COMP4300/8300 Parallel Systems Assignment 1, 2021

A Distributed Advection Solver Using MPI

Deadline: 17:00 Friday April 23

(Please report any errors, omissions and ambiguities to the course lecturer)
(amendments to this document after its release will be marked in blue).

This assignment is worth 25% of your total course mark. It will be marked out of 40.

Assignment Setup and Submission

This assignment must be submitted electronically. To obtain a copy of the template source files, you must *fork* the gitlab project `ps21-ass1` into your own gitlab project and *push* the final version of your files into your `ps21-ass1` gitlab project before the deadline.

The files that will be marked are a report `ps-ass1Rep.pdf` and a C file `parAdvect.c`.

Learning Objectives

- Implement a distributed algorithm using message passing.
- Model the computation and communication costs of the algorithm.
- Measure the performance of the implemented code, and compare against the model.
- Research and evaluate relevant parallel computing literature.

See the Advection Notes for the necessary background.

Setup

Your assignment project directory will contain a test program `testAdvect.c`, a file `serAdvect.c` containing a serial advection solver and support function, some header files, and a template parallel advection solver `parAdvect.c`. The test program can be built using the command `make`.

It also contains a template `ps-ass1Rep.pdf`, which you must overwrite with your own report.

The usage for the test program is:

```
mpirun -np p ./testAdvect [-P P] [-w w] [-o] [-x] M N [r]
```

with default values *p=1, P=p, w=1, r=1*. This will run an *M* by *N* advection simulation over *r* timesteps (repetitions) over a *P* by *Q* process grid, where *p=PQ*. *w* specifies the `halo' width (normally it is 1). If `-o` is specified, halo communication should be overlapped with computation. The `-x` is used to invoke an optional extra optimization.

There is also a `-v v` option which can be used for debugging (try using `-v 1`, `-v 2` etc).

The test program initializes the local field array (`u,ldu`), calls the appropriate parallel advection function (in `parAdvect.c`), and determines the error in the final field. It assumes a 2D block distribution of the global advection field. However, `parAdvect.c` determines the details of this distribution (i.e. how to deal with remainders), and exports:

- `M_loc`, `N_loc`: the local advection field size (excluding the halo).
- `M0`, `N0`: the local field element (0,0) is global element (M0,N0)

You should read the files and familiarize yourself with what they contain.

Your Tasks

Experimentation for this section should be done on the Gadi supercomputer using batch jobs.

1. *Deadlock issues.*
   The prototype function `parAdvect()` in `parAdvect.c` should work for $Q=1$. However, it can deadlock (try $p=2$, $M=2$ and a large N). In your report, write down for what values of N it deadlocks, and explain why. Without using non-blocking send or receives, or buffered sends, fix the halo-exchange code in `parAdvect.c` to avoid the deadlock. Describe the fixed code in your report.

2. *The effect of non-blocking communication.*
   Rewrite your halo-exchange code to use non-blocking sends and receives, and use a CPP switch so you can activate either version of communication. Do some experiments on up to 4 nodes to determine if this improves performance (choose parameters to maximize the impact of communication), and write the results in your report.

   From now on, use the better-performing (or otherwise preferred, if performance is much the same) version of communication, leaving the other version commented out.

3. *Performance modelling and calibration.*
   In your report, write a performance model for the computation, in terms of the above program parameters, and the coefficients for communication startup time (t_s or a), communication cost per word time (t_w or b), and per element computation time (t_f or c) for the advection solver.

   Run experiments to determine the values of these parameters on Gadi, and justify your methodology. *Hint:* you can use the test program itself with suitably chosen parameters for t_f; we have measured the communication costs previously. Comment whether the values are (roughly) what you would expect for Gadi.

   Within one Gadi node, compare predicted vs actual execution time for various p for strong scaling. Use the same value of M, N, r throughout, and justify why you think those values are suitable. *Hint: $r=100$* should be `big enough' to amortize startup effects; suggest keeping the field size to fit within shared L3$ of a single socket of the Intel Xeon 8274 (Cascade Lake).

   Tabulate the results in your report, commenting on any discrepancies. *Hint:* at certain points, other parts of the memory hierachy may have an effect on the timings.

