SnT 2023

CTBT: SCIENCE AND TECHNOLOGY CONFERENCE

HOFBURG PALACE - Vienna and Online

19 TO 23 JUNE

Performance Enhancement of Flexpart Atmospheric Transport Model for GPU Environment

Don Morton[1], Wolfgang Sommerer[2], Jolanta Kusmierczyk-Michulec[2], Robin Schoemaker[2], Anne Tipka[2]

[1]Boreal Scientific Computing, Fairbanks, Alaska, USA

[2]CTBTO, Vienna, Austria

Boreal Scientific Computing LLC

INTRODUCTION

This is an overview of an ongoing EU-funded project to enhance the performance of the Flexpart Atmospheric Transport Model by using Graphical Processing Units (GPUs)

METHODS/DATA

This section outlines the full workflow from GPU software installation, to setting up baseline Flexpart case, to compilation, execution, and first steps of OpenACC usage

START

RESULTS

Description of various problems and solutions, creation of test mockup, and ongoing results

CONCLUSION

This continues to be a work in progress. Much has been accomplished, and there is still much to do for verification of correctness and performance enhancement. Additional work includes preliminary test on Flexpart 11

Please do not use this space, a QR code will be automatically overlayed

P4.3-486

Introduction

The CTBTO Atmospheric Transport Modeling (ATM) group runs a large number of compute-intensive operational simulations on a daily basis in support of its mission.

The foundation of the simulations is the legacy Flexpart Lagrangian transport and dispersion model, which has undergone extensive modifications and enhancements in its 25 years.

With the recent CTBTO acquisition of GPU hardware, it was decided to explore the usefulness of modifying Flexpart for the new hardware in anticipation of shorter simulation times.

Previously, groups have explored Flexpart parallelisation with MPI (distributed memory) and OpenMP (shared memory) methods, finding that the most computationally-expensive components of the code are the processing of large GRIB datasets for the input of meteorology, and the computation of new particle positions at each timestep.

In this project, we aim to explore the application of GPU programming to Flexpart for the first time.

From *International Monitoring System (CTBTO)*

From *NVIDIA Tesla V100 GPU Architecture*

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.126.02   Driver Version: 418.126.02   CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:06:00.0 Off |                    0 |
| N/A   28C    P0    44W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  Off  | 00000000:07:00.0 Off |                    0 |
| N/A   29C    P0    44W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  Off  | 00000000:0A:00.0 Off |                    0 |
| N/A   29C    P0    42W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  Off  | 00000000:0B:00.0 Off |                    0 |
| N/A   26C    P0    42W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Tesla V100-SXM2...  Off  | 00000000:85:00.0 Off |                    0 |
| N/A   28C    P0    44W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Tesla V100-SXM2...  Off  | 00000000:86:00.0 Off |                    0 |
| N/A   29C    P0    44W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   6  Tesla V100-SXM2...  Off  | 00000000:89:00.0 Off |                    0 |
| N/A   29C    P0    43W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   7  Tesla V100-SXM2...  Off  | 00000000:8A:00.0 Off |                    0 |
| N/A   27C    P0    43W / 300W |      0MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```


Please do not use this space, a QR code will be automatically overlayed

Objectives

This is an exploratory venture intended to utilise newly-installed GPU hardware for the performance enhancement of the complex atmospheric transport model, Flexpart.

Several groups have previously implemented parallelism in Flexpart using MPI for distributed processing and OpenMP for a shared-memory paradigm. Through this work it has been recognized that the most logical components for Flexpart parallelisation are the ingest and processing of meteorological data files, and the computation of new particle positions through the timestepping process.

Before jumping into the GPU paradigm, it needs to be understood that this is simply another parallel paradigm, and as such, performance enhancement is limited – more than we often think – by the code that is inherently sequential.

Still, in a saturated operational environment, even "some" performance improvement will lead to the ability to run more workload.

The primary objectives for this work are to

- establish a functional GPU-based development and testing environment on the new CTBTO system
- use the current CTBTO Flexpart code to establish a baseline for model output and performance, serving as a control for subsequent development and testing
- conduct a performance analysis of the current CTBTO Flexpart code to find the computational "hot spots," and determine which part of the code will be targeted for GPU optimization
- create a development / test mockup environment of the targeted code to facilitate rapid and iterative experimentation of promising methods
- implement, test and assess the GPU-optimised code

INTRODUCTION
OBJECTIVES
METHODS/DATA
RESULTS
CONCLUSION

Methods/data (1 of 2)

Gfortran compilation and execution of the original Flexpart-CTBTO was performed, creating a baseline by which to compare results and performance of the future GPU-instrumented code.

