CEF Telecom Project No INEA/CEF/ICT/A2017/1566782

TRAFAIR: Understanding traffic flows to improve air quality

INEA CEF-TELECOM Project

Agreement No INEA/CEF/ICT/A2017/1566782

GRAL - Parallelization manual

Code	GPG
Date	04/2020
Type	
Responsible	CESGA
Participants	CESGA
Authors	Cecilia Grela, Javier Cacheiro, Markus Kuntner, Oettl Dietmar
Corresponding Authors	Javier Cacheiro
Version	1.0

Co-financed by the Connecting Europe Facility of the European Union

Executive Summary

In order to be able to use GRAL for TRAFAIR its execution time had to be reduced, for this the parallelization of GRAL (code and execution) was needed. GRAL creators (http://lampz.tugraz.at/~gral/) parallelized the code (using C# "Parallel.For" loop parallelization) and TRAFAIR partners tested several strategies on how to parallelize GRAL..

The objective of this manual is to provide general guidelines about how to speed-up GRAL based on the results of a set of comprehensive benchmarks created, executed and analysed by CESGA and performed using FinisTerrae II supercomputer resources at CESGA.

Parallelization of GRAL

Strategies

The strategies analyzed were the following:

A. **Pre-computation of the flow field files (GFF).**
B. **Emission Sources Splitting.**
C. **Spatial Domain Splitting.**
D. **Launch different GRAL execution for different heights.**

The strategies C and D had to be discarded because they produce wrong results just as expected from a theoretical point of view:

- Spatial Domain Splitting: splitting the domain and launching a GRAL execution for each part reduced significatively the time needed for the simulation but the results were not valid as the particles that will move through the domain will not be reflected in the results.
- Launch a different GRAL execution for different heights: the same happens with this strategy: when lower heights are simulated the particles from lower heights will not move to higher levels during the simulations. And it will be very difficult to know how many particles and were they should be when simulating higher heights.

A combination of emission sources splitting and flow field pre-computation showed to be the best approach.

Pre-computation of the flow fields (GFF)

To understand why this is useful, let's review the steps followed in a GRAL simulation:

- Step 1: calculation of the prognostic wind field around or through obstacles (buildings, vegetation – wind flow through the vegetation area, microscale terrain): this is realized in a simple CFD model. A further parallelization in a HPC is difficult, because of the underlying physics. However, in practical applications, one computes classified wind field one times, saves the wind fields (*.gff files), searches the best matching wind field for the expected meteo-situation and re-uses these wind fields. The calculation of the wind fields can be a pre-processing and for the usage on the HPC you need to select the best matching wind field file and share it with the HPC nodes.
- Step 2 + 3: compute the dispersion of the pollutants using the wind fields: in step 2 we calculate the dispersion of transient particles, that are particles still available from earlier time steps and in step 3 new released particles are computed. These loops are parallelized in Program.cs using parallel.for() loops. The traces of the particles are stored in Program.Depo_conz (deposition on the ground), Program.conz_sum (concentration), Program.conz3dp and Program.conz3dm (concentration for odour in layers above and below .conz_sum), Program.conz3d (concentration for transient particles) and some additional arrays to store the concentration temporarily.

Step 1 takes most of the time of a simulation, to avoid this situation wind fields can be precomputed and stored in .gff files to be reused in future simulations. More GFF could be added to the library in the future if needed.

Steps 2 and 3 can be shared on a HPC for one time step (usually ½ or 1 hour). The result arrays have to be combined and the process begins at step 1.

To precompute the flow fields we should take into account the two modes that affect the dispersion step of GRAL:

- **Steady-state mode**: GFF files are always precomputated. If a library of GFF is used but GRAL is launched in steady-state the GFF files will be rewritten. *This mode is used to pre-compute GFF*. In in.dat: steady state=1
- **Transient mode**: each line in "mettimeseries.dat" needs a corresponding line in "meteopgt.all"; the files must contain the same meteorological data for each line. In in.dat:single case=0

To pre-compute GFF

To pre-compute GFF this steps should be taken:

