COMP4300 Assignment 1

Jackson Kilrain-Mottram (u6940136)

April 2023

1 Question 1

The parallel advection solver deadlocks for values of N greater than or equal to 2^{15} (32768). The `MPI_Send` function has different behaviour depending on the total count of elements to send. In this case 2^15 is the upper limit to internally buffered, non-blocking behaviour. Beyond this threshold, the data is not copied into an internal buffer and as such must wait for the send to complete (a matching `MPI_Recv` is invoked) before returning control to the application. Given that all of the processes are using the same model of sending before waiting to receive, they will all send in a blocking manner and none will initiate a receive as it hasn't finished sending yet. To fix this, we can either start with a send or a receive depending on the rank modulo 2. This means that half of the processes will start sending and the other half will start by receiving, ensuring no deadlocks. MPI provides a neat abstraction in the form of `MPI_Sendrecv` to handle this per-rank organisation still using `MPI_Send` and `MPI_Recv` under the hood.

2 Question 2

In order to determine the impact of the non-blocking communication compared to the blocking communication, we need to devise a parameter set that maximises communication overhead. Given the nature of the solver is implemented via domain decomposition, the optimal method for increasing communication is to maximise the number of local domains.

That is to say, minimising `M_loc` and `N_loc` such that $P \times Q$ approaches $M \times N$. In the maximal case, there is a single process associated with each element of the matrix $M \times N$, with halo width one. This optimises for communication, where every element performs 8 individual exchanges (8 sends, 8 receives) along each of the four sides and the four corners. This results in the following equation modelling the total communications per timestep:

$$C = \left(4 + \frac{2M}{P} + \frac{2N}{Q}\right) \cdot P \cdot Q$$

We can look at solving for P and Q for some M, N, by maximising the value of C with respect to these variables. More formally:

$$\forall M, N \operatorname*{argmax}_{P,Q} \left\{ \left(4 + \frac{2M}{P} + \frac{2N}{Q}\right) PQ \;\middle|\; P, Q, M, N \in \mathbb{Z}^+ \land P \leq M \land Q \leq N \land (PQ) \bmod (MN) = 0 \right\}$$
$$\implies P = M, Q = N$$

Inspecting the function, we can see that it is monotonic due to the restriction of the modulo between PQ and MN along with the multiplicity requirement of the $\frac{2M}{P}$ and $\frac{2N}{Q}$ constraints in the equation.

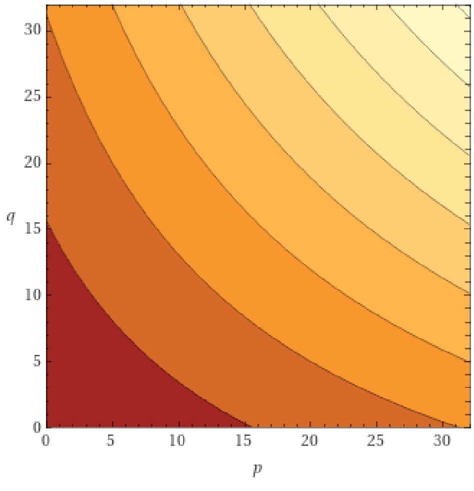


Figure 1: Contour plot of C with $M = N = 32$ without the modulo constraint

From figure 1 we can see that contour of this function strictly increases monotonically along the diagonal from the minimum values of $P = 1, Q = 1$ to the maximum values $P = M, Q = N$. All solutions of the argmax will lie at the maximum of this function, which is indicated to be the maximum of the diagonal. This shows that the optimal values for P, Q are M, N respectively for maximising communication. (This is not rigorous, more of a sketch as the focus is not on this statement of maximisation).

For a 1D process grid, we can maximise communication by choosing $P = M$ such that every process affords two top/bottom border exchanges as an entire row is done in a single communication. Additionally, two left/right border and four corner exchanges are done as column exchanges on the local advection field. This yields maximum communication while also minimising the flops per timestep, as each process is computing the minimum field evolution, thereby further skewing the overall performance towards the communication overhead.

We can't make the value of P, Q, M, N arbitrarily large, due to the limiting factor that process count scales exponentially with matrix size. We evaluate the following values for the parameters:

- $Q = 1, P = M = N = 192$, advection field size $36,864$, comms 384
- $Q = 1, P = M = N = 256$, advection field size $65,536$, comms 512
- $Q = 1, P = M = N = 512$, advection field size $262,144$, comms $1,024$

Blocking(GFLOPS)	**Non-Blocking (GFLOPS)**	**Blocking(Time)**	**Non-Blocking(Time)**
8.04e+00 4.19e-02 (PC)	3.71e+01 1.93e-01 (PC)	9.17e-03s	1.99e-03s

Table 1: Results of first configuration $Q = 1, P = M = N = 192$

Blocking(GFLOPS)	**Non-Blocking (GFLOPS)**	**Blocking(Time)**	**Non-Blocking(Time)**
9.06e+01 3.54e-01 (PC)	1.27e+024.97e-01 (PC)	1.45e-03s	1.03e-03s

Table 2: Results of second configuration $Q = 1, P = M = N = 256$

Blocking(GFLOPS)	**Non-Blocking (GFLOPS)**	**Blocking(Time)**	**Non-Blocking(Time)**
2.08e+02 4.06e-01 (PC)	2.13e+02 4.17e-01 (PC)	2.52e-03s	2.46e-03s

Table 3: Results of third configuration $Q = 1, P = M = N = 512$

From these results, we can gather that the non-blocking implementation is definitely better. It outperforms the blocking implementation in both time and GFLOPS. Interestingly, with higher field sizes and processor counts, the difference grows smaller, it's hard to say conclusively from these results whether this is absolute or an artefact of the limited scope here. In any case, the non-blocking variation can be seen to outperform the blocking.

3 Question 3

3.1 1D Model Derivation

In the 1D model, each process performs two communications of the same form, which can be modelled as

$$\frac{M}{P} \cdot 4t_w + t_s$$

Where the $\frac{M}{P}$ value denotes the row dimension of the local advection grid, each element of which contains doubles. Each double is contained in 4 words, so the $4t_w$ factor account for this. Lastly, the t_s supplies the startup overhead for communicating the row. Given that each process performs this twice, we multiply this expression by two:

$$2 \cdot \left(\frac{4Mt_w}{p} + t_w\right)$$

We need to include the cost of computing the advection field after the communication occurs. Given that the local advection field dimensions are factors of the process grid, we can express the local grid size as the product of the dimension ratios

$$\frac{M}{P} \cdot \frac{N}{Q}$$

Given that each element of the advection grid employs as cost of t_f (as given by the question statement), we multiply the ratio expression by this to result in the total computation cost for the local advection grid. Combining this with the expression for the communications and accounting for r iterations, the final model is

$$r \cdot \left(2 \cdot \left(\frac{4Mt_w}{P} + t_s\right) + \frac{MNt_f}{PQ}\right)$$

3.2 Measuring Coefficient Values

To gather experimental values for the time valued variables (t_s, t_w, t_f), we can use timers and numerical methods based on the timing requirement.

3.2.1 Evaluating t_f

The most straightforward to capture is t_f which can be done by wrapping the double loop within the `updateAdvectField()` function invocation with `MPI_Wtime()` before and after, then computing the average of all iterations averaged over invocations. For this we use, $48, 96$ and 192 cores in distinct runs over $r = 100$ iterations, with field dimensions $M = N = 1000$.

Core Count	Average t_f (Seconds)
48	$1.6338549417164163e - 09$
96	$1.8313747096703432e - 09$
192	$1.9773684770638774e - 09$

Table 4: Average measured times for a single field element update.

Examining the results, we can see that on average the duration for a local field element update is $\approx 1.8141993761502122e-09$ seconds or $\approx 1.8141993761502122062$ nanoseconds.

