Performance of a shallow water model using MPI

Joakim Kjellsson
Abubakr Salih
Department of Meteorology, Stockholm University

November 7, 2012

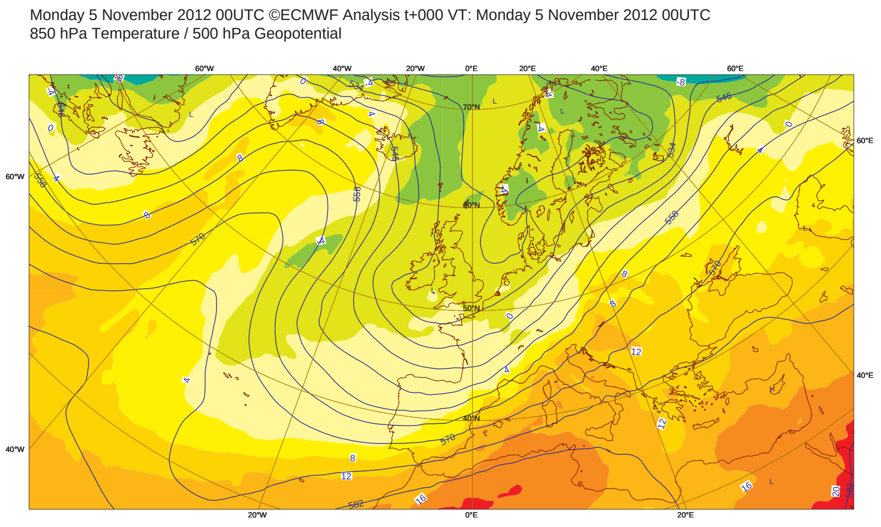
Monday 5 November 2012 00UTC ©ECMWF Analysis t+000 VT: Monday 5 November 2012 00UTC
850 hPa Temperature / 500 hPa Geopotential
60°W
40°W
20°W
0°E
20°E
40°E
60°E
70°N
60°N
50°N
40°N
40°W
60°E
40°E
546
558
570
582
0
4
8
12
16
20

Figure 1: *Geopotential height at 500 hPa and Temperature and 850 hPa. The wave structure is typical of a Rossby wave.*

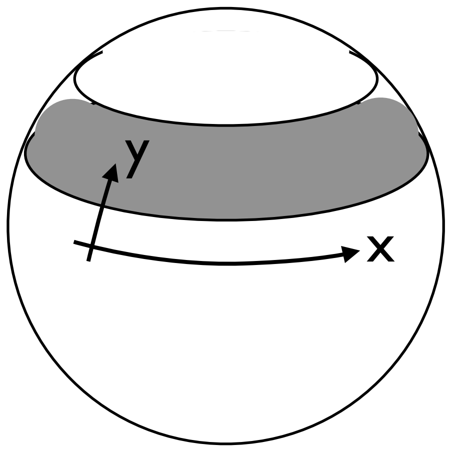

Figure 2: *Schematic of a middle latitude band in the atmosphere which the model can be thought to represent.*

1 Introduction

Rossby waves are planetary waves that are found in the middle latitudes of the Earth's atmosphere. The very first numerical weather prediction model predicted the large-scale winds at ~ 5 km height by *e.g.* calculating Rossby waves. We aim at simulating Rossby waves using the linear shallow water equations in two dimensions.

Although the numerical model we will use is very simple, we aim to run it as if it was highly sophisticated. We will parallelise the model using MPI, run it on a supercomputer, and investigate the performance. The simplicity of the numerical model will highlight some important features and aspects of high-performance computing.

2 The model

The linear barotropic shallow water model (eqs. (1)-(3)) is a very simple, yet powerful representation of large-scale geophysical flows.

2.1 Domain

We consider a domain that represents the Earth's atmosphere in a latitude band in the middle latitudes, e.g. 30°-60° (Fig. 2. In the domain, the Coriolis parameter and mean fluid depth (both described below) can vary with y. As the Earth is spherical, we use periodic boundaries in x and relax the velocities toward zero near the y boundaries, representing open boundaries.