1. Generate a meteopgt.all: a file with a line for each GFF to be pre-computed with the meteorological data that identified it. To decide the number of meteorological conditions to pre-compute the environmental experts should take into account the creation of a stability matrix adapted to their city and use it to generate the meteopgt.all as its content depends totally on meteorological characteristics of the city. Each meteopgt.all line contains:

   a. Wind direction sector,Wind speed class,stability class, frequency
2. Configure GRALusing in.dat
   a. Set steady state=1
      i. 0 !steady state=1,single case=0
   b. Set the number of particles to 1:
      i. 1 !Particlenumber
   c. Set the dispersion time to 500 (minimum value)
      i. 500 !Dispersion time
3. Launch a GRAL execution.
4. Generate a folder for the library.
5. Copy all the gff files in the library folder.
6. Generate a GFF_FilePath.txt with the path to the library: 1st line for a path for Windows (empty if not needed) and 2nd line for a path for Unix (empty if not needed).

To pre-compute GFF: parallelized

The number of GFF to pre-computate is usually high for that reason parallelizing this process is the best option, for that generating GFF files in batches is the best. The number of GFF on each batch to be executed in parallel depends on the computational resources that will be used by the city to precompute GFF.

CESGA recommends the precomputation of GFF in batches, so that each batch can be run in a different computing node. It is also important no to dedicate more than 6-12 cores to each computation (more cores are not used efficiently). In our case with compute nodes of 24 cores, we found the optimal was to run 2 GFF computations simultaneously in the same compute node. The batches are run in parallel in different computing nodes.

To pre-compute GFF in batches the following steps were followed:

1. Generate a directory for each batch to be launched with all the files needed to execute GRAL.
2. Generate a meteopgt.all with all the lines and copy it in the library folder.
3. Divide meteopgt.all in as many files as batches will be launched. Copy each file in its respective folder.
4. Configure GRALusing in.dat. and copy it in all folders.
   a. Set steady state=1
      i. 0 !steady state=1,single case=0
   b. Set the number of particles to 1:
      i. 1 !Particlenumber
   c. Set the dispersion time to 500 (minimum value)
      i. 500 !Dispersion time
5. Launch all GRAL executions.
6. Generate a folder for the library.
7. Copy all the gff files in the library folder: rename each one with the number of the line in meteopgt.all that corresponds to that gff.

8. Copy meteopgt.all in the library folder.
9. Generate a GFF_FilePath.txt with the path to the library: 1st line for a path for Windows (empty if not needed) and 2nd line for a path for Unix (empty if not needed). This file should be included in the simulations folder when launching GRAL using a GFF library.

GFF Pre-computation Optimizations recommendations

- Number of CPU cores (Max_Proc.txt): Best option will be 12 cores, the reason is that over 12 cores the speed-up will improve very little. If the HPC environment allows it in a node with 24 cores two executions of GRAL could be executed in parallel. Next best option will be 24 cores, one node in exclusive. This option should only be taken into account if the problem is too big (in terms of time, memory, etc.) to be executed in half a node.
- Integration Time including Steady-state (Integrationtime.txt): As seen in the figures, the best value for Integration Time (and the one suggested by GRAL programmers) is a maximum of 500. The reason is that convergence goes under 1 when reached that number of iterations but the time spent in all the process will increase significatively if the maximum is set for more than 500.
- Vertical Stretching Factors (GRAL.geb): The value suggested by GRAL programmers has a good performance but this parameter could only be set by the environmental experts of each city taking into account the availability of their computation resources.
    - 2,1.00,30,1.05,50,1.15,100,1.25,200,1.5 !cell-size for cartesian wind field in GRAL in z-direction, streching factor for increasing cells heights with height

After pre-computing GFF

After pre-computing all GFF needed we'll have a library of precomputed GFF files with the following:

- All the pre-computed GFF files for a specific city (also called pool of GFF). Each GFF is named with the number of its meteorological data line in meteopgt.all
- meteopgt.all: a file with a line for each GFF pre-computed with the meteorological data that identified it. This file should be copied (or linked) in each folder for the daily operative.
- GFF_FilePath.txt: a file with the path to the GFF pool. This file should be copied (or linked) in each folder for the daily operative. This file should only contain two lines:
    - 1st line is for the path in Windows: only needed if executing GRAL in Windows. This line could be empty but should exist.
    - 2nd line is for the path in Linux systems (like Red Hat, etc.): only needed if executing GRAL in Linux systems. This line could be empty but should exist.

