BFM Profiling Report

Esteban Damián Gutiérrez Mlot

July 21, 2014

Contents

Chapter 1

Introduction

This report will use different profiling tools to analyze the performance of the Biochemical Flux Model (*BFM*)[1] in standalone and coupled mode.

The objective of this report is to detect parts of the code susceptibles of improvement. Executing different tests (see Chapter 3) and using a serie of common profiling tools (see Section 2.2.1), performance will be evaluated measurements of memory usage and/or execution time.

BFM source code can be downloaded freely from the BFM website: `http://www.bfm-community.eu`.

Chapter 2

Technical Info

2.1 Hardware

Configurations will run on the cluster named "Athena". This is a IBM iDataPlex dx 360 m4 cluster. The idataplex cluster at CMCC is an Intel Xeon (Sandy Bridge E5-2670) system with the following characteristics:

- 8-core socket @ 2.6 GHz (with 3.0/3.3 turbo modes)
  - 32KB 8-way L1 cache, instruction + data per core
  - 256kb L2 cache per core
  - 20 MB L3 cache (shared)
  - 20.8 GFlops peak performance per core
  - Support for 2 HyperThreading instances per core, 32 hyperthreads per node
- 2 sockets per node, 16 core per node
- 4 memory controllers/socket
- 51.2 GB/s memory bandwidth per socket
- 64 GB memory/node (4GB/core), DDR3 1600MHz
- 14x Infiniband FDR10 interconnect on PCIe3 (40 Gb/s = 5 GB/s unidirectional b/w)

For an extensive information about hardware specification and architecture you can see the "PELAGOS Activity Report"[2]

2.2 Software

Profiling in information science, refers to the “process of construction and application of profiles generated by computerized data analysis”. Application of profiles will let us

to measure the performance of applications. Profiling combined with rerunning previously completed tests and checking whether program behavior has changed (regression test) serve as a guidance to detection of newly introduced bugs. This early detection of errors is fundamental to assure the reliability of programs output, as a BFM climate predictions results. With the objective of automatize the profiling process of BFM, the tool *BFMtest* has been created.

BFMtest

BFMtest is the tool for testing configurations in BFM. This tool automatize the process of compiling, running and analyzing the different existing presets of BFM. Similar to the configuration tool for BFM (bfm_configure), the configuration of a series of tests is based on presets and it is programmed using PERL[3] language. The analysis is carried out by other sub-tools (Valgrind[4] and nccmp[5]) and the outputs of these programs are inspected and summarized to show a complete report of tests.

Similarly to *bfm_configure*, *BFMtest* proceeding is divided in different phases. In total there are three phases:

Generation: Generate the test in the output directory. For that this phase will perform the generation, compilation and deployment steps of *bfm_configure* with the default configuration modified by the particular options of the *BFMtest* preset.

Run: Perform the execution of the test in the output directory. This phase includes waiting for the process to finish the run (also if is a batch job)

Analysis: Build the summary of the generation and run. It will collect if the previous phases were successful, show run-time and status of the optional diagnostics (Valgrind and comparison)

The execution of one preset will entails the execution of one or several tests (*bfm_configure* presets). The cycle of execution is, for each test performs the selected phases (Generations, Run and Analysis) consecutively and continue to next one. Then each test is treated independently, even if they are in the same *BFMtest* preset. See image 2.1. The command to run the tool with an specific preset is:

```
bfm_configure.sh -p PRESET_NAME
```

You can also use the verbose mode for getting more information:

```
bfm_configure.sh -vp PRESET_NAME
```

The following table summarize available options. For a up-to-date options table check the "EXAMPLE" file in the configuration directories.