3.2.2 Evaluating t_s

If we consider the composite form of a communication cost as an expression, it is comprised of the startup time t_s and the cost of sending n words of data as nt_w:

$$t_{\text{comm}} = t_s + nt_w$$

So, we can imagine, that if we send no data for a given communication, then the cost is simply

$$t_{\text{comm}} = t_s$$

This is precisely how we can measure the value of t_s. Implementation wise, this can be done by a ping-pong message exchange between two processes with a message of size zero. In order to determine a value accurately, we need to account of the overhead of the timers themselves. This can be done by taking the average of a timer that is started and immediately stopped several times then dividing the value by two. We reduce the final t_s value measured by this amount to achieve an accurate measurement. (See `startup.c` for more details on the implementation).

Buffer size	Measurement (Seconds)
100	$2.7e - 04$
1000	$3.35e - 04$
10000	$2.9e - 04$
100000	$2.96e - 04$
1000000	$2.46e - 04$
10000000	$3.34e - 04$
Average	$2.951666666667e - 04$
Average (One Way)	$1.47583e - 05$

Table 5: Measurements of t_s without timer overheads with given buffer sizes. Note that the `MPI_Send` and `MPI_Recv` calls specify a size of 0 explicitly.

Computing the average of the averages, we see that t_s is approximately $1.47583e - 05$ seconds, or ≈ 14.7583 microseconds.

3.2.3 Evaluating t_w

In the previous section, we characterised the value of t_s through experimental analysis. That is an important stepping stone to evaluating t_w, as we need to eliminate the overhead from our measurements. Given that we have already performed the necessary measurements to determine the ping-pong time for various word sizes in previous lab tasks, we can use the values to calculate the cost of t_w.

In prior, we evaluated several transfers of varying word counts from 1 to 4194304. Each of which gave slightly varying values for duration for ping-pong to complete on average. In the below table, the raw data is set out, including only the ones with large enough buffer sizes that the measurements can be used accurately.

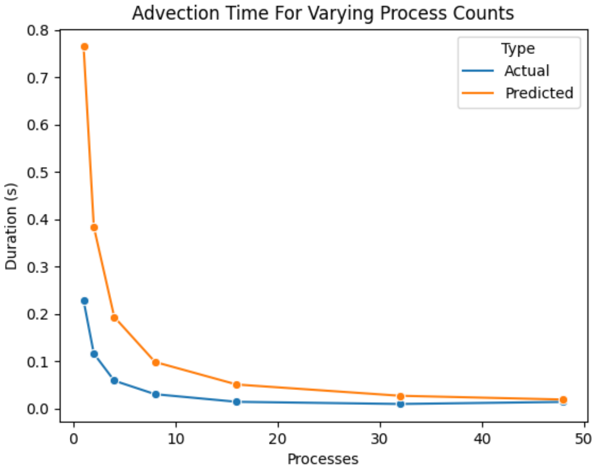
Word Count	Duration (Seconds)	t_w (Seconds)
16384	$1.51e-05$	$2.0855712890624925e-11$
65536	$5.56e-05$	$6.231948852539063e-10$
262144	$2.06e-04$	$7.295291900634766e-10$
1048576	$8.05e-04$	$7.536332130432129e-10$
4194304	$4.66e-03$	$1.1075119256973267e-09$

Table 6: Raw measurements of ping-pong for various sizes payloads (words) [Sourced from course site LAB1 notes]

The values of t_w computed in the table above is done according to the following formula $\frac{\text{duration}-t_s}{\text{word count}}$. We can take an average of all of these values to find an approximation for the value of $t_w \approx 6.469449853897094e-10$. Note that t_s substantially (several orders of magnitude) larger than the value of t_w. This gives a good indication that smaller message sizes will incur a heavier overhead as the actual transfer time may well be less than that of the startup time. This would suggest that to effectively armortise the startup time over communication, one should choose to use larger messages that ideally incur the same cost for transfer in total as that of the startup time.

3.3 Strong Scaling Analysis

From the output of `lscpu` on Gadi, we can see that the L3 cache size of an Intel Xeon 8274 is 36608KiB (binary units, not SI). As such, the equivalent in bytes is 37486592, which implies a maximum of 4685824 doubles (8 bytes per double) stored contiguously, can fit in the L3 cache. Therefore our maximum field dimensions, that are closest to being square are 2048×2288. In the interest of making this easier to handle, we will opt for a purely square variant of a slightly smaller set of dimensions 2048×2048, which fills $\approx 89.51\%$ of the L3 cache. Given that this is a power of two, it also allows for analysis to be done on an exponential basis, given a more precise indication of how it scales.



Processes	Predicted (s)	Actual (s)
1	$7.649420e-01$	$2.28e-01$
2	$3.839468e-01$	$1.16e-01$
4	$1.934492e-01$	$5.87e-02$
8	$9.820045e-02$	$3.01e-02$
16	$5.057606e-02$	$1.40e-02$
32	$2.676386e-02$	$9.42e-03$
48	$1.882646e-02$	$1.38e-02$

Figure 2: Comparison of advection solution time against process counts for the 1D advection problem with non-blocking communication. Upper limit of 48 here is based on a single Gadi node process capacity.

We can see that in figure 2, the advection time exponentially decreases in accordance with process count. Intuitively, this is clear as the overall problem space is divided up into smaller and more numerous parallel components. That being said, it appears to asymptote, in relative decrease in solution duration, for $p \geq 32$.

Note the discrepancy with 48 processes, increasing in advection time in comparison to 32. This result was consistent, thought did fluctuate with almost identical performance to 32 processes or slightly worse. We could conclude that the parallelisation of the problem reaches an upper limit around the $32-48$ process mark for the particular problem size of 2048×2048. The local advection field size at this scale is roughly $42.\tilde{6} \times 2048$, which suggests that the communication of

43 elements in the row for top and bottom is relatively matched by the processing time of assigning 2048 elements from the left column to the right and vice-versa.

Examining the predicted duration against the actual, we can see that the model does indeed follow the same behaviour in terms of the scaling relative to the process count. However, in actuality, it does not accurately model the results seen for smaller process counts. Logically, this is accountable insofar as it is a narrowed view of the core elements of computation and communication. In addition to this, the behaviour of the L1-L3 caches is likely to be less performance with the 48 process count execution, given that little space is left to manage other variables of the program that are regularly referenced. For instance the segments of the local advection field for communication and computation could partially evict some parts of the advection field array.

There is potential for memory hierarchy instability to become a component in the fluctuation in the expected results. Another key component of the view on memory is to consider that for larger local advection fields, the organisation or row/bank accesses in RAM will be less optimal, especially for larger contigous arrays. This can cause slower access periods due to the need to constantly switch between banks and rows as the locality depends heavily on the sizes and RAM organisation. A degree of latency uncertainty can be introduced here.

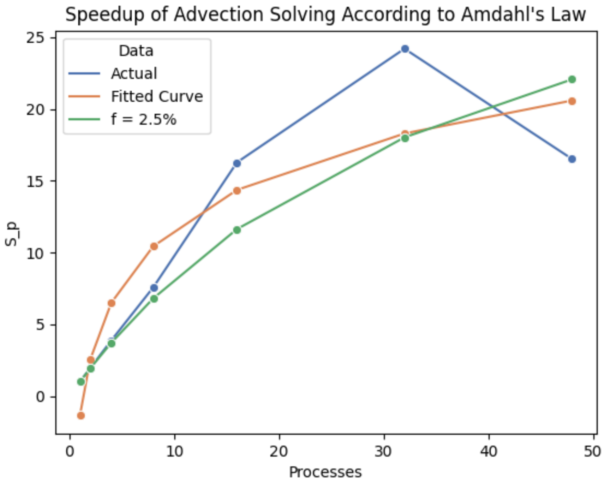


Figure 3: Speedup of Advection relative to single process solution with fitted logarithmic curve and plotted approximate sequential portion curve.

Calculating the relative speedup ($S_p = \frac{t_{\text{seq}}}{t_{\text{par}}}$) of the parallel variant of the advection solver over the single process variant, we can see that it has relative high speedup. This implies that the parllelisation of this problem scales well in a strong-scaling context of a fixed problem size. However, it is not immune to the issues of limiting sequential portions. We can see that the fitted logarithmic curve shows that it has exponential decay away from the apparent linearity seen for process count $1 - 16$. This points to a limiting factoring being achieved for 32 and 48 processes. Noting the plot of the curve of f relative to the problem space (in its closest satisfying form), we can see that there is $\approx 2.5\%$ of the problem space being sequential, as the limiting factor.

