}, therefore $t_w = (3.82 \times 10^{-5}/r - \frac{N}{Q}t_f)/2 = 56.5$ns.

Table 3: Result of second set of experiments to determine t_s and t_w. Varying M and keeping r, N, Q fixed.

M	10^3	2×10^3	3×10^3	4×10^3	5×10^3
t_{total} (s)	4.42×10^{-2}	8.00×10^{-2}	1.14×10^{-1}	1.57×10^{-1}	1.97×10^{-1}

4.4 Summary

So $t_s = 18.3\mu$s, $t_w = 56.5$ns and $t_f = 8.69$ns.

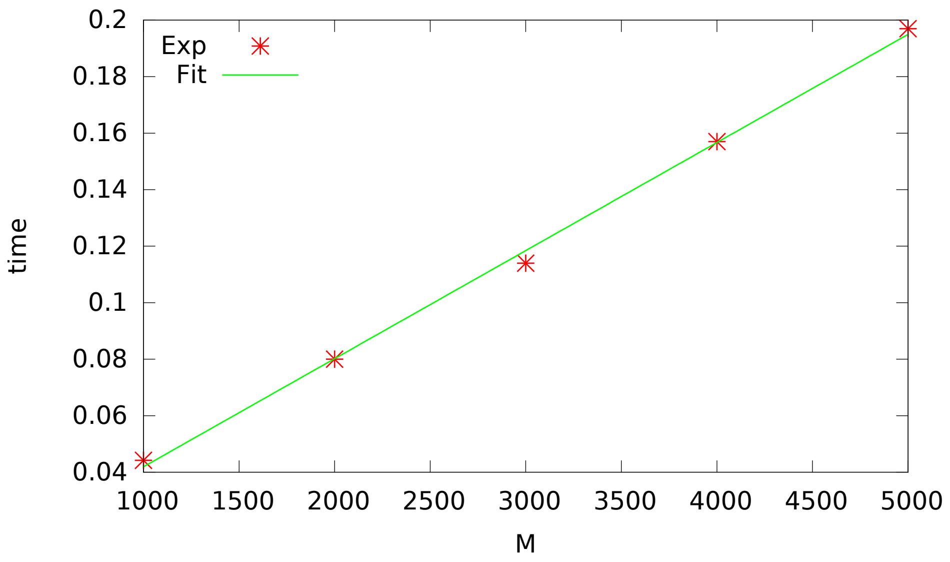

Figure 2: Plot of second set of experiments. The slope is $r(2t_w + \frac{N}{Q}t_f)$, and intercept is $2rt_s$, hence $t_s = 18.3\mu\text{s}$, and $t_w = 56.5\text{ns}$.

5. Task 4 The Effect of 2D Process Grids

5.1 Implementation

The case of $P > 1$ is similar to that of $P = 1$ except the top-bottom communications. Since the buffers in the same row is not continuous in memory, it will lead to two different approaches to do top-bottom communication. One of them will have a better performance than the other.

Bad approach send every element in a row one by one to its neighbors. Therefore, the cost is $(N_loc+2w)\cdot(t_s+t_w)$. This approach is expensive since it will run startup algorithm $N_{loc} + 2w$ times.

Good approach firstly collect all the element in a row to a continuous buffer, then send that buffer in one time to its neighbors. The cost is $t_s + (N_loc + 2w)\cdot t_w$.

In my implementation, I used the "good approach".

5.2 Performance Model

The assumptions made in this model is similar to that in previous section, except that we loose the assumption $P = 1$. We assume that $P > 1$ in this section. We also assume Q is larger than 1. Furthermore, since we use the "good approach", in each top-bottom communication there will be some overhead due to collecting the elements to a buffer. However, this overhead is negligible.

The communication cost and computation cost read

$$\begin{aligned} t_{\text{comp}} &= M_{\text{loc}}\cdot N_{\text{loc}}\cdot t_f \\ t_{\text{comm}} &= 2\left[2t_s + \left(N_{\text{loc}} + M_{\text{loc}} + 2w\right)\cdot t_w\right] \end{aligned}$$

$N_{loc} = N/Q$ and $M_{loc} = M/P$. In the left-right communications, only M_{loc} double is sent and received, however, in the top-bottom communications, $N_{loc} + 2w$ is communicated. There are four communications per iteration.

Consequently, we can re-write down the performance model

$$\begin{aligned} t_{\text{total}} &= r\cdot\left(t_{\text{comp}} + t_{\text{comm}}\right) && (5.1)\\ &= r\cdot\left[4t_s + 2\left(\frac{N}{Q} + \frac{M}{P} + 2w\right)t_w + \frac{M\cdot N}{P\cdot Q}t_f\right] && (5.2) \end{aligned}$$

5.3 Aspect Ratio Effect

We will analysis Eq 5.2 in this subsection. Notice that $PQ = n$ (n is the number of processes) is fixed, $M = N$, the following important relation holds

$$\frac{1}{P} + \frac{1}{Q} = \frac{1}{P} + \frac{P}{n} \geq 2\sqrt{\frac{1}{P}\cdot\frac{P}{n}} = \frac{2}{\sqrt{n}}$$

the equality holds when $P = Q = \sqrt{n}$.