NAME: Name of the test (mandatory)

ACTIVE: Specify execution stage to perform. Valid values are (you can combine them):

- **Y**: all execution stages: generation, running and analysis (default)
- **N**: no execution. Test will not be generated, run or analyzed.
- **G**: only generation
- **R**: only running
- **A**: only analysis

PRESET: Name of the preset for using as parameter in *bfm_configure* (see *bfm_configure* option '-p')

COPY: Copy other generated test

- In generation time, The test will not be generated, only copied and deployed (*bfm_configure* option '-d')
- Source test must exists in tmp directory (use ACTIVE=G to only execute generation of source test)

RUN: Mode to run the experiment. Valid values are:

- **sh**: default, for running in a unix shell
- **bsub**: for running in batch mode

RUNEXE: Specify prototype for generating the experiment running script (see *bfm_configure* option '-R')

PROC: Number of processors to use in compilation and running (see *bfm_configure* option '-r')

ARCH: Specify compilation Architecture file (see *bfm_configure* option '-a')

QUEUE: Specify queue in job mode (see *bfm_configure* option '-q')

PAREXE: Command to execute in parallel. Ex: openmpirun (see *bfm_configure* option '-e')

- You can use $PROC inside this command

PRECMD: Commands to execute prior compiling and prior running

PRERUN: Commands to execute prior running

FORCING: Forcing files to link in the run directory (see *bfm_configure* option '-i')

VALGRIND: Valgrind command and options for analysis

COMPARE: Directory with test output to compare results

- Run command "nccmp -V" first to check if nccmp command works

COMPTHR: Compare threshold for nccmp command (default 1e-3)

All the options follow these rules for specification of values:

- Options are separated by lines, and values by coma ’,’
- Write an option in multiple lines using backslash ’\’ at the end of the line
- Use two single quotes ’’ to specify empty values.