4 Question 4

4.1 2D Model Derivation

In order to account for the extra dimension of communication in the 2D model, we need to express the communication of the columns and corners in this model. Notice that the computation cost does not change, as it is not bound by any communication factors.

To remodel the equation of communication, we consider the extended case of replicated communications about the diagonal of the local advection field. That is to say, that we have a single column and a single row exchange with two corners.

In our existing expression, we can include a similar expression for the column cost as have done for the row. Here we exchange the dimension specific terms and include it as follows

$$2 \cdot \left(\frac{4Mt_w}{P} + \frac{4Nt_w}{Q} + t_s \right)$$

For each corner, we need to send a single double which is modelled as the cost of four words. Within the same expression, we include this as a separate term

$$2 \cdot \left(\frac{4Mt_w}{P} + \frac{4Nt_w}{Q} + 8t_w + t_s \right)$$

Note that we now have a total of four distinct communications occurring, so we account for this by scaling the t_s factor by four to model each communication startup overhead cost individually.

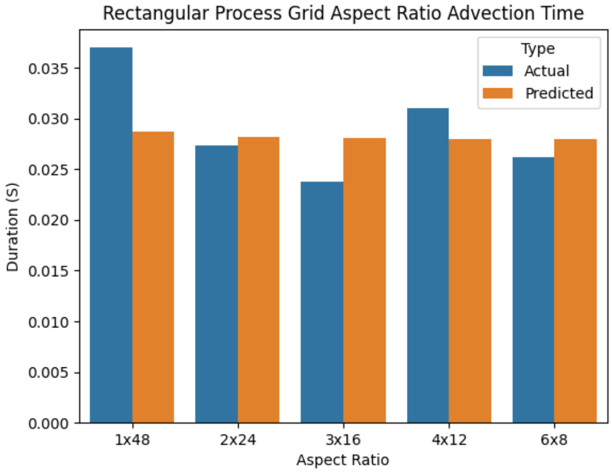
$$2 \cdot \left(\frac{4Mt_w}{P} + \frac{4Nt_w}{Q} + 8t_w + 4t_s \right)$$

Using this new expression for communication, we can replace the original in our model, with some factoring of terms to achieve the following model.

$$r \cdot \left(8 \cdot \left(\frac{Mt_w}{P} + \frac{Nt_w}{Q} + 2t_w + t_s \right) + \frac{MNt_f}{PQ} \right)$$

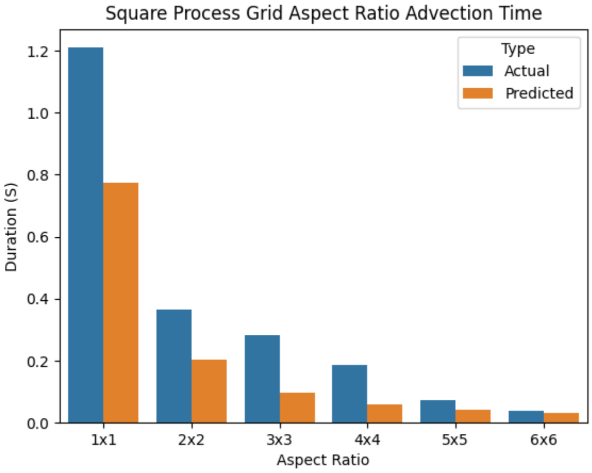
4.2 Aspect Ratio Analysis

Analysing the predicted durations for the advection simulation (figures 4 & **??**), we can see that the most optimal aspect ratio predicted was 3×16 at $2.807996e-02$ seconds. Interestingly, this matches the experimental results, with a slightly lower actual duration of $2.38e-02$. Overall the model scales well according to the experimental results, however the error in smaller process grids is clear. This is to be expected as the model can be considered to have taken the problem at it's most optimal solution (for the given context it was derived in). As such discrepancy is to be expected when the sequential portion of the performance outweighs the parallel portion (in a notion of strong scaling).



Aspect Ratio	Predicted (s)	Actual (s)
1×48	$2.874243e-02$	$3.70e-02$
2×24	$2.823453e-02$	$2.73e-02$
3×16	$2.807996e-02$	$2.38e-02$
4×12	$2.801371e-02$	$3.10e-02$
6×8	$2.796954e-02$	$2.62e-02$

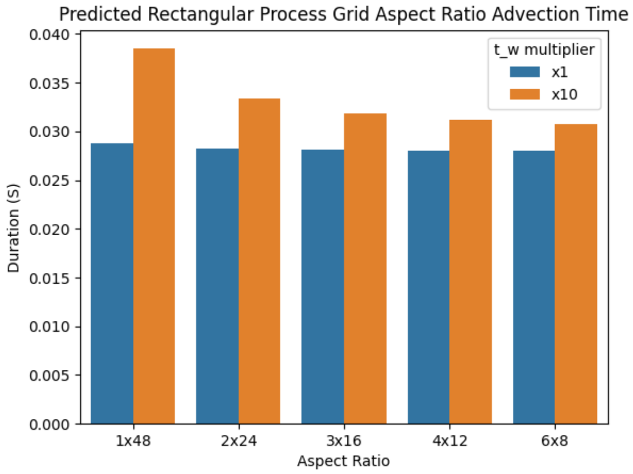
Figure 4: Rectangular aspect ratios.



Aspect Ratio	Predicted (s)	Actual (s)
1×1	$7.748580e-01$	$1.21e+00$
2×2	$2.031002e-01$	$3.65e-01$
3×3	$9.706213e-02$	$2.83e-01$
4×4	$5.989580e-02$	$1.85e-01$
5×5	$4.266887e-02$	$7.15e-02$
6×6	$3.329795e-02$	$3.83e-02$

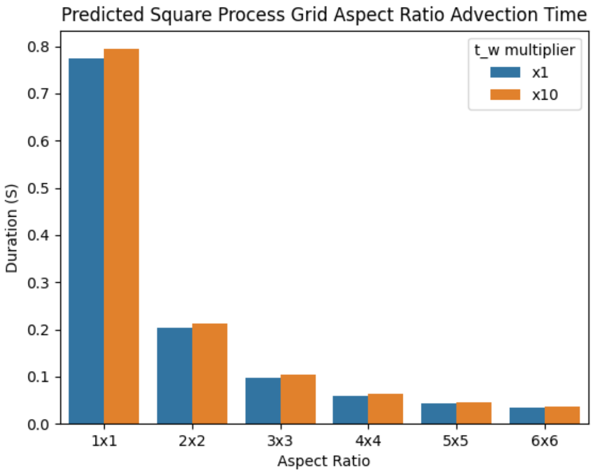
Figure 5: Square aspect ratios.

For single node performance, we can see that rectangular aspect ratios outperform the square aspect ratios. Note that this is with regard to the top two performing square aspect ratios of 5×5 and 6×6. That being said, the rectangular aspect ratios are slightly larger in the total quantity of processes available. This small difference in performance could be easily accounted for here. It should be noted that the rectangular aspect ratios show little difference between them overall, amounting to a standard deviation of $4.598956e-03$, an order of magnitude lower than the results themselves, showing minor significance.



Aspect Ratio	$t_w \times 1$ **(s)**	$t_w \times 10$ **(s)**
1×48	$2.874243e-02$	$3.849008e-02$
2×24	$2.823453e-02$	$3.341113e-02$
3×16	$2.807996e-02$	$3.186536e-02$
4×12	$2.801371e-02$	$3.120289e-02$
6×8	$2.796954e-02$	$3.076124e-02$

Figure 6: Comparison of ×1 and ×10 multipliers for t_w in predicted rectangular aspect ratios.