2.2 Shallow water equations

The model is based on a linearised and simplified form of the Navier-Stokes equations known as the *shallow water*[1] equations in two dimensions,

$$\begin{aligned}
\frac{\partial u}{\partial t} - fv + \frac{\partial h}{\partial x} &= \frac{\partial \tau}{\partial z} + A\nabla^2 u - \mu u && (1)\\
\frac{\partial v}{\partial t} + fu + \frac{\partial h}{\partial y} &= A\nabla^2 v - \mu v && (2)\\
\frac{\partial h}{\partial t} + D\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) &= 0, && (3)
\end{aligned}$$

where u and v are zonal (east-west) and meridional (north-south) velocities respectively, and h is the height field. $f(y)$ is the Coriolis parameter, $D(y)$ is the mean depth of the fluid, and $\tau(y)$ is a zonal driving force. Only f varies with y, while τ and D are kept constant. The left hand sides of eqs. (1)-(3) describe the model dynamics. The right hand sides are then the parameterisations (sometimes known as "model physics"). A is the coefficient for Laplacian diffusion, and μ is the coefficient for Rayleigh friction. In our simulations, $A = 0$ and $\mu = 0$.

2.3 Discretisation

Eqs. (1)-(3) are discretised on an Arakawa C-grid (Mesinger and Arakawa, 1976) with a leap-frog scheme in time, yielding 2nd order accuracy in both space and time. The leap-frog scheme is initialised by using the Euler forward scheme for the first time step. Two-level schemes such as leap-frog result in numerical modes that, in worst case scenario, makes the solution unstable. This problem is solved by using a Robert-Asselin filter (Asselin, 1972) with $\gamma = 0.2$. The driving force τ generally varies with z, but is due to the lack of a vertical dimension, set to $\frac{\partial \tau}{\partial z} = C_D \rho_a u_s^2/(\delta_E \rho_w)$, where C_D is the drag coefficient, u_s is the surface wind, δ_E is the Ekman depth and ρ_a and ρ_w are the densities for air and water respectively. For simplicity, in these experiments, $\frac{\partial \tau}{\partial z} = 10^{-6}$ m s^{-2}.

2.4 Parallelisation

The model is parallelised by sub-dividing the domain into subdomains. Fig. 3 shows a schematic of how the domain is sub-divided. Division is done in the x direction, i.e. each subdomain is of the size NX = IMT/(nranks − 1) and NY = JMT, where (IMT, JMT) is the size of the total domain. Each subdomain is given "ghost cells" to the west of the first point and to the east of the last point containing the end points of the adjacent subdomains. Hence, $u_n(x = 0) = u_{n-1}(x = \text{end})$, where n is the subdomain index. The east-most subdomain is assumed to lie to the west of the west-most subdomain to imitate

[1]One of the key assumptions is that the horizontal length scales are much greater than the vertical, hence a shallow fluid

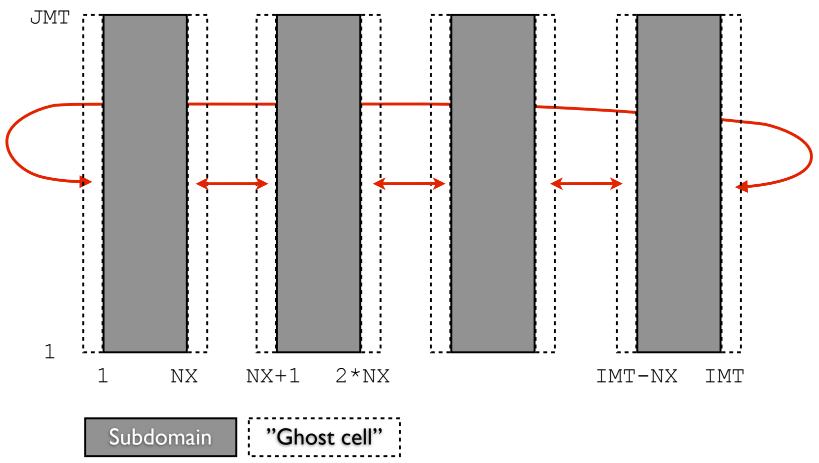

Figure 3: *Dividing the domain into subdomains. Each subdomain has "ghost cells" to the left and right, which contain data from the adjacent cells. This way, gradients can be calculated between subdomains. MPI is used to keep the "ghost cells" in sync with the adjacent subdomains. The left and rightmost subdomains communicate since the domain is periodic in x.*

Table 1: *Some specifications of the Ekman and Lindgren clusters*

	`lindgren.pdc.kth.se`	`ekman.pdc.kth.se`
Cores/node	24	8
Chip	AMD Magny-Cours (2.1 GHz)	AMD 2374HE (2.2 GHz)

periodic boundaries. Message-Passing is used to communicate initial conditions and results between the subdomains, and each subdomain is computed as a separate task.