When using a GFF library, GRAL should be executed in transient mode (flag to be set in in.dat file: single case=0). Then it will work in the following:

1. It takes the 1st meteo situation from the file mettimeseries.dat (meteorological data to be simulated).
2. Searches the corresponding line in the meteopgt.all file (each line correspond to one GFF) and reads it. If not available it will calculate a new *.GFF file with the number of the line in meteopgt.all as name.

Emission Sources splitting

The emission sources can be splitted and then computed in parallel. The use of this strategy makes possible to use both the intra-node "Parallel.For" parallelization and the inter-node emission sources splitting parallelization to speed-up GRAL.

This technique combined together with the GFF precomputation is the best approach in terms of parallelization and it is especially useful when you use a large number of particles (>2000) because in this case the dispersion computation time increases.

line.dat file contains all emission sources, if all sources are in the same group GRAL will analyze each one after the other and if the number of sources is high (big cities will have a high number of sources) this will increase the time for the execution.

The line.dat file is one of the most important for the simulation as each line is an emission source. GRAL will distribute all the particles (this variable is setted in in.dat) between all sources and then analyze them consecutively (a high number of particles will affect the time needed for the simulation).

GRAL has an issue with line.dat: if all emission sources belong to the same group it could lead to locking and all the simulation will slow down significatively as can be seen in the following table.

The best solution to locking is to divide all sources in line.dat into several groups as homogeneously as possible: all groups must have more or less the same amount of emissions in total. Then GRAL will divide the particles between the groups based on some internal algorithms and then execute the simulation for each group in parallel and this process will reduce significatively the time needed for the simulation.

If this approach is adopted, then the following issues must be taken into account:

- The separation of the emissions by groups has to be balanced, in such a way that the sum of all emissions in all groups is more or less the same. Then each group can be computed in approximately the same time.

- Concentration files are binary, so they cannot be merged directly unless this task and read/write operations are programmed ex profeso. Due to this reason, specific code to merge the binary files has been developed (see the GRAL/conmapmergerC++ folder in the TRAFAIR GitLab repository). If ASCII files are requested (this variable is setted inside in.dat file) this issue is not a problem as this type of files could be merged with python code as shown in aggregate_concentrations.py

- GRAL will distribute all particles used for the simulation (this variable is setted inside in.dat) between all groups, this should be taken into account when deciding the number of particles.

To use particle source splitting

1. Analyze all sources in line.dat and add a group number to each source makin groups as homogeneous as possible.
2. Add a line to GRAL.greb with the number of all groups. For example:
    a. 1,2,3,4,5,6,7,8,9,10,11, !Source groups to be computed separated by a comma
3. Create an emissions_timeseries.txt files with the modulations for each group.

Execution of GRAL: emission sources splitting and GFF pre-computation

The previous two strategies can be combined to obtain the best results. It is also recommend to review "Annex I: Dispersion Computation Optimizations".

To parallelize GRAL using particle source splitting and GFF pre-computation all previous points should be taken into account and all the files needed should be prepared.

Expected Speed-ups

In the following table we give a summary of the maximum speed-up expected depending on the number of particles used in the simulation and the maximum recommended source groups splits for that given number of particles. It is not recommended to go below 100 particles per split because the simulation will be really inefficient.

Particle Number of the sequential calculation	Number of source groups splits to use	Speed-up
500	5	1.8
1000	10	2.6
2000	20	4.6
3000	30	6.4
4000	40	8.2
5000	50	10.1
6000	60	12.0

Table 1: Expected speed-ups

As you can see in Table 1, in the case that we are simulating 6000 particles, then the best speed-up would be to divide the emission sources into 60 splits and run each of them independently using 100 particles. In this case we would expect to reduce the time of the calculation in a factor of 12.

Results

The previous approaches were tested for Santiago de Compostela inside the TRAFAIR project. We summarize the results in Table 2.