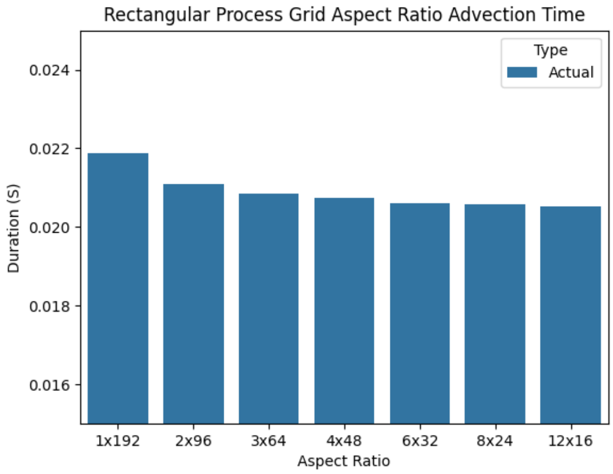
Aspect Ratio	$t_w \times 1$ **(s)**	$t_w \times 10$ **(s)**
1×1	$7.748580e-01$	$7.939465e-01$
2×2	$2.031002e-01$	$2.126491e-01$
3×3	$9.706213e-02$	$1.034312e-01$
4×4	$5.989580e-02$	$6.467491e-02$
5×5	$4.266887e-02$	$4.649402e-02$
6×6	$3.329795e-02$	$3.648713e-02$

Figure 7: Comparison of ×1 and ×10 multipliers for t_w in predicted square aspect ratios.

Figures 10 & 11 show the predicted advection time for rectangular and square process grids with a ×1 and ×10 multiplier on the t_w coefficient. We can see that for the rectangular aspect ratios, as the configuration becomes more distributed in the quantity of communication per-process, the overhead of the additional cost of communication begins to taper off. This would suggest that for the rectangular aspect ratios, we are approaching a bound on the effectiveness of processes in use versus the performance improvement we achieve overall.In the case of the square aspect ratios, we see that overall, it makes little difference to the overall performance at the higher end of process counts. In much the same format as the rectangular, for the same reasoning.

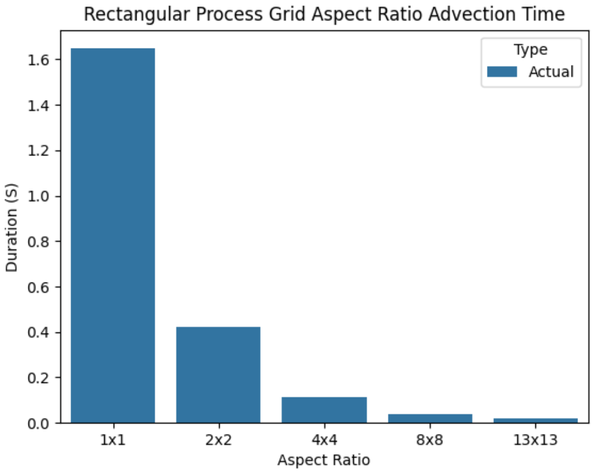
4.3 Dual Socket Analysis

As previously noted, the L3 cache capacity of an Intel Xeon 8274 is 36608KiB (binary units, not SI). As such, the equivalent in bytes is 37486592, which implies a maximum of 4685824 doubles (8 bytes per double) stored contiguously, can fit in the L3 cache. For 2 sockets, this amounts to a capacity of 9371648 doubles. The closest square field dimensions are 2816×3328, while filling the entirity of both caches. We will opt for 3000×3000 which fills $\approx 96.034\%$ of the two L3 caches. This also allows for analysis to be done on an exponential basis, as the scaling factor can be exponential in terms of factors of $2, 5, 10$,etc.



Aspect Ratio	Predicted (s)	Actual (s)
1×192	$2.187249e-02$	–
2×96	$2.110424e-02$	–
3×64	$2.085355e-02$	–
4×48	$2.073225e-02$	–
6×32	$2.061903e-02$	–
8×24	$2.057051e-02$	–
12×16	$2.053817e-02$	–

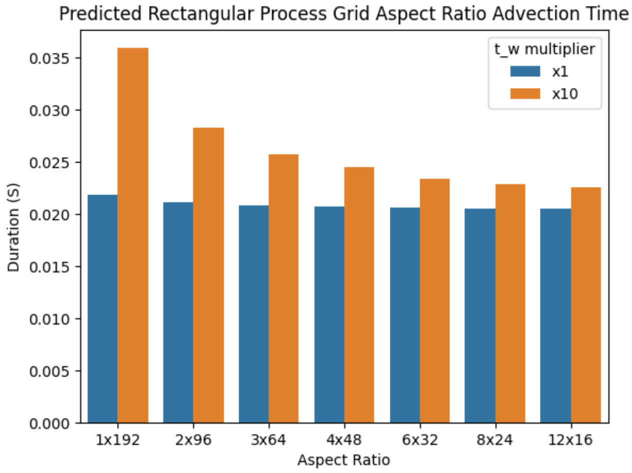
Figure 8: Four node rectangular aspect ratios (lack of measurements due to NCI account limits having been reached).



Aspect Ratio	Predicted (s)	Actual (s)
1×1	$1.647692e+00$	–
2×2	$4.215552e-01$	–
4×4	$1.146327e-01$	–
8×8	$3.770802e-02$	–
13×13	$2.170796e-02$	–

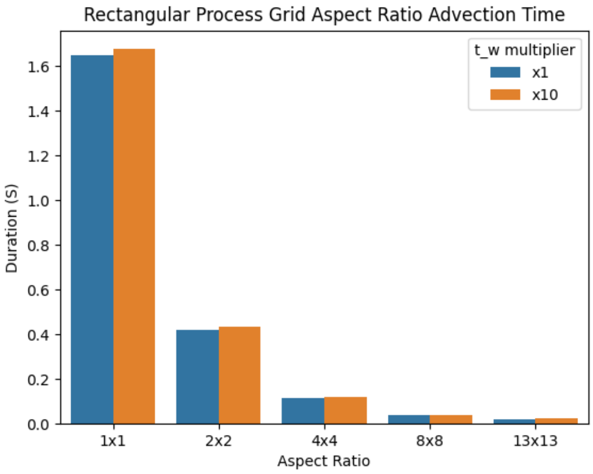
Figure 9: Four node square aspect ratios (lack of measurements due to NCI account limits having been reached).

From the predictions made by the 2D model, we can see that the rectangular aspect ratios are more optimal. Even comparing against the closest square aspect ratios of 8×8 and 13×13. The most optimal is 12 with $2.053817e-02$ seconds. However, that being said, it is clear that the same pattern emerges that there is little differences at the higher end of the aspect ratios. This is the same as what was observed for the smaller, single node cases above.



Aspect Ratio	$t_w \times 1$ (s)	$t_w \times 10$ (s)
1×192	$2.187249e-02$	$3.592860e-02$
2×96	$2.110424e-02$	$2.824613e-02$
3×64	$2.085355e-02$	$2.573921e-02$
4×48	$2.073225e-02$	$2.452619e-02$
6×32	$2.061903e-02$	$2.339404e-02$
8×24	$2.057051e-02$	$2.290883e-02$
12×16	$2.053817e-02$	$2.258536e-02$

Figure 10: Comparison of ×1 and ×10 multipliers for t_w in predicted rectangular aspect ratios.



Aspect Ratio	$t_w \times 1$ **(s)**	$t_w \times 10$ **(s)**
1×1	$1.647692e+00$	$1.675650e+00$
2×2	$4.215552e-01$	$4.355385e-01$
4×4	$1.146327e-01$	$1.216290e-01$
8×8	$3.770802e-02$	$4.121084e-02$
13×13	$2.170796e-02$	$2.386713e-02$

Figure 11: Comparison of ×1 and ×10 multipliers for t_w in predicted square aspect ratios.

Looking at the predictions for a ×10 multiplier on the t_w coefficient, we can see the pattern pattern emerge as seen in the single node variants. Note that, however, the effect is more pronounced. This would suggest, that for larger proccessing grids, relative to larger advection grids, t_w contributes more to the performance characteristics. Logically, this follows, since there are more processes and on average, larger messages being sent. One should be careful with this conclusion however, as the imapct of the computation of the local advection grid has also scaled proportionally, in the same manner. As such, the actual conclusion drawn should be exponential scaling according to both problem size and process availability. This naturally gives rise to the notion that is captured in Gustafson's Law of weak scaling.