Taking advantage of above relation, the total cost reads

$$t_{\text{total}} \geq r\cdot\left[4t_s + 2\left(\frac{2M}{\sqrt{P\cdot Q}} + 2w\right)t_w + \frac{M^2}{P\cdot Q}t_f\right] \quad (5.3)$$

Consequently, when $P = Q$, the equility holds and the performance is best. As the difference between P and Q increases, the total cost also increases.

5.4 Experiments

According to the analysis in above subsection, we set up the experiments as following. The number of processes is chosen to be $nproc = 24$, and we do a set of experiments with varying $P = 2, 3, 4, 6, 8, 12$. Other parameters are $M = N = 1000, r = 100$. The results of the experiments are listed in Tab 4. The results fit about argument. We can see when $P = 6, Q = 4$, the cost is lowest among all the cases. It is interesting to see that the case of $P = 6, Q = 4$ is different from $P = 4, Q = 6$. The reason of this phenomena is the top-bottom communications have overhead, such as collecting elements in a row to a temporary buffers.

Table 4: Result of experiments to show the aspect ratio effects. Varying P

P	2	3	4	6	8	12
t_{total} (s)	7.41×10^{-2}	1.00×10^{-1}	7.66×10^{-2}	6.27×10^{-2}	7.74×10^{-2}	6.52×10^{-2}

5.5 If t_f is Smaller

In this model, if t_f is decreased, the overall trends will not change. Since the prefactor of t_f term is a constant for a given number of processes and given M.

6. Task 5 Overlapping Communication with Computation

6.1 Implementation

The main part of function `parAdvectOverlap(reps, *u, ldu)` is divided into three steps.

First steps Update the halo by non-blocking sends/receives, and `MPI_Wait` will not be used in this part.

Second steps Update the bulk (inner) field. Bulk is the part of local field which doesn't adjacent to halo. To be more precisely, its index is (r, c) where $r \in [w+1, N_{loc}+w-2]$ and $c \in [w+1, M_{loc}+w-2]$. Since there is no dependancy on halo field in updating the bulk field. Therefore, this part of computation can overlap with non-blocking communication.

Third steps Firstly `MPI_Wait` is called in this part. Then making use of the updated halo values, we can update the boundary field in turn. The boundary is the grids between bulk and halo. To be more precisely, its index ranges are $(w, w) \sim (w, N_{loc}+w-1)$, $(M_{loc+w-1}, w) \sim (M_{loc+w-1}, N_{loc+w-1})$, $(w+1, w) \sim (M_{loc+w-1}, w)$, $(w+1, N_{loc+w-1}) \sim (M_{loc+w-2}, N_{loc+w-1})$. *Attetion*: in order to update the boundary field, four temporary buffer `utmpr`, `utmpl`, `utmpt` and `utmpb` are required! They are used to update right, left, top and bottom boundary, respectively.

6.2 Performance

Using non-blocking sends and receives to update halo field firstly, the program will continue to run the subsequent codes (updating bulk field) which don't depend on halo field values. Therefore, the communication and computation are overlapping.

The effets on performance model The performance model consist three terms, t_f term, t_w term and t_s term. The last two terms are communication terms. The overlapping approach will decrease t_w term significantly by overlapping the communication with computation. However, it will not affect t_s term too much. Furthermore, if the computation time of the concerned problem is small, the performance will not be improved too much, since the performance is improved by overlapping between computation and communication. We will use it as a guildline to design experiments.

Experiments Following the guildline given in previous paragraph, we will vary N and keep M, r and Q fixed. The reason of doing so is in the case of $P = 1$, the communication cost only depends on M and computation cost depends on both M and N. Fixing M and varying N is equivalent to keep communication cost fixed and change the computation cost. As we argued in previous paragraph, the performance of overlapping method depends on the mount of overlapping between communication and computation. Therefore, we increase N to increase the overlapping part of computation. The experiment settings is $M = 1000$, $Q = 32$, $P = 1$, $r = 100$ and $N \in \{10^2, 10^3, 10^4, 10^5\}$ We run these experiments with and without overlapping method. The results are shown in Tab 5. Generally, the performance of overlapping is better than that without overlapping, except for small N. For small N overlapping is a little worse than non-overlapping, since in my overlapping algorithm, there is some overhead, such as copy values to a temporary field buffers which are used to calculate boundaries. However, for large N, the performance of overlapping is better, and the larger value of N, the better performance.