2.5 Resources

The model is written in Fortran 90/95 with Message-Passing and output to netCDF files. Parameters such as the slope of f and the domain size are set in a namelist.

Model integrations are done at the Lindgren and Ekman supercomputers at PDC. Lindgren is the country's fastest machine in terms on flop/s, and Ekman is the cluster used for climate modeling at Stockholm University and KTH. A brief overview of both clusters is presented in Table 1.

2.6 Model results

The result of the model is propagating Rossby waves (Fig. 5). In the initial state, a Gaussian height perturbation is in geostrophic balance with the velocity field. At the end of simulations, after 10 model days, the single perturbation has dispersed into two wave crests and one trough. While Rossby waves always have a negative phase speed (individual crests move to the left), the group velocity (at which a wave packet travels) may be either negative or positive.

3 Optimizing a non-parallel simulation

To obtain the best performance of a parallel simulation, we will first optimize the model for running as a single process. This is mainly a two-fold problem; optimizing the code, and using the optimal software. Firstly, the code is optimized by using the cache memory efficiently (loop ordering), non-blocking communication between the processes, and minimizing the communication between processes overall. Secondly, several configurations were tested to find the optimal software and hardware. Table 2 shows the total simulation time when using various fortran compilers and optimization flags on both Ekman and Lindgren. The flags differ between compilers, but we selected flags to include vectorization, unrolling of small loops, and some loss of precision. The model outputs to netCDF files, which limits us to only using the Intel and GNU compilers on Ekman, and only the Intel and PGI compilers on Lindgren.

For the Intel and PGI compilers, `-O2` is default, while `-O0` is default for GNU. We note that using the `-O2` is better than using `-O0`, and that the difference is huge when using the Intel compiler, less huge for GNU, and relatively small for PGI. In fact, `ifort -O2` is the fastest configuration of all simulations. Additional optimizations such as loop unrolling will only effect the short loops in the initialization process, and the model is fairly simple, such that inter-procedural optimization will not have a large impact. For the PGI compiler the `-tp=amd64 fast` optimizes for the specific architecture at Lindgren, and enables vectorization and faster, less-accurate floating-point operations. This resulted in some speed-up, while the Intel compiler did not benefit from faster math at all. Naturally, the Intel compiler does not have any optimizations for AMD architectures, but vectorization with SSE2 instructions was enabled by default at Lindgren. We also note that Ekman is considerably faster than Lindgren, at least with the Intel compiler.

In conclusion, by using the optimal compiler and flags, we increased performance by orders of magnitude with very little additional work. The model output is written to netCDF files, which is generally slower than writing to unformatted binary files. Also, output is written for several time steps, rather than the model accumulating the time series and writing only at the end. This saves memory, but has a negative impact on performance. We have constructed

Table 2: *Average of run time with different compilers and different compilation flags. The lines with * are performed on Ekman, however the others are run on Lindgren.*

Flags	Intel	PGI	GNU
-O0	41m27s	12m48s	
-O0*	22m18.312s		23m17s
-O2	3m41s	10m11s	
-O2*	2m56s		6m 33s
-O2 tp=amd64 fast		9m26s	
-O2 -unroll4 -ip	3m40s		
-O3* -ffast-math			6m1s
-O2 -fp-model fast -ip	3m42s		

the model this way because it is the general behavior of more sophisticated atmospheric/oceanic general circulation models.