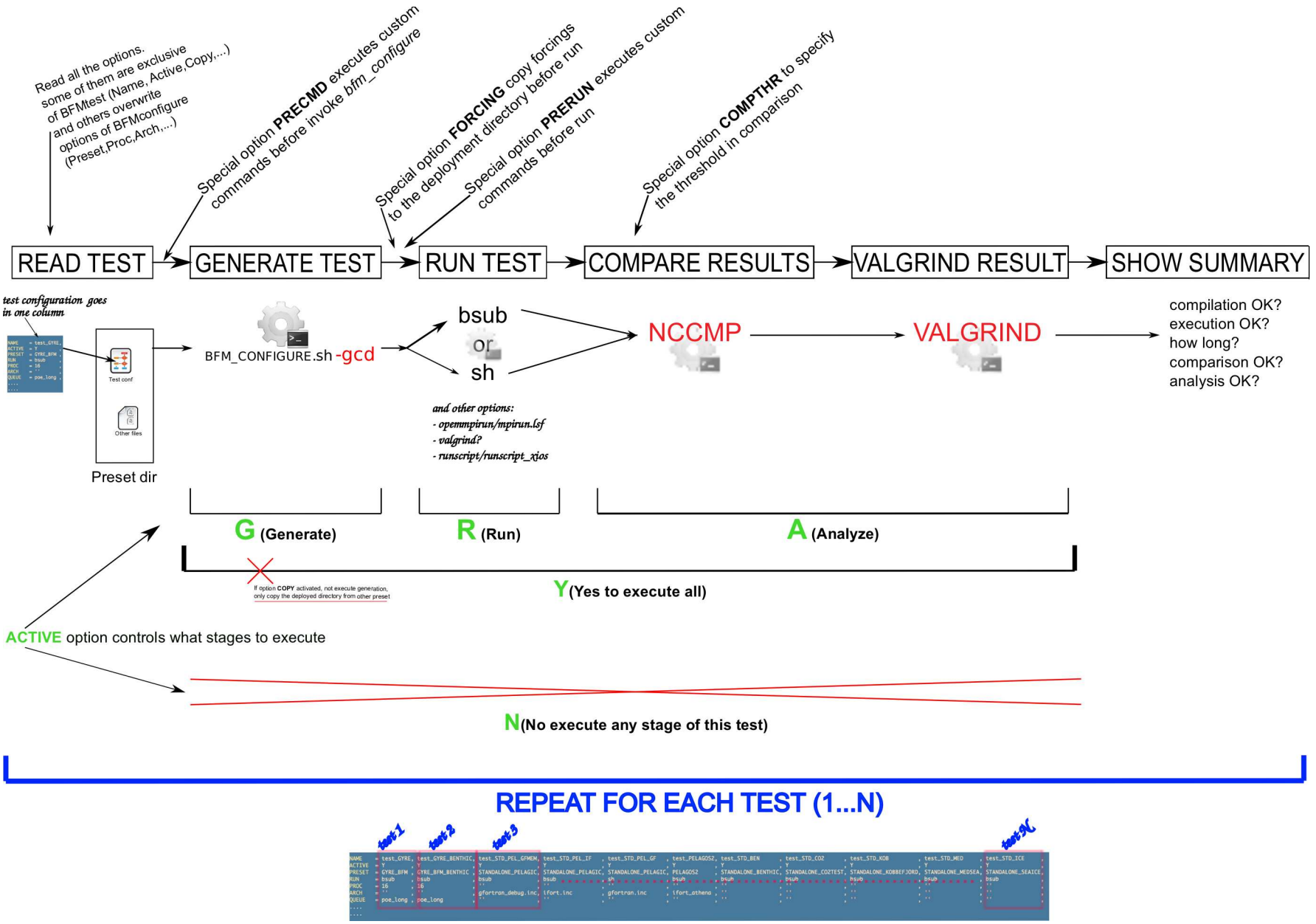

Figure 2.1: BFMtest flow diagram

2.2.1 Tools

NCCMP

Nccmp compares two NetCDF files bitwise or with a user defined tolerance (absolute or relative percentage). Comparisons are done in local memory without requiring temporary files. This tool is very useful for running regression test of scientific models.

As an example you can execute the command:

```
nccmp -dgmf NETCDF_FILE1 NETCDF_FILE2
```

This command will compare the data (-d), global attributes (-g), metadata (-m) and will execute the comparison until the end (-f).

To download and learn how to use it go the tool online web page[5]).

Valgrind

Valgrind is an strumentation frameworks for building analysis tools. Valgrind distribution contains useful quality tools. The tools used in this report are these:

- **Callgrind**[6] is a profiling tool that records the call history among functions in a program's run as a call-graph. By default, the collected data consists of the number of instructions executed, their relationship to source lines, the caller/callee relationship between functions, and the numbers of such calls. Optionally, cache simulation and/or branch prediction (similar to Cachegrind) can produce further information about the runtime behavior of an application.
- **Massif**[7] is a heap profiler. It measures how much heap memory your program uses. This includes both the useful space, and the extra bytes allocated for bookkeeping and alignment purposes.

For more information of these tools, go to the online web page [4].

Other useful tools related with Valgrind but developed independiently are:

- **KCachegrind**[8] Callgrind output visualizer.
- **Massif-Visualizer**[9] Massif output visualizer.

Chapter 3

Tests

3.1 Standalone Pelagic

3.1.1 Configuration

IFORT

OPTION	VALUE
NAME	test_STD_PEL
ACTIVE	A
PRESET	STANDALONE_PELAGIC
RUN	bsub
PROC	”
ARCH	ifort.inc
RUNEXE	”
PAREXE	”
QUEUE	”
PRECMD	module load HDF5/hdf5-1.8.11_parallel NETCDF/netcdf-4.3_parallel NETCDF/parallel-netcdf-1.3.1
FORCING	”
PRERUN	”
COMPARE	”
VALGRIND	”

GFORTRAN

OPTION	VALUE
NAME	test_STD_PEL_C
ACTIVE	A
PRESET	STANDALONE_PELAGIC
RUN	bsub
PROC	”
ARCH	gfortran_debug.inc
RUNEXE	”
PAREXE	”
QUEUE	”
PRECMD	module load HDF5/hdf5-1.8.11_parallel NETCDF/netcdf-4.3_parallel NETCDF/parallel-netcdf-1.3.1
FORCING	”
PRERUN	”
COMPARE	”
VALGRIND	”