Table 5: Comparison between overlapping and non-overlapping. Varing N

N	10^2	10^3	10^4	10^5
without overlap	1.08×10^{-2}s	4.13×10^{-2}s	4.24×10^{-1}s	4.19s
with overlap	1.40×10^{-2}s	4.37×10^{-2}s	4.12×10^{-1}s	4.16s

7. Task 6 Wide Halo Transfers

7.1 Implementation

The implementation of wide halo method is quite straightforward. Update the halo with w width every w simulation time steps. For each time step in a cycle of total w, the part of field to be upated decreases by two. Therefore, at time step 0, the size of that part is $(M_{loc} + 2 * w - 2)(N_{loc} + 2 * w - 2)$, at the step i, the size is $(M_{loc} + 2w - 2 - 2i)(N_{loc} + 2 * w - 2 - 2i)$. Again, similar to Task 4, in each communication, we collect w rows/columns into a temporary buffer firstly and then send/receive in one time. This will reduce the t_s term in communication cost.

7.2 Performance Model

There is a trade off in this approach. The startup time t_s part of communication time is decreased by a factor of w. However, the computation cost increases, since one has to do more computation for additional part of field to be updated.

In derivation of the following performance model, we assume that simulation time cycle r is much larger than w, since if it is not, the wide halo method will be worse than the method without wide halo, which is not our interest. Furthermore, for simplicity, we assume that $r\%w = 0$.

The total time cost is

$$t_{\text{total}} = \frac{r}{w} t_{\text{cycle}}$$

where $t_{\text{cycle}} = t_{\text{comp}} + t_{\text{comm}}$ is the cost in one cycle (w) of time steps.

For the communication cost per cycle

$$t_{\text{comm}} = 2\left[2t_s + w \cdot (N_{loc} + M_{loc} + 2w)\, t_w\right]$$

For the computation cost per cycle

$$\begin{aligned} t_{\text{comm}} &= \sum_{0}^{w-1} (M_{loc} + 2w - 2 - 2i)(N_{loc} + 2 * w - 2 - 2i) t_f \\ &= w M_{loc} N_{loc} t_f + w(w-1)(M_{loc} + N_{loc}) t_f + \frac{2}{3} w(w-1)(2w-1) t_f \end{aligned}$$

where the summation over i reflects the part of field to be updated changes in each time step.

Collecing all the terms, the final total cost is

$$t_{\text{total}} = r\left[\frac{4}{w} t_s + 2\left(M_{loc} + N_{loc} + 2w\right) t_w + M_{loc} N_{loc} t_f + (w-1)(M_{loc} + N_{loc}) t_f + \frac{2}{3}(w-1)(2w-1) t_f\right] \tag{7.1}$$

where $N_{loc} = N/Q$ and $M_{loc} = M/Q$.

Comparing wide halo Eq 7.1 with non-wide halo Eq 5.2, there are two differences. The first one is the startup term in communication cost decreases by a factor of w, since t_s is a relative large term, so this reduction is not negligible. The second one is the additional computation cost, which is proportional to w or w^2, if w is not small and comparable to N_loc or M_loc, this term is comparable to the non-wide halo computation term $M_{loc} N_{loc} t_f$. Therefore, one would be better not to choose a large w.

Consequently, in the case where the communication cost is not negligible, the wide halo will have a better performance due to the reduction of startup communication t_s term.

7.3 Experiments

According to the argument in previous subsection, we should choose the case in which communication cost is large to do experiments. Hence we choose $P \neq Q$ and $M \neq N$ from the aspect ratio argument in Task 4.

In order to investigate the effect of wide halo method, we choose $P = 4$, $Q = 8$, $N = 1 \times 10^4$, $M = 2 \times 10^4$, $r = 100$, and vary $w = \{1, 5, 10, 500\}$. I choose a large w to show that when w is large the total cost will increase due to the additional computation cost. The results are shown in Tab 6. The case of $w = 1$ is the one without wide halo effect, we can see that when w increases (except the case $w = 500$), the total time decreases, so the wide halo method is effective. Furthermore, when $w = 500$, the total cost increases by almost a factor of 2. This is because when $w = 500$, which is near $N_{loc} = 10^4/8 = 1250$, the additoinal computation cost is comparable to the original one.

Table 6: Wide halo effect. Experiment Set 1, varying w and keep r, P, Q, M, N fixed.

w	1	5	10	500
time	8.65s	8.52s	8.21s	16.7s

7.4 Comparison with Overlapping Method

The wide halo method will reduce the startup communication cost, the t_s term. The overlapping method will also reduce the communication cost, however, it reduce the data transfering term, the t_w term, by overlapping it with computation.

Consequently, both methods will reduce the communication cost, but from different aspects.