5 Question 5

5.1 Discussion of Performance Impact

Overlapping communication and computation (specifically with regards to the implementation here), follows the following sequence of events (Baruffa 2020):

1. Copy data of the ghost cells to send buffers
2. Halo exchange with `MPI_Isend`/`MPI_Irecv` calls
3. Compute (part 1): Update the inner field of the domain
4. `MPI_Waitall`
5. Copy data from receive buffers to the ghost cells
6. Compute (part 2): Update the halo cells
7. Repeat N times

It should be noted that the computation is split into two sections, where the overlapped portion is the inner field of the advection field. In our configuration with a halo width of one, this corresponds to the majority of the field. It is hard to characterise with certainty, the true amount of parallelism achieved with the overlap. As such, we should compute the max of the halo exchange and the inner field computation. For whichever is the largest, we use that quantity as the total cost of the overlapped portion. Let us first examine, the generalised format of the 2D model as it stands currently:

$$r \cdot (t_{\text{comm}} + t_{\text{comp}})$$

As we have understood, the t_{comm} section of this model is no longer a set of communication events. Instead, we need to compute the max of the communication of the halo and the update of the inner field of the advection field. Specifically, we need to exclude the inner halo update from it, this can be done by simply subtracting the total halo width and height from the field dimensions. As such our new t_{comm} is

$$\max\left\{8 \cdot \left(\frac{Mt_w}{P} + \frac{Nt_w}{Q} + 2t_w + t_s\right), \left(\frac{M}{P} - 2\right) \cdot \left(\frac{N}{Q} - 2\right) \cdot t_f\right\}$$

Given that we update the inner field in parallel to the halo exchange, we to re-evaluate the t_{comp} section of the model. Specifically, we need to include an expression for only the inner halo updates.

$$2t_f \cdot \left(\left(\frac{M}{P} - 2\right) + \frac{N}{Q}\right)$$

Combining the new expressions for our components of the overlapped 2D model, we achieve the following

$$r \cdot \left(\max \left\{ 8 \cdot \left(\frac{Mt_w}{P} + \frac{Nt_w}{Q} + 2t_w + t_s \right), \left(\frac{M}{P} - 2 \right) \cdot \left(\frac{N}{Q} - 2 \right) \cdot t_f \right\} + 2t_f \cdot \left(\left(\frac{M}{P} - 2 \right) + \frac{N}{Q} \right) \right)$$

5.2 Overlapped Comparison

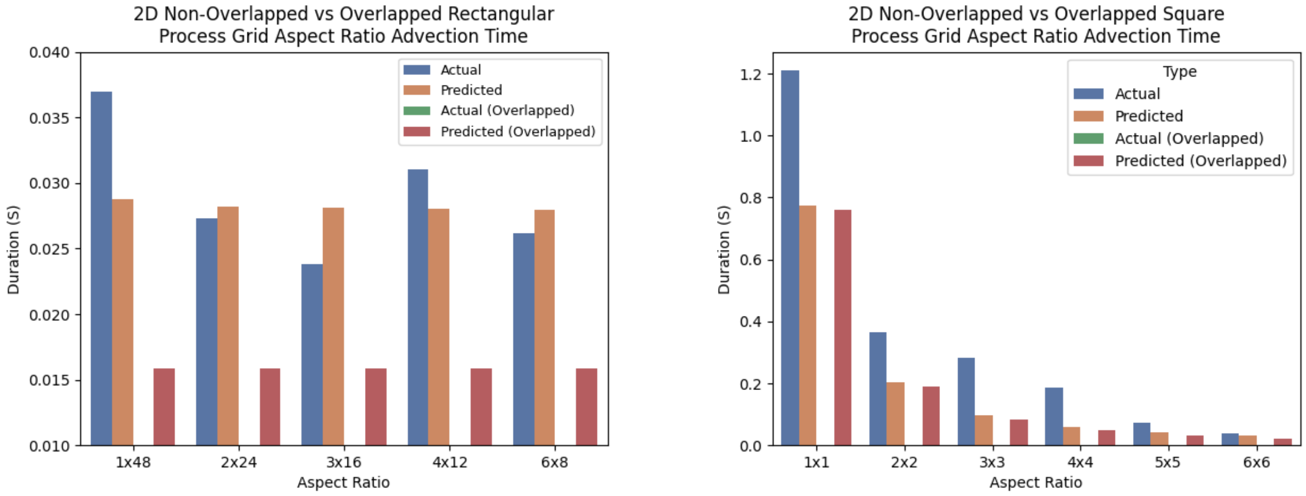


Figure 12: Comparison of overlapped and non-overlapped aspect ratios for advection time.

As we can see, both in the predicted and actual results, the overlapped communication does indeed have an affect on performance. Quite a significant one at that, even at the most optimum point of a 3 × 16 aspect ratio, there is a -43.54% change in performance between the non-overlapped and overlapped variants. On average, there is a -43.80% change for rectangular aspect ratios and -14.06% change for square aspect ratios.

We can see that for the square aspect ratios the effect of computation overshadows that of communication, given that they are smaller process counts compares to the rectangular aspect ratios. This would also suggest that the scaling of the aspect ratio matters more than the actual configuration of a fixed processor count. Logically this makes sense, as it corresponds to more parallelism in the solver and subsequently greater overlap. It should be noted that overlapped communication is not immune to the effects of strong scaling, as we can see that it falls in effectiveness at approximately the same rate as the non-overlapped counterpart.

6 Question 6

6.1 Wide Halo 2D Model Derivation

Accounting for halo width within our existing 2D model, can be done be re-evaluating the basis for which unique operations are performed within the r iterations. Notice that, we are performing communication of the halo every w iterations, and within these iterations successively smaller advection computations are combined to form the advection field updates.

With this in mind, we can change our model to be based on the accumulation of this chunk of work repeated $\lfloor \frac{r}{w} \rfloor$ times. It is important to recognise that we still get complete solutions even if iterations terminate mid-way through a chunk. Including this in our new model, we have the following form

$$\left\lfloor \frac{r}{w} \right\rfloor (t_{\text{comm}} + t_{\text{comp}})$$

Given that our model now considers chunks, we know that for a given chunk, communication only occurs once. As such we can directly use the form of the communication cost derived in the original 2D model. Including this results in the following

$$\left\lfloor \frac{r}{w} \right\rfloor \left(8 \cdot \left(\frac{Mt_w}{P} + \frac{Nt_w}{Q} + 2t_w + t_s \right) + t_{\text{comp}} \right)$$

A given chunk, will perform w successive advection field computations, where each computation is incrementally smaller in its size to evaluate the full halo over the independent centre region. Let us first consider our original 2D model expression for the computation cost

$$\frac{M}{P} \cdot \frac{N}{Q} \cdot t_f$$

For each of the local field dimensions, expressed here as ratios, we need to include an additional number of variable halo cells specified by w (Let D be a given field dimension ratio as $\frac{M}{P}$ or $\frac{N}{Q}$). Given that the halo is around the whole local field, we extend the notion of dimension to include this

$$D + 2w$$

Assuming we have some index i which identifies which iteration within the current chunk is being performed, we can adjust the expression uniformly based on this index to decrease the halo for each iteration

$$D + 2w - 2i$$

If we replace the original dimension ratios with this new expression, we have a form that expresses the computation cost for any given iteration within a chunk. Given that our new 2D model is in terms of chunks, we wrap this with a sum over the w iterations within a chunk for the final model

$$\left\lfloor \frac{r}{w} \right\rfloor \left(8 \cdot \left(\frac{Mt_w}{P} + \frac{Nt_w}{Q} + 2t_w + t_s \right) + \sum_{i=1}^{w} \left[\left(\frac{M}{P} + 2w - 2i \right) \cdot \left(\frac{N}{Q} + 2w - 2i \right) \cdot t_f \right] \right)$$