Installation of full NVIDIA HPC SDK in user space was surprisingly straightforward, but in order for it to work correctly, compatible low-level NVIDIA libs must be installed at root-level. This was all subsequently tested on a basic linear algebra code in serial, CUDA and OpenACC to ensure a correct development environment.

Initial attempts at compilation with the NVIDIA HPC nvfortran were problematic with both minor and major issues needing to be resolved. One issue – limiting the size of the problems we could handle - required a bug report to NVIDIA with a subsequent fix in the next release of the compiler. An additional problem is currently being considered for a bug report.

Numerical and visual comparison of the gfortran vs nvfortran codes was performed with satisfactory results.

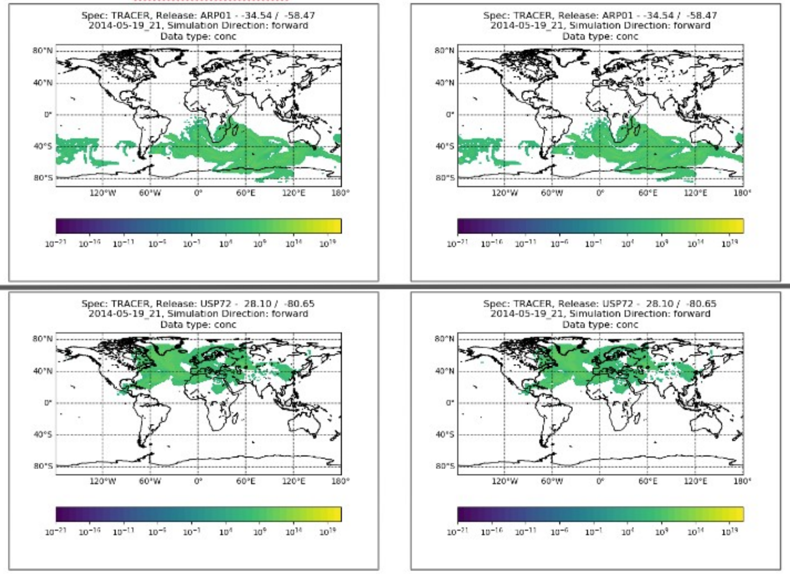
```
integer :: idummy = -7
real    :: settling = 0.
```

```
$ nvfortran -c     -acc  -Minfo=acc    aplay2.f90
NVFORTRAN-W-0155-Compiler failed to translate accelerator region (see -Minfo
messages): No device symbol for address reference (aplay2.f90: 1)
advance:
      1, Generating acc routine seq
         Generating NVIDIA GPU code
NVFORTRAN-F-0704-Compilation aborted due to previous errors. (aplay2.f90)
```

```
integer :: idummy
real    :: settling
.
.
.
idummy = -7
settling = 0.
```

An example of a common Fortran construct that nvfortran sometimes rejects

gfortran vs nvfortran, end of 10-day fwd simulation



Please do not use this space, a QR code will be automatically overlayed

Methods/data (2 of 2)

Profiling of the code with custom and NVIDIA-supplied tools suggested – with little surprise – that the most promising component for GPU-isation was the primary particle computation loop.

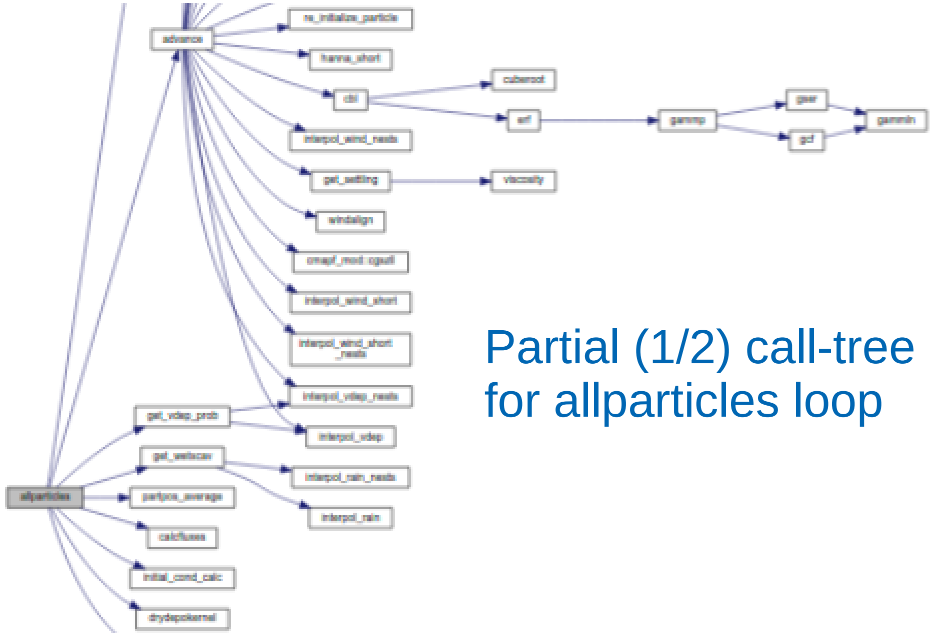
In a perfect world, all iterations would be independent of each other, and a simple OpenACC directive would accomplish our parallelisation with five minutes of effort.