8. Task 7 Literature Rewiew on Cache Oblivious

In following paragraphs, we will first introduce cache oblivious algorithm, and then discuss its application and impacts in stencil computation.

Reduction of cache miss is one of important aspect of algorithm optimization. It is also true in parallel computation, which highly emphasizes on the performance of algorithm. Many programmers is aware of this fact and implement their algorithm with the consideration of cache miss. However, many of such algorithms will depend on the parameters of cache, such as cache line size and the number of cache line. These cache parameters are actually relevant to hardware, which is out of control of programmer. Consequently, one program optimized with some specific cache parameter may have very low performance on a machine with another set of cache parameters. Furthermore, in modern computers, there are more than one level of caches with difference cache parameters, which make the above issues more serious. To solve this problem, Frigo and his collabrators [1] proposed the idea of cache oblivious algorithm for tall cache in parallel programming. Their cache oblivious algorithm doesn't know the cache parameters, but uses caches as effeciently as those algorithms who knows cache parameter foreahead.

In the master thesis of Prokop [4] applied the cache oblivious method to stencil problem for the first time. But his algorithm can only work for one dimension space problem with square spacetime region. In 2005, Frigo proposed an algorithm which can work for arbitrary stencil and arbitrary spacetime dimension [2]. In this paper, the spacetime concerned in the problem is divided into *trapezoidal regions*. If the flow velocity is zero, the the trapezodial regions become rectangular regions. Once the the field value u at the bottom edge(or surface or cube, depending on the space dimension) of the trapezodial region is known, the whole u values in the trapezodial region can be obtained by applying the approciate fluid equations. These trapezodial regions can be divided into smaller trapezodial region with nearly equal size. This dividing process can be done iteratively until each trapezodial region is small enough to entirely fit in cache. Therefore, in this method, the field values are not calculated in the order of their spatial index any longer, but in the order of trapezodial region index. Frigo proofs this method is cache efficient [2].

Making use of this method, cache miss is reduced. For example, Frigo [3] applied this method to heat diffusion problem. For two dimension problem, the cache miss will be reduced by a factor of 5 on average, and for one dimension heat diffusion problem, the cache miss will be reduced to 1% [3], Therefore, the cache oblivious method improves the performance of parallel stencil problem by reducing the cache miss.

9. Task 8 Optional

I choose to implement overlapping code for $P > 1$ as an optional task.

The idea of the solutioin to this problem is similar to the case for $P = 1$ in task 5. But in this case, the main part of `parAdvectExtra` contains five parts.

First part Non-blocking communication the left and right halo. Don't do `MPI_Wait` in this step.

Second part Update the field values in the bulk. The definition of bulk is the same as that for $P = 1$ in task 5.

Third part `MPI_Wait` for left and right halo communication here.

Fourth part Top and bottom communications. I implement in non-blocking communication but `MPI_Wait` immediately. Therefore, it is essential a blocking communication.

Fifthe part Update the field value on the boundary. The definition of boundary is the same as that for $P = 1$ in task 5. Also use four temporary buffers.

I run two experiments with parameters $P = 4$, $Q = 2$, $M = N = 1000$, $r = 100$ with and without `-x` option. The one with overlapping method uses time 0.152s, and the one without overlapping method uses 0.154s.

10. Deficiencies

I should make the code more robust. For example, w should be smaller than N_{loc} and M_{loc} mathematically. I should add some code to tell the user if they input the wrong parameters.

In task 4 ratio aspect effect, from model, the larger difference in M and N, the more total cost. However, my results has some fluctuations.

In task 5 overlapping, I use temporary buffer *utmpt*, *utmpb*, *utmpl* and *utmpr* to update the field values on boundaries so that we can decouple the bulk calculation with the field values on the halo. The trade off of this approach is that we have to copy values to this four buffers, which generates some overhead. So in some situation, the overlapping method will be a little worse than the non-overlapping one, although their different is very small.

In task 8 overlapping for $P > 1$, there are 4 communications in each iteration. But only first two of them is overlapped with computation. The last two communications don't overlap with computation. This is a deficiecy.

11. Feedback

In the experiments results, there is always some "fluctuations", I can't understand what's the reason of these fluctuations, so I think it is better to know some details on message communication, such as what's factor will affect t_s and so on.

Acknowledgments

Thanks Peter for nice lectures, Sara for her lab tutorial and Raijin for providing the computing resource.

References

[1] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 285–297, 1999.

[2] Matteo Frigo and Volker Strumpen. Cache oblivious stencil computations. In *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, pages 361–366, New York, NY, USA, 2005. ACM.

[3] Matteo Frigo and Volker Strumpen. The memory behavior of cache oblivious stencil computations. *J. Supercomput.*, 39(2):93–112, February 2007.

[4] Harald Prokop. Cache-oblivious algorithms. Master's thesis.