4. *The effect of 2D process grids.*
   Extend your code in `parAdvect()` for two-dimensional process grids, i.e. $Q\geq 1$, debug, and test. Extend your performance model accordingly.

   Hints: the existing code assumes $Q=1$. You will need to modify `initParParams()` for $Q>1$, possibly some of the top-bottom halo exchange in `updateBoundary()` as well as adding message passing code for the left-right exchange. 2D block distributions have a number of valid variants; you may choose any of these.

Conduct experiments with fixed *M=N* on one node to see if the process grid aspect ratio has any effect on performance. Of most interest is whether a (near-) square ratio has a different effect to the default (*Q=1*). What is the optimum aspect ratio predicted by your model? Use your model to predict how much difference there would be if the coefficient t_w was 10 times larger.

Also conduct experiments on 1 to 4 nodes (48 to 192 cores). Do this also for experiments in the questions below. *Hint:* restrict the problem size to fit within the combined L3$ of 2 Cascade Lake sockets (48 cores).

5. *Overlapping communication with computation.*
   An advanced technique in message-passing parallel algorithms is to hide communication costs by updating the data that must be communicated (for the next step) first, and send this before updating the rest of the data. In this case, it is the `inner halo'. Implement this technique in `parAdvectOverlap()`.

   In your report, discuss what the performance impact of this technique might be, and describe how it would affect your performance model. Using the `-o` option, run appropriate experiments to determine whether it is effective, and record and discuss in your report. It will be sufficient to implement this just for 1D process grids, i.e. for *Q=1*. However, in this case, you should explain why achieving overlap for 2D communication is difficult.

6. *Wide halo transfers*.
   In stencil computations, *wide halos* are a technique intended to reduce parallel overheads. Instead of exchanging a halo of width 1 at process boundaries every simulated timestep, a halo of width $w \geq 1$ can be exchanged every *w* timesteps. In each process, let the local advection field size (without halos) be *(m)*x*(n)*. You will notice that the test program allocates an *(m+2*w)*x*(n+2*w)* array to support this. Using the widened halo, we perform *w* updates to the local field of sizes *(m+2*w-2)*x*(n+2*w-2)*, *(m+2*w-4)*x*(n+2*w-4)*, ..., *(m)*x*(n)*.

   Like the normal halo, the widened halos are generated from the interior points on the current or neighbouring processes. However this is problematic if *w > m, n*. The function `checkHaloSize()` checks for this problem, but creates an untidy shutdown. Fix this so that if the above condition is violated in any process, all processes exit and only a single error message is printed out.

   In your report, discuss what you think is the potential advantage of wide halos, and describe how it would affect your performance models. In the function `parAdvectWide()` in `parAdvect.c`, extend your 2D codes with the wide halo technique and test it using the `-w` option. Run appropriate experiments to determine whether it is effective, and record and discuss in your report.

7. Search the literature and report on techniques to improve the performance of stencil computations. *Hint:* the *tiled stencil* technique may be interesting. Provide a summary of the technique, including its motivation and its effectiveness (150-400 words). Fully cite all sources (2+ sources are expected).

8. *Optional.*
   The performance of stencil computations is primarily limited by memory system performance. Briefly explain whether a combination of two of the techniques mentioned above may alleviate this, and if so what are the performance tradeoffs of such an approach.

9. *Optional.*
   Find and describe some new optimization that could potentially improve the performance of the parallel advection solver. If possible, implement and evaluate this on Gadi. Put your code in `parAdvectExtra()`, which is activated by the `-x` option.

Requirements

Your code in `parAdvect.c` will be tested against the standard `serAdvect.c` and `testAdvect.c` and must compile and run correctly with them. Solutions which are highly incompatible may be rejected entirely. You code should be written with good programming style (and so should not produce any warnings when compiled with the `-Wall` option!). It should be well commented, so as to enable the reader (this includes the tutor/marker!) to understand how it works.

In your report, include:

- A disclaimer with your name and student number stating what (if any) contributions from others (not including course staff) are in your submission. Note that significant contributions may require a revision of your final mark, as this is intended to be an individual assignment.
- Your answers to all the questions given above.
- Details of any notable deficiencies, bugs, or issues with your work.
- Any feedback on what you found helpful in doing this work, or what from the course could be improved in helping you to carry out this work.