4 Parallel simulations with Message-Passing

For the parallel simulations, we used the Intel compiler with the `-O2` flag on Lindgren and Ekman. However, as the performance analysis tools on Lindgren do not work with Intel, we use `pgf90 -O2` to do some deeper analysis of the performance. We allocated up to 101 cores. Fig 6 shows the speed-up on the two clusters for the Intel compiler. They both scale nicely up to 51 cores, but then the Ekman does not benefit as much anymore. There is only a very small gain by using 101 cores instead of 51 on Ekman. The Lindgren cluster gives a speedup from 51 to 101 cores, but also deviates from the optimal line. It is possible that the communication time becomes comparable to the computational time at this point. Lindgren does not suffer as much slowdown, possibly because the communication between nodes is smaller since only 3 nodes are needed for a 51 core run, while 7 are needed on Ekman.

Fig. 4 shows the percentage of the simulation time spent on different actions for a 3 and an 11 core run with PGI on Lindgren. Calculating u, v, h (which is done in three separate subroutines) accounts for most of the total time in both runs. However, for the 11 core run, almost 1/5 of the time is spent on MPI processes, and utilizing more cores would lead to larger percentage of MPI time. At some limit, the MPI processes take up such a substantial amount of time that increasing the number of cores will not lead to any significant speedup. We believe that for our runs, this point is at about 51 cores on Ekman and near 101 cores on Lindgren (Fig. 6).

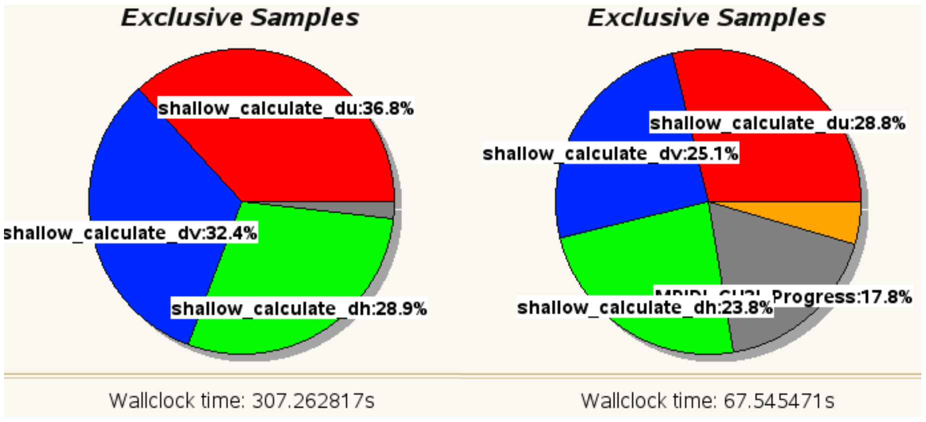

Figure 4: *Percentage of total time per task spent on different groups of code. Left pie chart shows a simulation on 3 cores, and the right shows a simulation on 11 cores. On the left, grey area shows MPI + other processes, while to the right grey shows MPI only. Increasing the number of cores decreases the percentage of time spent on calculations, but increases the percentage spent on communicating.*

5 Concluding remarks

We have performed simulated Rossby waves using a linear two-dimensional shallow water model on the Ekman and Lindgren clusters. We have measured the simulation time for parallel simulations where each core is assigned to a subdomain. The result is that there is a large decrease in simulation time with an increasing number of cores, up to a limit around 51 cores. At this point, the simulation time is a few seconds, and optimal speedup becomes hard to achieve.

We also conclude that it is important to find the optimal compiler and optimization flags for it. With default settings, the PGI and GNU compiler took 13 and 23 minutes to simulate. Using the Intel compiler with the simple optimization `-O2` (which is default) took nearly 4 minutes on Lindgren and 3 minutes on Ekman. In the climate modeling community at Stockholm University, most sophisticated simulation are done at Ekman with the Intel compiler, which thus seems a proper choice.

References

Asselin, R. A., 1972: Frequency filter for time integrations. *Mon. Weather Rev.*, **100**, 487–490.

Mesinger, F. and A. Arakawa, 1976: Numerical methods used in atmospheric models. *GARP Publication Series*, **17**.