Santiago de Compostela

Strategy used	Time (hours)
Normal execution (sequential execution)	10.35
Using pre-computed GFF (sequential execution)	6.41
Split sources (parallel execution)	1.32

In all cases 2000 particles were used. If more particles are used, as recommended by the GRAL developers for large domains, then the simulation times will increase and in this case the need for the split sources parallel approach will be obvious.

Table 2: Results for Santiago de Compostela using the different approaches

ANNEX I: Dispersion Computation Optimizations

Apart from the parallelization strategies discussed, it is also possible to optimize the dispersion computation, that reflects in an overall improvement the overall GRAL performance.

- **Number of CPU cores used(Max_Proc.txt)**: the recommendations for this value should be read in conjunction with the ones for "Number of particles": Best option will be 12 cores, this configuration will allow two executions of GRAL in one node. Next best option will be 24 cores, one node in exclusive. This option should only be taken into account if the problem is too big (in terms of time, memory, etc.) to be executed in half a node.

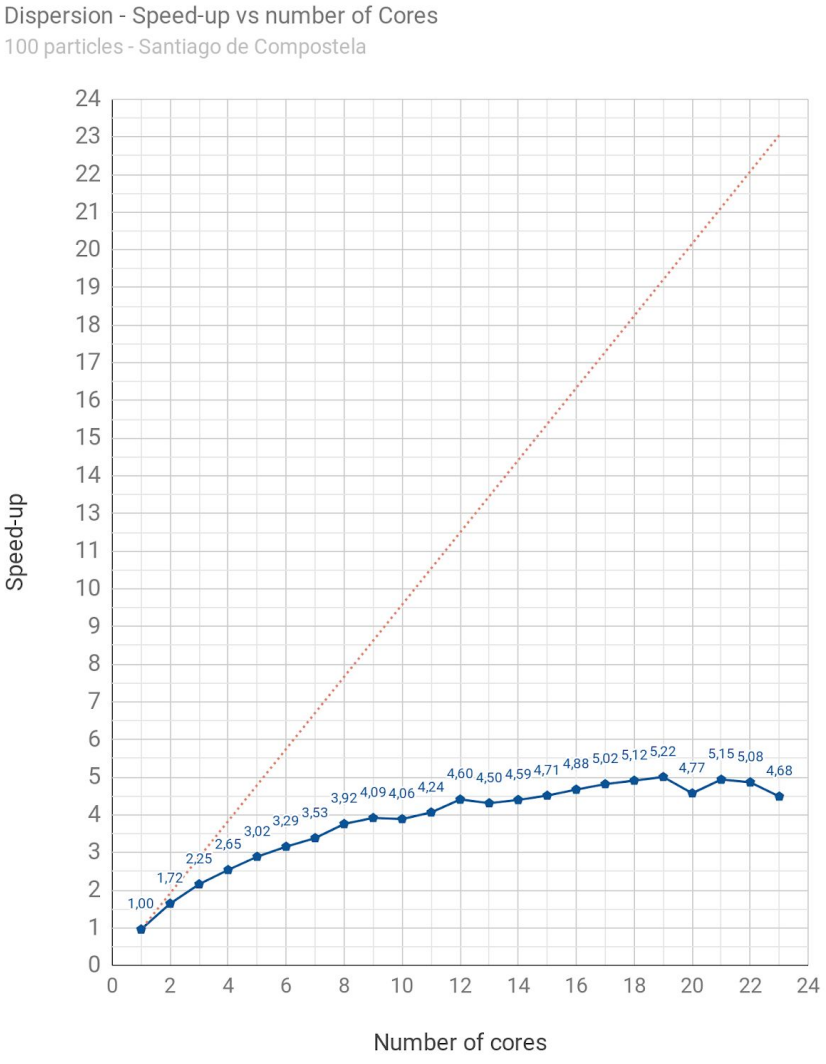

Figure: Speed-up with respect to number of cores for Santiago de Compostela using pre-computed GFF and 100 particles.

- **Number of Particles (in.dat)**: Taking all of the studies into account the recommendations is 2000 particles from the point of view of time and efficiency when using a node (or even half a node) when executing a simulation in GRAL but this number should be decided by environmental experts.
    - Should be taken into account that the number of particles affects significatively the time of the execution but will give higher resolution maps.
    - If split sources approach is adopted it should be taken into account that GRAL will distribute all particles used for the simulation between all groups.