However, in Flexpart, this code has a deep hierarchy of subroutine and function calls, and they are all highly-dependent on each other through their use of over 200 global variables defined in various Fortran modules.

```
.
.
.
!$ACC kernels
    do j=1,numpart


  ! If integration step is due, do it
  !***********************************

      if (itra1(j).eq.itime) then
.
.
.
     endif

    end do !loop over particles
!$ACC end kernels
.
.
.
```

24 hours, 4 releases, 10M particles

ECMWF, 0.5 degree	GFS, 0.5-degree
TOTAL SECONDS: 1134.01	TOTAL SECONDS: 513.73
------------------------------	------------------------------
gridcheck_cum_secs: 47.19 (4.2%)	gridcheck_cum_secs: 4.46 (0.9%)
timemanager_cum_secs: 1084.65 (95.6%)	timemanager_cum_secs: 507.39 (98.8%)
----- Reading/processing met data ----	----- Reading/processing met data ----
getfields_cum_secs: 683.61 (60.3%)	getfields_cum_secs: 157.98 (30.8%)
readwind_cum_secs: 501.19 (44.2%)	readwind_cum_secs: 88.15 (17.2%)
verttransform_cum_secs: 54.49 (4.8%)	verttransform_cum_secs: 27.45 (5.3%)
calcpar_cum_secs: 91.08 (8.0%)	calcpar_cum_secs: 26.86 (5.2%)
calcpv_cum_secs: 64.43 (5.7%)	calcpv_cum_secs: 15.43 (3.0%)
----- Particle computations ----	----- Particle computations ----
allparticles_cum_secs: 349.27 (30.8%)	allparticles_cum_secs: 306.09 (59.6%)
conccalc_cum_secs: 49.79 (4.4%)	conccalc_cum_secs: 41.54 (8.1%)



Partial (1/2) call-tree for allparticles loop

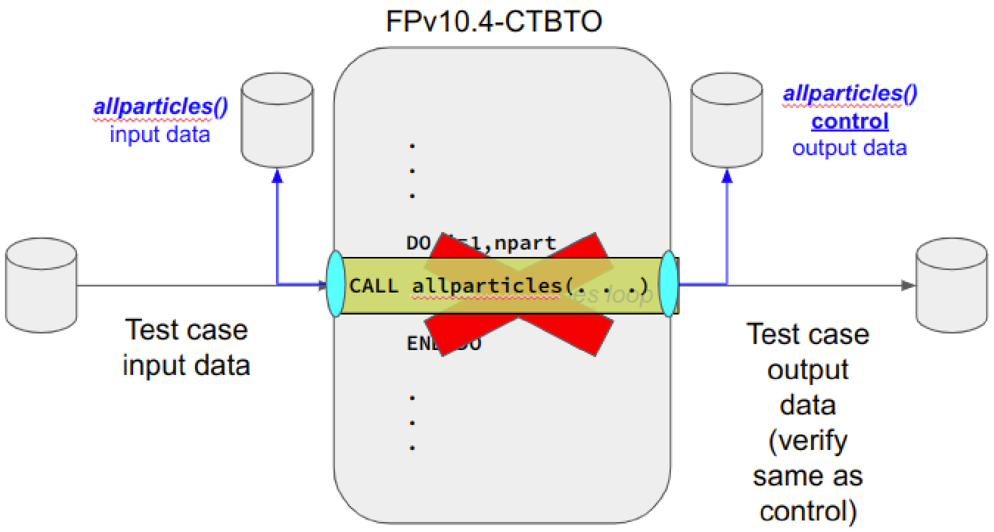
Please do not use this space, a QR code will be automatically overlayed