Or the alternative expanded form

$$\left\lfloor \frac{r}{w} \right\rfloor \left(8 \cdot \left(\frac{Mt_w}{P} + \frac{Nt_w}{Q} + 2t_w + t_s \right) + \frac{wt_f \cdot (3M \cdot (N + Q \cdot (w-1)) + P \cdot (w-1) \cdot (3N + 4Qw - 2Q))}{3PQ} \right)$$

6.2 Performance Evaluation

Let us examine the predictions made by the new model for the same advection configuration as that of the standard 2D model with dimensions $M = N = 2048$. In addition, we will add a new parameter of halo width which will be of increasing powers of 2, up until the largest uniformly supported halo for all cases ($2^{w_{\max}}$). This is determined by

$$w_{\max} = \left\lfloor \log_2 \left(\frac{\min\{M, N\}}{\max\{P, Q\}} \right) \right\rfloor$$

In our case, given the 2048×2048 field dimensions and largest grid dimension of 48, we will be using the following.

$$w_{\max} = \left\lfloor \log_2 \left(\frac{2048}{48} \right) \right\rfloor = 5$$

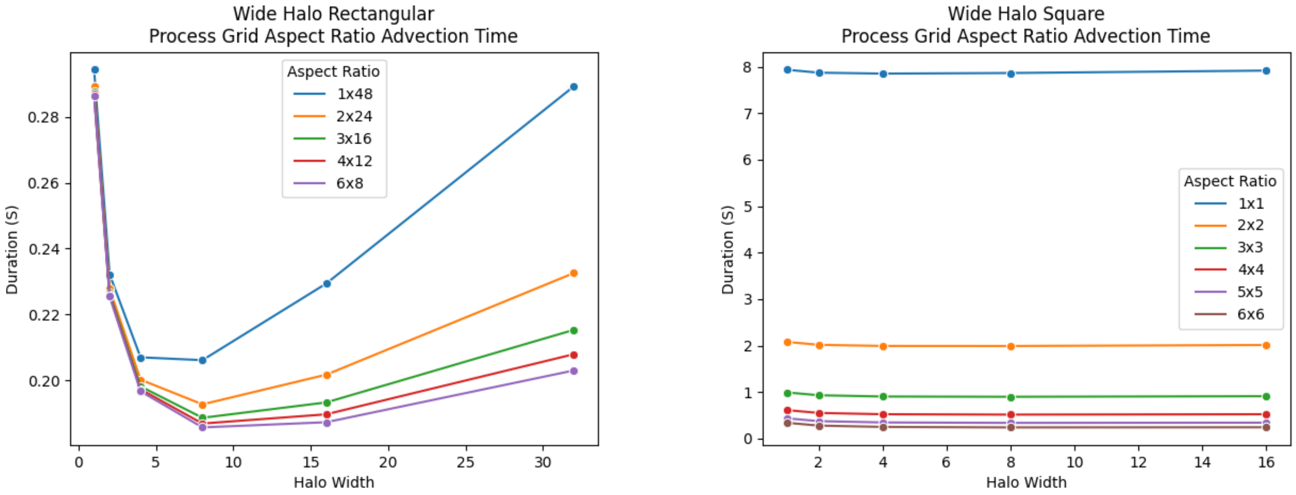


Figure 13: Advection time of rectangular and square aspect ratios for varying powers of 2 halo widths with an upper bound of half the advection field size ($2048/2 = 1024$)

Through figure 13, we can see that the behaviour of the increasing halo width is not totally degenerate, but limited in it's scope of improvement. For rectangular process grids, the balance between communication overhead and computation overhead holds in favour of optimisation until the $8 - 16$ halo width range. Beyond this point the performance improvement deteriorates as the computation overhead rivals the benefits of reduced communication cost.

Interestingly, for square process grids, there is substantially less (essentially none), performance improvement. To wit, the tapering appears to subside beyond a halo width of 2. However, it is easy to overlook the scaling factor, that for higher dimension process grids, this seemingly small improvement is actually fairly substantial. Notably for that of the 5 × 5 and 6 × 6 configurations, compared to the 2 × 2 for instance. Adjacent to this point, we should consider the scale of the problem space. In the grander scheme, 2048 × 2048 is not that large relative to the sizes of process-wise decomposition. One could infer the potential to see more significant improvements in that of larger problem spaces with larger decomposition.

6.3 Discussion

The use of wide halos is definitely beneficial, in that provides a means to reduce the overhead of communication at every iteration. In theory this could provide better scaling performance for wider halos, though a balance needs to be maintained between the overhead of large communications versus the cost of computation of larger local advection fields. With smaller processor availability this is a viable solution to catering for performance with communication hungry configurations, with skewed ratios of processes for P and Q.

However, for larger process grids, this becomes harder to optimise as there is a limit to effective halo size, being that of the local advection field size. At this point, the overlap is absolute and over-computation becomes dominant, leading to inefficiency of a different kind.

7 Question 7

Iterative Stencil Loops (ISLs) are computational models designed to evolve a grid with some given transition function for every cell based on considering a defined neighbourhood pattern. These are highly parallelisable computations at the behest of requiring communication and synchronisation between the threads to ensure consistency within the processing loops. The overhead introduced by the synchronisation and communication between the parallelised units of work can impede the performance of the solver.

Tiled stencil computations (TSCs) were introduced to address the overheads of communication and synchronisation by breaking up the problem space into an atomic unit of execution known as a tile (Ramanujam & Sadayappan 1992). During the execution of a tile, no communication or synchronisation is used to amortise communication start-up cost over the larger execution time of a given tile (Ramanujam & Sadayappan 1992). Over the problem space, tiles are required to be identical everywhere except for the boundary. Boundary tiles should be regular in the pattern as a consequence of regularity and identicality in the wider space. This is to address the complexities of partitioning the problem space efficiently and deterministically to ensure a balanced execution load for tiles. In order to reduce the communication and synchronisation even further, ghost zones can be used to partially overlap computation between tiles to avoid the need for communicating boundary context between tiles (Meng & Skadron 2010). One can look to overlap the boundary communication with the computation of the interior tile elements in order to amortise the boundary communication time.

2D TSCs are well defined and provide highly optimised data profiles in terms of array access and partitioning the data amongst tile sets. This extends to the choice of tile size which is relative to the problem space and infrastructure available. Prior work has explored the usage of automatic tile size determination to optimise this case. A supplementary technique for large tiled computations is to use Fourier transforms (Ahmad et al. 2021) on both the data set an convolution kernel (Yan et al. 2021) defined for the particular stencil computation (Januario et al. 2016). This technique optimises the matrix-matrix operations required to compute an evolution of a particular convolution kernel for large, dense matrices. Figure 14 outlines the general process.

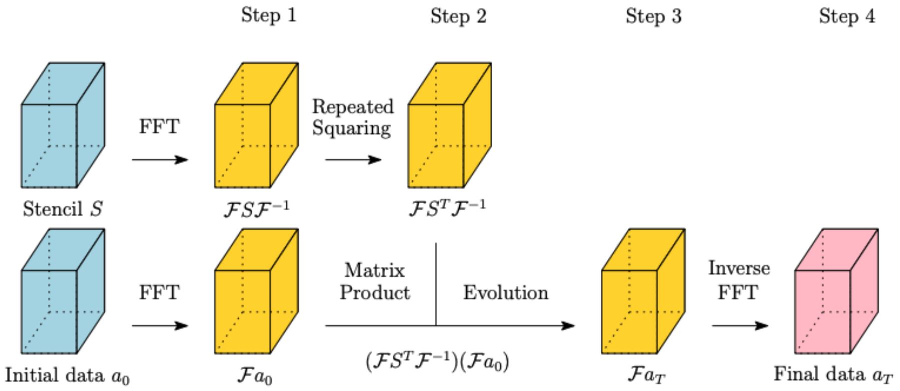
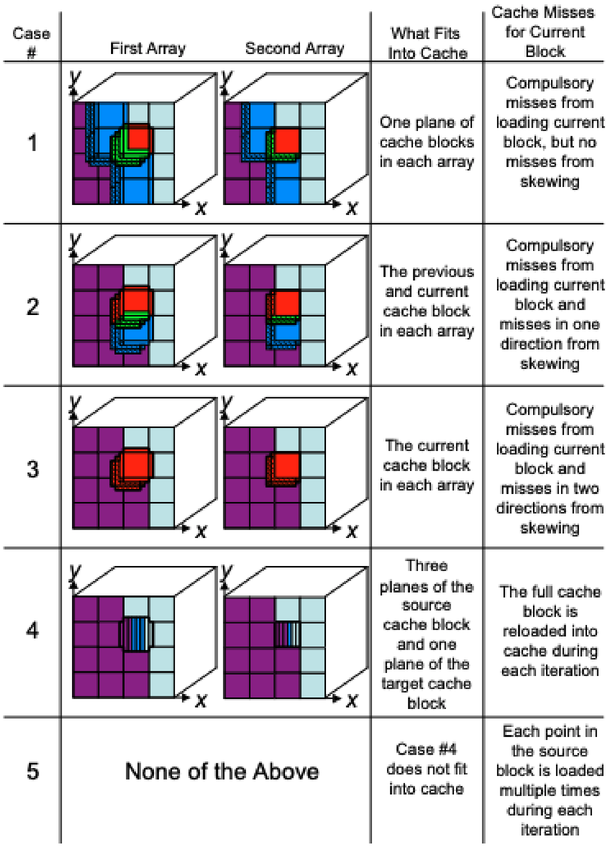