VALGRIND

OPTION	VALUE
NAME	test_STD_PEL_M
ACTIVE	A
PRESET	STANDALONE_PELAGIC
RUN	bsub
PROC	”
ARCH	gfortran_debug.inc
RUNEXE	”
PAREXE	”
QUEUE	”
PRECMD	export NETCDF=/users/home/ans040/local; export LD_LIBRARY_PATH=/users/home/ans040/local/lib:$LD_LIBRARY_PATH
FORCING	”
PRERUN	”
COMPARE	’/users/home/ans040/dev/repo/bfm/tools/bfm_test/tmp/test_STD_PEL’
VALGRIND	valgrind –tool=massif –run-libc-freeres=no –stacks=yes

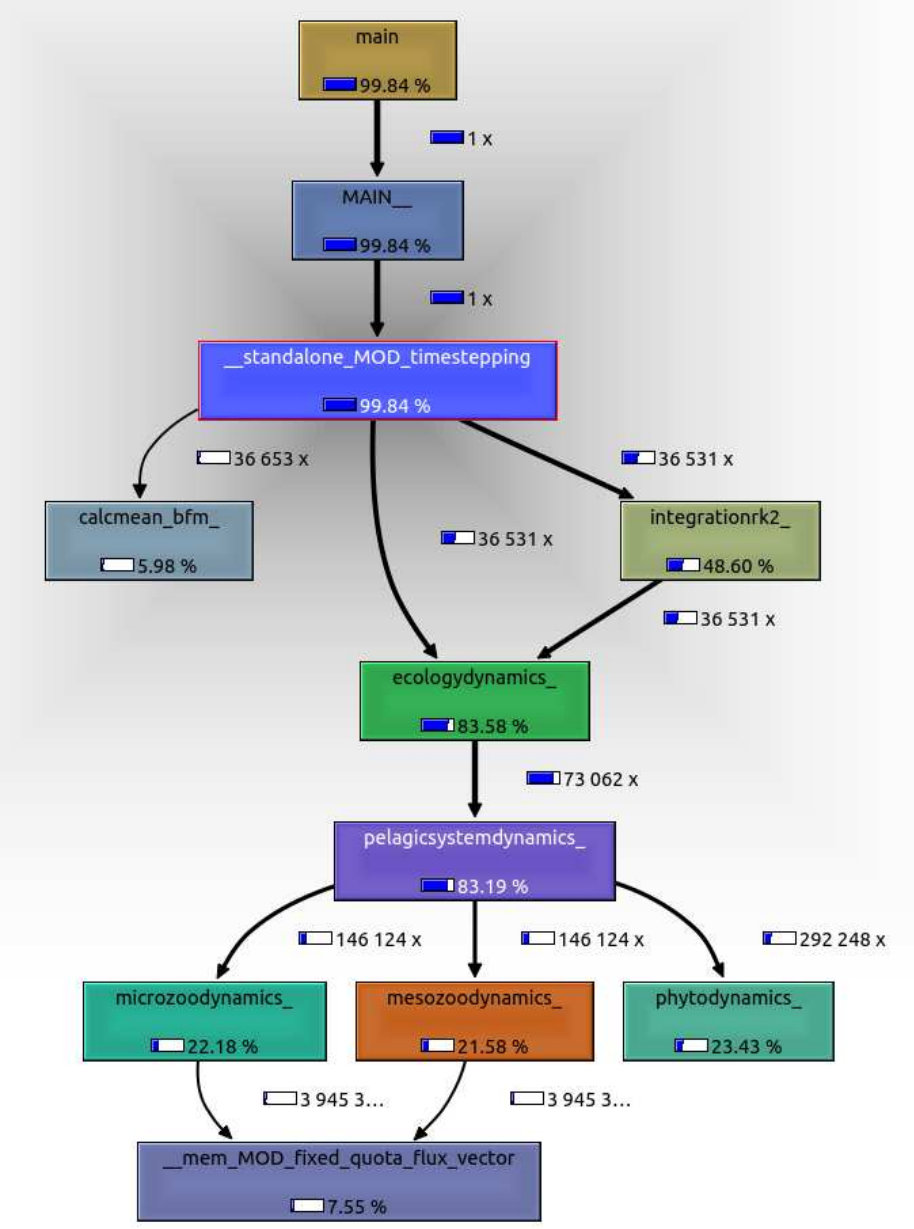

Figure 3.1: Standalone Pelagic Call Graph

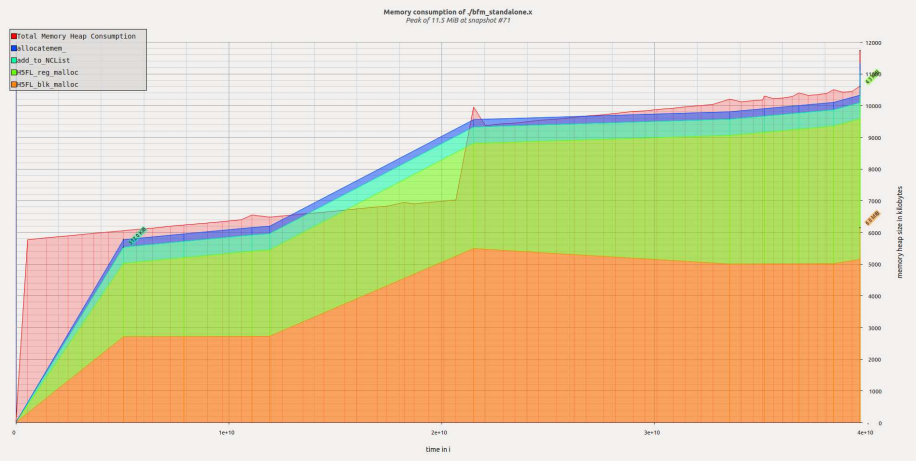

Figure 3.2: Standalone Pelagic Memory Use Graph

3.1.2 Result

3.2 Gyre

3.2.1 Configuration

IFORT

OPTION	VALUE
NAME	test_GYRE_BFM
ACTIVE	A
PRESET	GYRE_BFM
RUN	bsub
PROC	”
ARCH	Ifort_athena_xios
RUNEXE	”
PAREXE	”
QUEUE	’poe_long’
PRECMD	module load HDF5/hdf5-1.8.11_parallel NETCDF/netcdf-4.3_parallel NETCDF/parallel-netcdf-1.3.1
FORCING	/work/ans040/FORCING/NEMO/*;/work/ans040/FORCING/BFM/*
PRERUN	cp $BFMDIR/tools/bfm_test/configurations/PROFILING_TESTS/pelagos_cfg ./namelist_cfg
COMPARE	”
VALGRIND	”

GFORTRAN

OPTION	VALUE
NAME	test_GYRE_BFM_CI
ACTIVE	A
PRESET	GYRE_BFM
RUN	bsub
PROC	”
ARCH	ifort_athena_xios_debug
RUNEXE	”
PAREXE	”
QUEUE	’poe_long’
PRECMD	module load HDF5/hdf5-1.8.11_parallel NETCDF/netcdf-4.3_parallel NETCDF/parallel-netcdf-1.3.1
FORCING	”
PRERUN	”
COMPARE	’/users/home/ans040/dev/repo/bfm/tools/bfm_test/tmp/test_GYRE_BFM’
VALGRIND	valgrind –tool=callgrind –collect-jumps=yes –dump-instr=yes