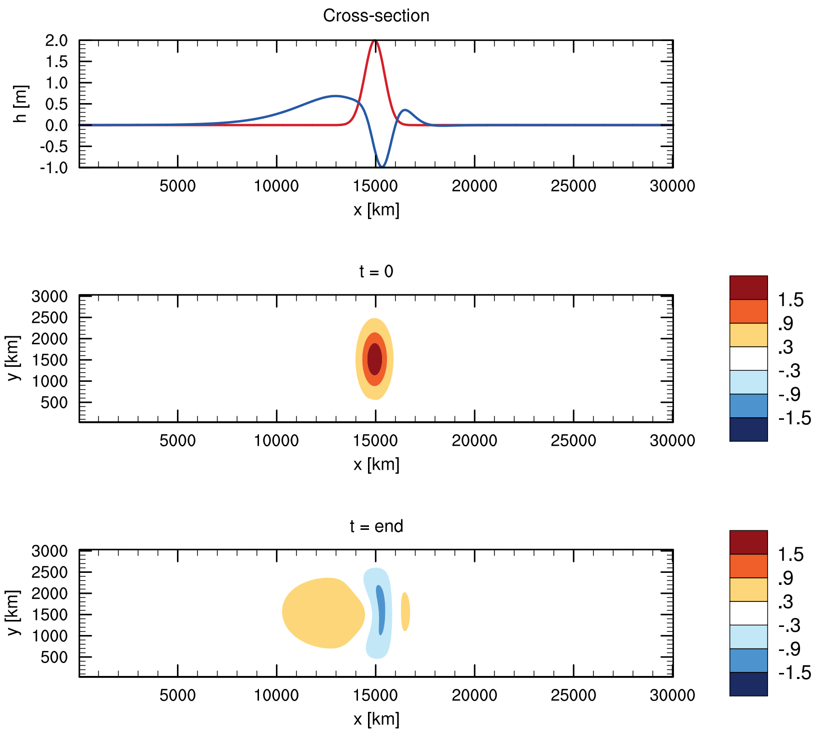

Figure 5: *The result from the model. The initial Gaussian disturbance is shown in the middle panel, and a cross section of it is shown as a red line in the top panel. The end result after 10 model days is shown in the bottom panel and a cross section of it is shown as a blue line in the top panel.*

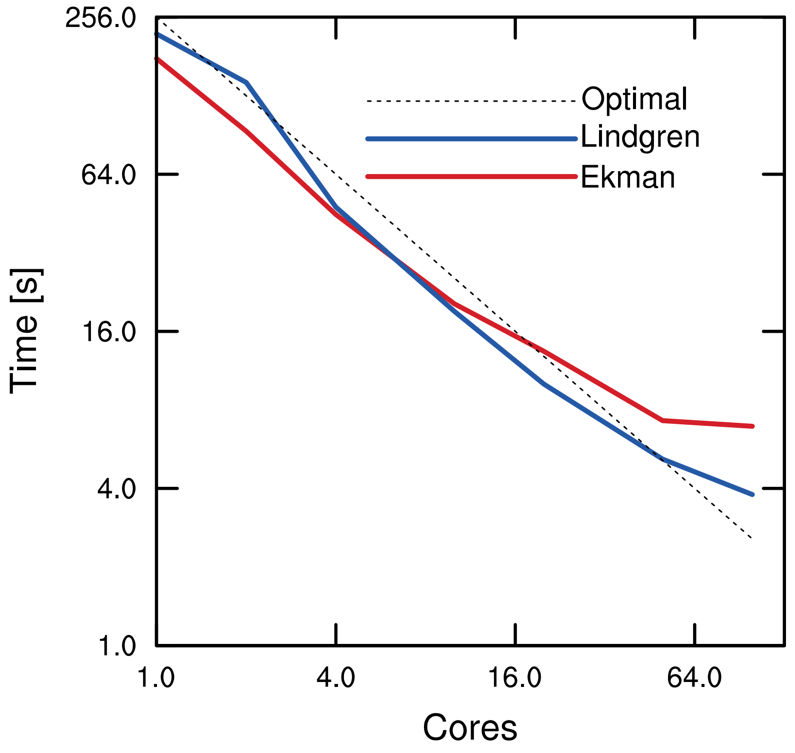

Figure 6: *Simulation time as a function of number of cores* ($\mathrm{nranks}-1$) *on Ekman (red) and Lindgren (blue) and optimal speedup (black dashed). If the scaling is optimal, the solid lines would have the same slope as the dashed line. Scale is* $\log_2$.