Figure 14: Outline of applying a FFT convolution kernel to a ISC tile. Image sourced from Ahmad et al. (2021)

Moving to the 3D and higher dimensional problem spaces introduces new issues. Suppose a 3D tiled Jacobi iteration with Von Neumann neighbourhood 1, cache accesses are limited by the extra dimensional constraint which bridges tile hyperplanes in the problem space (Datta et al. 2008). This introduces overhead in the processing time and also limits the access behaviour, noting that cache lines are replaced frequently where the array of data in the strip of a particular hyperplane is actively replacing the same cache line. This causes access conflict misses and poor cache locality (figure 15) (Rivera & Tseng 2000).



Case #	First Array	Second Array	What Fits Into Cache	Cache Misses for Current Block
1			One plane of cache blocks in each array	Compulsory misses from loading current block, but no misses from skewing
2			The previous and current cache block in each array	Compulsory misses from loading current block and misses in one direction from skewing
3			The current cache block in each array	Compulsory misses from loading current block and misses in two directions from skewing
4			Three planes of the source cache block and one plane of the target cache block	The full cache block is reloaded into cache during each iteration
5	None of the Above		Case #4 does not fit into cache	Each point in the source block is loaded multiple times during each iteration

Figure 15: Structure of arrays for 3D and their cache behaviour. Image sourced from Datta et al. (2008)

Wittmann et al. (2010) present optimisation techniques for both shared and distributed memory computation of ISLs. Specifically, they present a concept of temporary to "perform multiple in-cache updates on each grid cell before the result is evicted to memory thereby reducing effective code balance". In using this technique the cache behaviour is synchronised, to an extent, to the current cell state, ensure that cache lines are saturated by as much of the in-flight cell data during calculation. This attempts to push as much of the memory persistence to the end of a cell processing cycle. (They further expand on this idea with a pipelined model, though specific to shared memory architectures and out of the scope of the context of this report.)

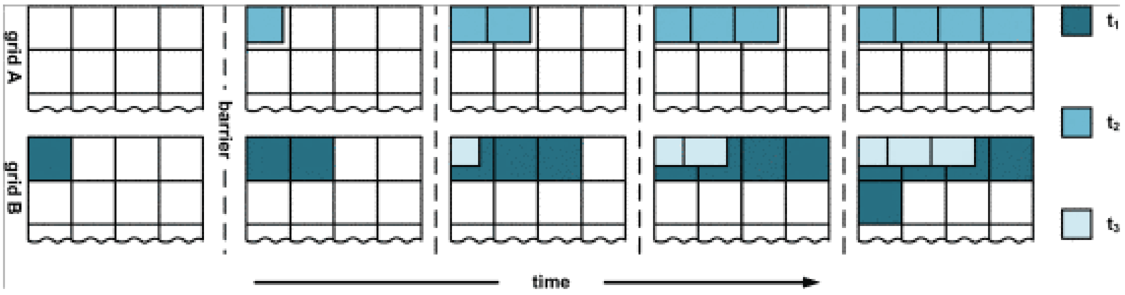


Figure 16: Illustration of cell-to-cache behaviour with the temporal blocking scheme. Image sourced from Wittmann et al. (2010)

Conventionally, the temporal blocking schemes presented in prior (Kowarschik 2004) cause multiple smaller update patterns to occur for a given cell, resulting in non-overlapping in-cache stencil updates when persisting cell data. Other approaches such as the cache-oblivious algorithms presented by Frigo & Strumpen (2005), attempt to consider the problem in a different direction with indepdence to cache size in the model itself, though resulting in irregular access patterns and large quantities of dTLB misses.

8 Question 8

Optimising memory hierarchy performance for TSLs is within the grasp of techniques previously described. We can consider the employment of the temporal blocking scheme over a standard distributed memory system for advection. We would expect to see optimised performance both on cache line usage, ensuring that data for timestep evolution is held primarily within the cache. Depending on how this is implemented, with regard to the cache size dependency, we could see improved virtual memory handling implying lower dTLB misses. This however would be contingent on problem scaling and available compute hardware, whether that be cores or nodes.

An interesting complimentary technique would be using TSCs to optimise the overheads of communication and synchronisation. This could potentially have positive effects on the remaining aspects of the implementation not directly bounded be the computational limitations (memory hierarchy inclusive). That being said, one could expect a trade-off to be formed on the interference in the cache behaviour during communication phases. To that end, depending on the implementation details of achieving armortised start-up cost with large halos, this can in theory could inconsistent cache membership lifetimes. More specifically, frequent full cache misses for overlapped computation during this phase could hamper the desired optimisations.

Touching on the techniques presented with Rivera et al.'s analysis of cache line behaviour, partial reads in regular intervals can be beneficial for hardware based prediction of both full-lookup memory access and cache hits. In noting this, we should focus on larger and more dense tiling, given that optimisations of access (and re-access) of larger memory areas that exceed the socket cache size can be exceedingly difficult.

In any case, there is a careful balance to be drawn between the two techniques, employing the use of tooling such as `vmtouch` and `vmprobe` to inspect and tailor OS level caching may provide an additional edge for addressing roughness in behaviour at a more general page level. These techniques would need to be tuned to the specific problem space depending on the hardware architecture and solution type (methodologies).

9 Question 9

9.1 Motiviation

From the previously detailed optimisation techniques for stencil computations, the FFT accelerated kernel convolution of tiles was implemented. Before analysing the performance results, we will first discuss the key aspects of the advection problem and it's links to kernel convolutions.

Firstly, the existing solver implements the Lax-Wendroff technique for solving systems hyperbolic conservation laws [ref: encyc of math] of the form:

$$\partial_t u + \partial_x f(u) = 0$$

> Where $u(x,t)$ is the vector of conserved variables, and $f(u)$ is the physical flux; the independent variables are x and t and are usually associated with space and time, respectively, where $x \in [0, L]$, for some positive constant L, and $t > 0$.