VALGRIND

OPTION	VALUE IFORT	VALUE GFORTRAN
NAME	test_GYRE_BFM_MI	test_GYRE_BFM_MG
ACTIVE	A	A
PRESET	GYRE_BFM	GYRE_BFM
RUN	bsub	bsub
PROC	”	”
ARCH	ifort_athena_xios_debug	gfortran_athena_xios_debug
RUNEXE	”	”
PAREXE	”	”
QUEUE	’poe_long’	’poe_long’
PRECMD	[1]	[2]
FORCING	”	”
PRERUN	”	”
COMPARE	[3]	[4]
VALGRIND	[5]	[6]

3.2.2 Result

[1] module load HDF5/hdf5-1.8.11_parallel NETCDF/netcdf-4.3_parallel NETCDF/parallel-netcdf-1.3.1
[2] export NETCDF=/users/home/ans040/local; export LD_LIBRARY_PATH=/users/home/ans040/local/lib:$LD_LIBRARY_PATH
[3] ’/users/home/ans040/dev/repo/bfm/tools/bfm_test/tmp/test_GYRE_BFM’
[4] ’/users/home/ans040/dev/repo/bfm/tools/bfm_test/tmp/test_GYRE_BFM’
[5] valgrind –tool=massif –stacks=yes
[6] valgrind –tool=massif –run-libc-freeres=no –stacks=yes