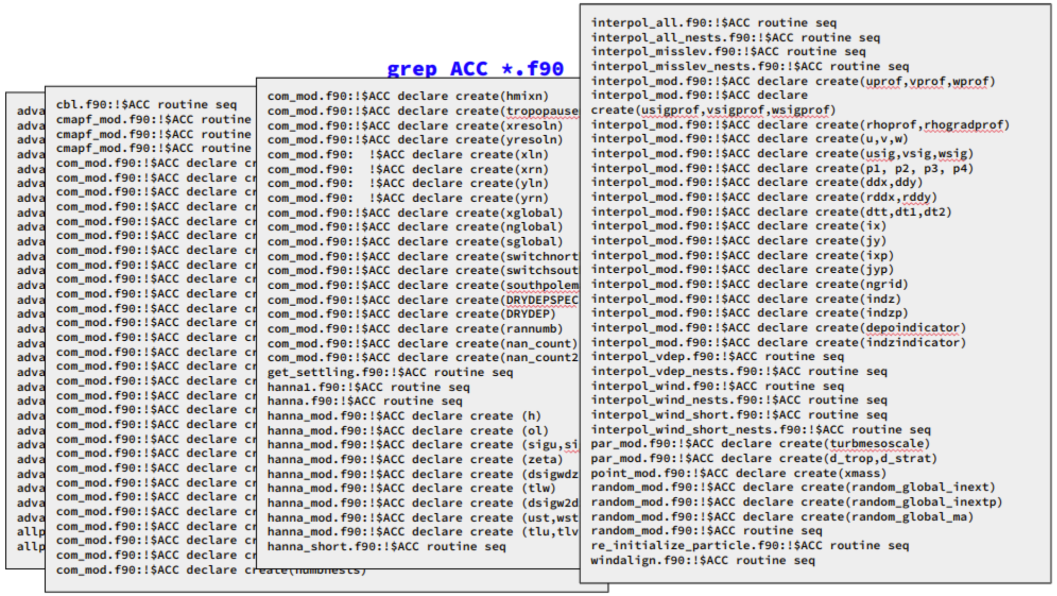
Results

Knowing that this work would require huge amounts of experimentation and iteration, it was deemed essential to extract the allparticles loop into a simpler, fast-executing standalone test program, initialised by a dump of intermediate values from a full Flexpart run.

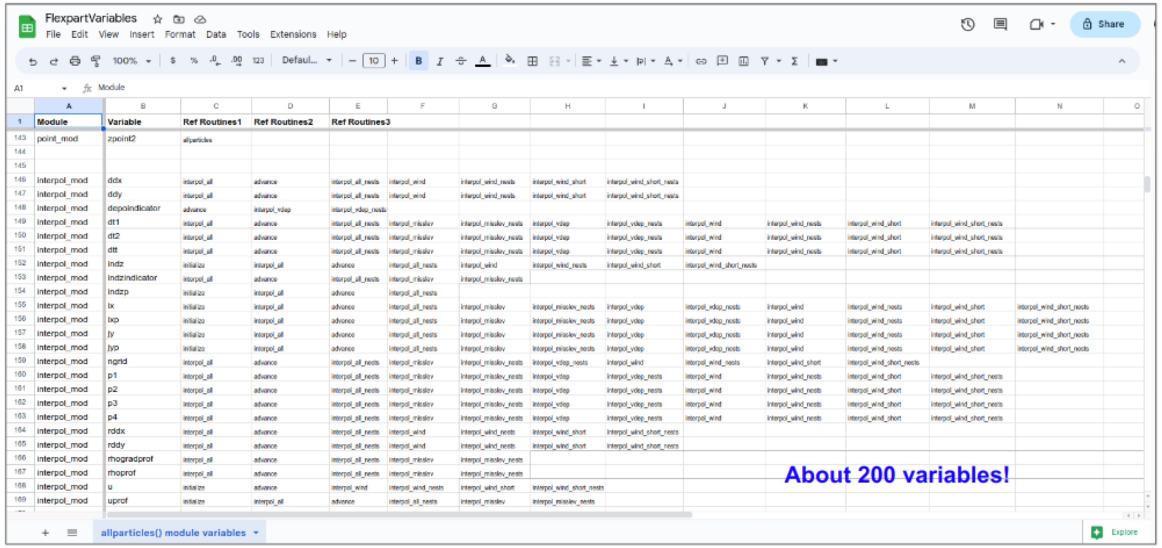
The application of OpenACC directives – just to get it to compile and execute – has been quite complicated and tedious, requiring many directives, and the creation of a spreadsheet to maintain an inventory of all 200+ global module variables referenced in the underlying code.