Quote 1: Defintion of Lax-Wendroff as defined by Encyclopedia of Mathematics (Springer)

Solving systems such as the one above, is typically done with conservative methods over a discretised spatial domain $[0, L]$ into M cells $I_i = \left[x_{i-0.5}, x_{i+1/2}\right]$. The dimension (single side) of these cells is given by the delta between elements of I_i, similarly, the temporary domain is discretised over timesteps of t^n. Generally, the conservative method is given in a form of calculating the evolution of a given timestep of the data set $\{u_i^n\}$ into $\{u_i^{n+1}\}$:

$$u_i^{n+1} = u_i^n + \frac{\Delta t^n}{\Delta x}\left[f_{i-0.5} - f_{i+0.5}\right]$$

This method is used for solving for linear advection. More specifically, for some scalar $u(x,t)$ denoting the position and time of a particular part of the advection space evolved by the flux defined as a linear function of $u(x,t)$; $f(u) = au$, where a is some constant speed of propagation. Using a Taylor series expansion in time, replacing time derivatives for spatial derivatives and approximating them via central differences, the solution for an element is as follows:

$$u_i^{n+1} = b_{-1}u_{i-1}^n + b_0 u_i^n + b_1 u_{i+1}^n$$

Where the coefficients b_k are functions of the Courant number

$$c_x = a\frac{\Delta t}{\Delta x}$$

$$b_{-1}^x = \frac{c_x}{2}(1 + c_x)$$

$$b_0^x = 1 - c_x^2$$

$$b_1^x = -\frac{c_x}{2}(1 - c_x)$$

For the M cells, the above forms are applied as kernel convolution over the previous state $\{u_i^n\}$ on a per-index basis. For the 2D generalisation, the M cells are grid of dimensions $p \times q$, with corresponding y variable coeffecients of the same form:

$$c_y = a\frac{\Delta t}{\Delta y}$$

$$b_{-1}^y = \frac{c_y}{2}(1 + c_y)$$

$$b_0^y = 1 - c_y^2$$

$$b_1^y = -\frac{c_y}{2}(1 - c_y)$$

as such the linear form of the next element is to be applied over the n dimension as a full 2D convolution of the form:

$$\mathcal{M} = \begin{bmatrix} b_{-1}^x \cdot b_{-1}^y & b_{-1}^x \cdot b_0^y & b_{-1}^x \cdot b_1^y \\ b_0^x \cdot b_{-1}^y & b_0^x \cdot b_0^y & b_0^x \cdot b_1^y \\ b_1^x \cdot b_{-1}^y & b_1^x \cdot b_0^y & b_1^x \cdot b_1^y \end{bmatrix}$$

$$\forall i \in [0, m], j \in [0, n] \quad u_{i,j}^{n+1} = u_{i,j}^n * \mathcal{M}$$

This is precisely the methodology we see implemented in `serAdvect.c` in `updateAvectField`. The b_k^d coefficients of the Courant numbers of each dimension are given by the `N2Coeff` method invoked for each dimension of the advection parameters. It is important to recognise that the form of this convolution is identical for all timesteps t^n, and at that the application of $\mathcal{M}_n$ is periodic for periodic boundary conditions. Here we can use the circular discrete convolution (given that $\mathcal{M}_n$ is applied as a periodic summation of $\mathcal{M}$), limited to the interval of $[0, n-1]$ reducing the convolution to

$$\left(f * g_N\right)[n] = \sum_{m=0}^{N-1} f[m] g_N[n-m]$$

$$= \sum_{m=0}^{n} f[m] g[n-m] + \sum_{m=n+1}^{N-1} f[m] g[N+n-m]$$

$$= \sum_{m=0}^{N-1} f[m] g[(n-m)_{\bmod N}]$$

$$\triangleq \left(f *_N g\right)[n]$$

With this framework, we can utilise the convolution theorem for the discrete-time Fourier transform (DTFT) for computing the convolution of two sequences as the inverse transform of the produt of the individual transforms. In this case, the two sequences are the current state $\left\{u_{i,j}^n\right\}$ and the kernel $\mathcal{M}_n$:

$$x * y_N = \mathrm{DTFT}^{-1}\left[\mathrm{DTFT}\{x\} \cdot \mathrm{DTFT}\{y_N\}\right] = \mathrm{DFT}^{-1}\left[\mathrm{DFT}\{x_N\} \cdot \mathrm{DFT}\{y_N\}\right]$$

Which leads to the convolution of the form

$$\mathcal{F}_{\mathrm{DFT}^{-1}}^{-1}\{X \cdot Y\}_n = \sum_{\updownarrow=0}^{N-1} x_{\updownarrow} \cdot y_{(n-\updownarrow)\mathrm{mod}N}$$

9.2 Implementation

Utilising the above detailing of our newly transformed problem space into that of utilising the convolution theorem for DFTs, we can implement a variation of this using the `FFTW3` library for fast Fourier transforms (FFTs), in a manner similar to that of image processing techniques. A skeleton of the convolution process using `FFTW3` is as follows

Note: Almost all standard distributions of Linux have the `FFTW3` *library available, either natively or as a package via* `apt` *or* `apk`. *Gadi has a module for the use of FFTW3,* `module load fftw3` *can be used to load it.*

```
static double complex* laxWendroffKernel = padMatrix({
    cim1 * cjm1, cim1 * cj0, cim1 * cjp1,
    ci0 * cjm1, ci0 * cj0,ci0 * cjp1,
    cip1 * cjm1, cip1 * cj0, cip1 * cjp1
}); // All dependent variables known at startup
double complex* u = { ... }; // Input data

double complex* lwk_f;
dft(/*in:*/ lwk, /*out:*/ lwkf);
repeatedSquaring(lwk_f); // Parallel

double complex* u_f;
dft(/*in:*/ u, /*out:*/ u_f);
product(/*in:*/ u_f, /*inout:*/ lwk_f); // Parallel

double complex* v; // Output data
idft(/*in:*/ lwk_f, /*out:*/ v);
```

This method is directly referenced from the prior work of Ahmad et al. (2021) for the periodic basis. Note that the original implementation uses OpenMP as the parallelisation framework for the various matrix and vector operations. We replace this usage with MPI interfaces via collection, for `MPI_Scatter` and `MPI_Gather` abstractions. In this regard, our parallel decomposition is on the basis of linear algebra computations as a distributed workload. Given that we had previously noted that the optimisations from utilising Fourier transforms rely on the periodicity of g_N convolved over f as $f * g_N$, we require absolute periodicity over the problem space. As such decomposition into discrete parallel computations over the spatial domain is not feasible. Instead, parallelisation is achieved through the various linear algebra computations balanced against the communication overhead and efficiency of the `FFTW3` library.

References

Ahmad, Z., Chowdhury, R., Das, R., Ganapathi, P., Gregory, A. & Zhu, Y. (2021), 'Fast stencil computations using fast fourier transforms'.

Baruffa, F. (2020), 'Overlapping computation amp; communication to improve mpi performance'.
URL: *https://www.intel.com/content/www/us/en/developer/articles/technical/overlap-computation-communication-hpc-applications.htmlgs.vj0erg*

Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J. & Yelick, K. (2008), Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, *in* 'Proceedings of the 2008 ACM/IEEE Conference on Supercomputing', SC '08, IEEE Press.

Frigo, M. & Strumpen, V. (2005), Cache oblivious stencil computations, *in* 'Proceedings of the 19th Annual International Conference on Supercomputing', ICS '05, Association for Computing Machinery, New York, NY, USA, p. 361–366.
URL: *https://doi.org/10.1145/1088149.1088197*

Januario, G. C., Rosenburg, B. S., Park, Y., Perrone, M., Moreira, J. & Carvalho, T. C. (2016), Speeding up stencil computations with kernel convolution, *in* '2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)', IEEE.
URL: *https://doi.org/10.1109/sbac-pad.2016.18*

Kowarschik, M. (2004), Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchical Memory Architectures, PhD thesis.

Meng, J. & Skadron, K. (2010), 'A performance study for iterative stencil loops on GPUs with ghost zone optimizations', *International Journal of Parallel Programming* **39**(1), 115–142.
URL: *https://doi.org/10.1007/s10766-010-0142-5*

Ramanujam, J. & Sadayappan, P. (1992), 'Tiling multidimensional iteration spaces for multicomputers', *Journal of Parallel and Distributed Computing* **16**(2), 108–120.
URL: *https://www.sciencedirect.com/science/article/pii/074373159290027K*

Rivera, G. & Tseng, C.-W. (2000), Tiling optimizations for 3d scientific computations, *in* 'Proceedings of the 2000 ACM/IEEE Conference on Supercomputing', SC '00, IEEE Computer Society, USA, p. 32–es.

Wittmann, M., Hager, G. & Wellein, G. (2010), Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory, *in* '2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)', pp. 1–7.

Yan, Q., Jiang, S. W. & Harlim, J. (2021), 'Kernel-based methods for solving time-dependent advection-diffusion equations on manifolds'.