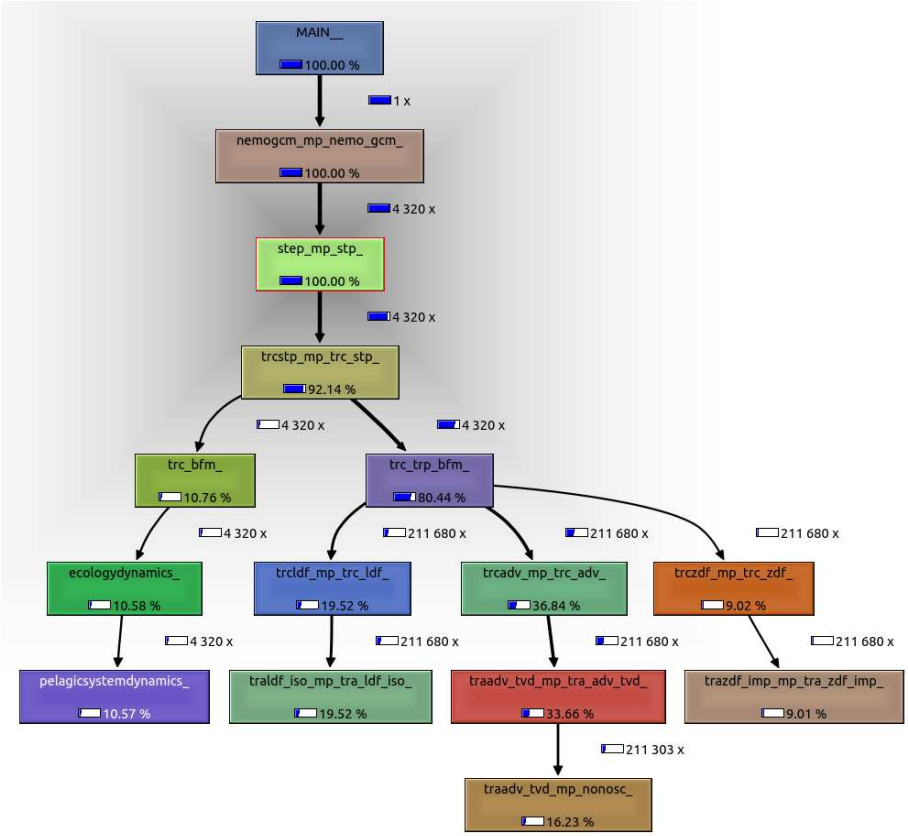

Figure 3.3: Gyre Call Graph

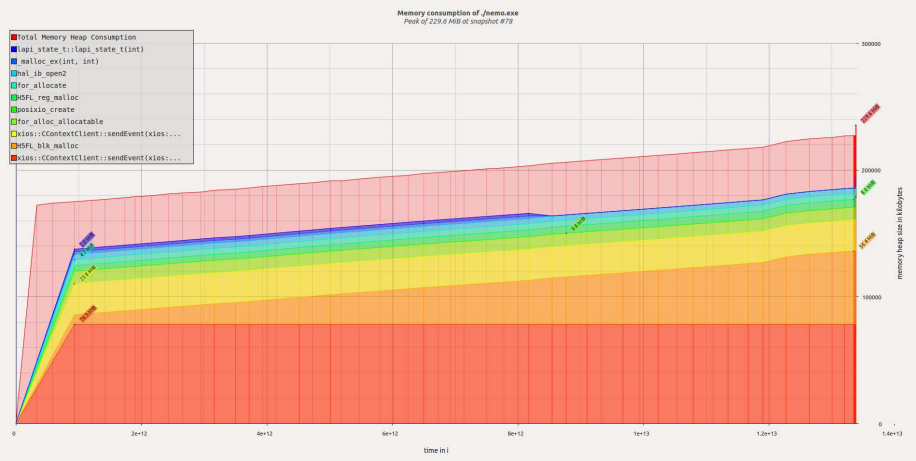

Figure 3.4: Gyre with Ifortran compiler Memory Use Graph

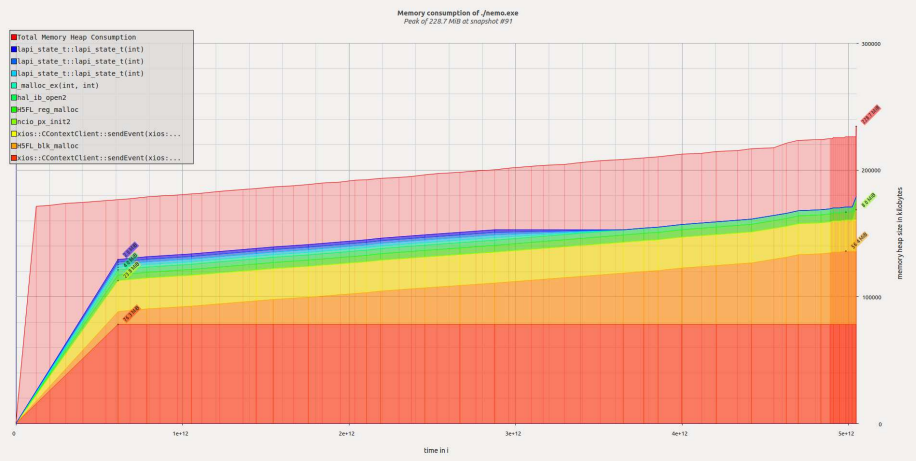

Figure 3.5: Gyre with Gfortran compiler Memory Use Graph

Bibliography

[1] V. M. et al, *The Biogeochemical Flux Model (BFM), Equation Description and User Manual*, 1st ed., 2014. [Online]. Available: http://bfm-community.eu

[2] J. Brunson, *CMCC/PELAGOS - Final Activity Report*, 2013.

[3] The perl directory - perl.org. [Online]. Available: http://www.perl.org/

[4] Valgrind: an instrumentation framework for building dynamic analysis tools. [Online]. Available: http://valgrind.org/

[5] nccmp compares two netcdf files. [Online]. Available: http://nccmp.sourceforge.net/

[6] Callgrind: a call-graph generating cache and branch prediction profiler. [Online]. Available: http://valgrind.org/docs/manual/cl-manual.html

[7] Massif: a heap profiler. [Online]. Available: http://valgrind.org/docs/manual/ms-manual.html

[8] Profile data visualization for callgrind. [Online]. Available: http://kcachegrind.sourceforge.net/html/Home.html

[9] Profile data visualization for massif. [Online]. Available: https://projects.kde.org/projects/extragear/sdk/massif-visualizer