- **Vertical Stretching Factors (GRAL.geb)**: the recommendations should be the same as the ones for GFF Precomputation.

- **Result File Compression (in.dat)**: The set for this parameter is recommended to be compression v2 or uncompressed if there is no need of saving space.

- **Concentration Grid**: It is interesting to note that in several GRAL examples in the documentation they use 4m horizontal, 0.5m vertical extension and 1m above ground. This parameter depends on the size of the problem, the resolution wanted for the results and other parameters and should be decided between the environmental experts and the IT experts taking into account the resources available.

- **Building roughness**: Each city should decide the value for this parameter.

- **The surface roughness length (in.dat)** has an influence on the dispersion of the particles, not on the computing time. In a city, 0,5 m to 2 m are recommended, significantly higher value is recommended if the city has very tall buildings that stand very close together.

Execution of GRAL

CESGA's HPC environment Finis Terrae II uses Slurm (https://slurm.schedmd.com/documentation.html) as workload manager and this script is the one used to launch the jobs:

run.sh

```
#!/bin/bash
#SBATCH -J Santiago-YYYY-MM-DD   # Job name
#SBATCH -o gral.o%j       # Name of stdout output file(%j expands to jobId)
#SBATCH -e gral.e%j       # Name of stderr output file(%j expands to jobId)
#SBATCH -p thinnodes,cola-corta        # partition
#SBATCH -N 1              # Number of nodes, not cores (24 cores/node)
#SBATCH -n 1              # Total number of MPI tasks (if omitted, n=N)
#SBATCH -c 24             # Cores per task
#SBATCH -t 02:00:00       # Run time (hh:mm:ss)

module load gral/V20.01
echo $SLURM_CPUS_PER_TASK > Max_Proc.txt
time gral
```

Configuration files

in.dat

```
100   !Particlenumber

3600        !Dispersion time

0     !steady state=1,single case=0

4     !Meteorology:
inputzr.dat=0,meteo.all=1,elimaeki.prn=2,SONIC.dat=3,meteopgt.all=4

1     !Receptor points: Yes=1

0.2   !Surface Roughness in [m]

47    !Latitude

N     !Meandering Effect Off=J/On=N

NOx   !Pollutant: NOx,CO,PM,HC

3.5,        !Horizontal slices [m] seperated by a comma (number of slices has
to be defined in GRAL.geb!)

3     !vertical grid spacing in [m]

1     !Start calculation with weather number...

2,15        !How to take buildings into account? 1=simple mass conservation,
2=mass conservation with Poisson equation + advection, Factor for prognostic
sub domains

0     !Stream Output for SOUNDPLAN 1=activated 0=standard ASCII output

compressed        ! write compressed output files

NoKeyStroke       ! do not wait for a keystroke when exiting GRAL

ASCiiResults 1    ! write additional ASCii result files
```

GRAL.geb

```
4                !cell-size for cartesian wind field in GRAL in x-direction

4                !cell-size for cartesian wind field in GRAL in y-direction

2,1.00,30,1.05,50,1.15,100,1.25,200,1.5              !cell-size for cartesian wind field in GRAL in z-direction, streching factor for increasing cells heights with height

1388             !number of cells for counting grid in GRAL in x-direction

1326             !number of cells for counting grid in GRAL in y-direction

1                !Number of horizontal slices

1,2,3,4,5,6,7,8,9,10,11,                !Source groups to be computed seperated by a comma

534680                  !West border of GRAL model domain [m]

540232                  !East border of GRAL model domain [m]

4744944                 !South border of GRAL model domain [m]

4750248                 !North border of GRAL model domain [m]
```