Current status is that compilation (with the discovery of another nvfortran compiler bug) and execution of the full particles loop has been accomplished. It's doubtful that computations are correct or efficient, and this is what needs to be explored next.






```
interpol_all.f90:!$ACC routine seq
interpol_all_nests.f90:!$ACC routine seq
interpol_misslev.f90:!$ACC routine seq
interpol_misslev_nests.f90:!$ACC routine seq
interpol_mod.f90:!$ACC declare create(uprof,vprof,wprof)
interpol_mod.f90:!$ACC declare
create(usigprof,vsigprof,wsigprof)
interpol_mod.f90:!$ACC declare create(rhoprof,rhogradprof)
interpol_mod.f90:!$ACC declare create(u,v,w)
interpol_mod.f90:!$ACC declare create(usig,vsig,wsig)
interpol_mod.f90:!$ACC declare create(p1, p2, p3, p4)
interpol_mod.f90:!$ACC declare create(ddx,ddy)
interpol_mod.f90:!$ACC declare create(rddx,rddy)
interpol_mod.f90:!$ACC declare create(dtt,dt1,dt2)
interpol_mod.f90:!$ACC declare create(ix)
interpol_mod.f90:!$ACC declare create(jy)
interpol_mod.f90:!$ACC declare create(ixp)
interpol_mod.f90:!$ACC declare create(jyp)
interpol_mod.f90:!$ACC declare create(ngrid)
interpol_mod.f90:!$ACC declare create(indz)
interpol_mod.f90:!$ACC declare create(indzp)
interpol_mod.f90:!$ACC declare create(depoindicator)
interpol_mod.f90:!$ACC declare create(indzindicator)
interpol_vdep.f90:!$ACC routine seq
interpol_vdep_nests.f90:!$ACC routine seq
interpol_wind.f90:!$ACC routine seq
interpol_wind_nests.f90:!$ACC routine seq
interpol_wind_short.f90:!$ACC routine seq
interpol_wind_short_nests.f90:!$ACC routine seq
par_mod.f90:!$ACC declare create(turbmesoscale)
par_mod.f90:!$ACC declare create(d_trop,d_strat)
point_mod.f90:!$ACC declare create(xmass)
random_mod.f90:!$ACC declare create(random_global_inext)
random_mod.f90:!$ACC declare create(random_global_inextp)
random_mod.f90:!$ACC declare create(random_global_ma)
re_initialize_particle.f90:!$ACC routine seq
windalign.f90:!$ACC routine seq
```

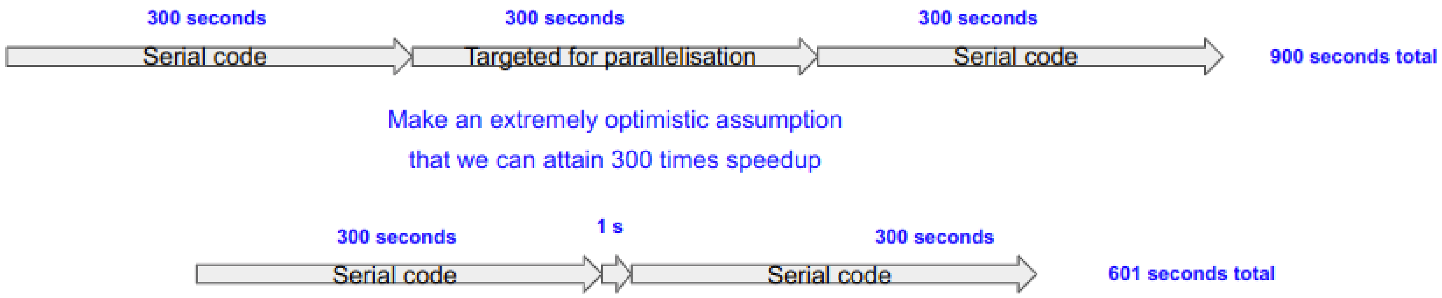


Conclusions

Boreal
Scientific Computing LLC

We knew this was going to be difficult for many reasons, and we knew that even upon completion, Amdahl's Law would significantly limit overall speedups that could be obtained.

Example of Amdahl's Law as applied to a typical, complex legacy code



However, given the popularity of Flexpart and its long legacy for 25 years, and its complexity, we see it as an opportunity to explore and report on – for better and worse – issues related to parallelisation in general, and GPU-isation specifically

This continues to be a work in progress. Upcoming activities include

- Evaluate – and revise current test code where necessary – the correctness of outputs and the performance of the new code
- Time permitting, experiment with cleaner allparticles loops, where iterations are truly independent (requires immense code refactoring)
- Apply what we have learned to the evolving Flexpart 11 in order to assess the value of continued efforts in this area

INTRODUCTION
OBJECTIVES
METHODS/DATA
RESULTS
CONCLUSION