Sum concentrations

aggregate_concentrations.py

```
#!/usr/bin/env python2


from __future__ import print_function, division

import numpy as np

import glob


def ascii_header(filename):

	"""Retrieve ESRI ASCII raster format header"""

	with open(filename) as raster:

	header = []

	for _ in range(6):

		header.append(raster.readline().strip())

	return '\n'.join(header)


def read_concentrations(filename):

	"""Read concentrations from filename and return a numpy array"""

	with open(filename) as out:

	rows = out.readlines()[6:]

	data = []

	for row in rows:

	data.append(row.split())


	concentrations = np.array(data, dtype='float64')
```

```
        return concentrations


def write_concentrations(header, concentrations, filename):
        """Write the contentrations to an ascii file using the given header"""
        np.savetxt(filename, concentrations, delimiter=' ',
                header=header, comments='')


def computed_groups():
        """Returns a list with the numbers of the source groups computed"""
        return [int(filename[7:9]) for filename in glob.glob('00001-1*.txt')]


def computed_hours():
        """Returns the number of hours computed"""
        return len(glob.glob('000*-101.txt'))


def main():
        # Number of hours simulated
        hours = computed_hours()
        # List of source groups used for the calculation
        groups = computed_groups()
        # All concentration output files will share the same header
        # FIXME: We assume there will always be a source group named 1
        header = ascii_header('00001-101.txt')


        for hour in range(1, hours + 1):
        print('Aggregating concentrations for simulation hour {}'.format(hour))
```

```
        # We read the concentrations of first group for the given hour
        total = read_concentrations('000{:02d}-1{:02d}.txt'
                                    .format(hour, groups[0]))
        # And then we accumulate the rest of groups generating the total
        for group in groups[1:]:
            total += read_concentrations('000{:02d}-1{:02d}.txt'
                                         .format(hour, group))
        write_concentrations(header, total, '000{:02d}.total.txt'.format(hour))


if __name__ == '__main__':
        main()
```

Glossary

- **GRAL.geb**: stores basic information about the GRAL grids and the model domain (see GRAL documentation for more info about this file).
- **in.dat**: this file stores the main parameters to run GRAL, such as the number of particles to be released per second, the dispersion time, etc. (see GRAL documentation for more info about this file).
- **Integrationtime.txt**: (optional) used to set the minimum and maximum number of iterations for the prognostic microscale flow field model algorithm used by GRAL. By default the values are: 0, 500.
- **Max_Proc.txt**: maximum numbers of threads when GRAL is executed using parallel computing. It can be considered equivalent to the number of CPU used.
- **meteopgt.all**: stores meteorological data. It could be equal to mettimeseries.dat (in terms of data, not structure) or be used to describe all the precomputated GFF in a GFF library (see GRAL documentation for more info about this file).
- **mettimeseries.dat**: series of meteorological data used in the simulations. During daily operative all the cities will fill this file with the daily meteorological info that will be simulated that day. Each line of this file stores a meteorological condition organized in columns (separated by commas) like this:
    - day.month, hour, Wind speed, wind direction, stability class
- **line.dat**: includes all line sources for the simulation (see GRAL documentation for more info about this file). Each line stores emission sources organized in columns (separated by commas) like this:
    - StrName,Section,Sourcegroup,x1,y1,z1,x2,y2,z2,width,noiseabatementwall,Length[km],--,NOx[kg/(km*h)],--,--,--,--,--,deposition data
- **GRAL_FlowFields.txt**: see GRAL documentation for more info about this file.
- **GFF_FilePath.txt**: stores the path to the GFF library.
- **Steady-state mode**: Computation of steady-state concentration fields: In this case particles are tracked until they leave the model domain regardless the time they need to do so. There is no dependence of concentrations on the selected dispersion time.
- **Transient mode**: Computation of concentrations fields, which are dependent on the averaging time chosen: In this case particles are only tracked until the dispersion time is elapsed. Moreover, the last particle's position is rendered into a three-dimensional concentration field, which is stored for

the following weather situation, where this concentration field is again transformed into a particle mass. For each grid element of the three-dimensional concentration field one single particle is released. The concentration grid is based on the Cartesian grid used for the microscale flow-field simulations in the horizontal direction. In vertical direction it uses the height of the first grid cell of the flow-field grid, but has an independent vertical stretching algorithm, which is not adjustable by the user. The grid itself is terrain-following. All these secondary particles share the same properties with regard to deposition and sedimentation velocities for each user defined source group. Emissions can be modulated for each weather situation and source group using the input file “emissions_timeseries.txt”.