Profiling Tutorial #2

ProfVis Setup and REGIONS

Outline:

1) Creating a BL_PROF profiling database at runtime.
2) Building an AMRVis executable that can read the database.
3) Adding REGIONS to improve your ability to filter through the database.
4) Open your database and try it out.

PROFILE

Creating a profiler database: PROFILE

AMReX Make flags:

PROFILE=TRUE

AMReX CMake flags:

AMReX_BASE_PROFILE=ON

(This inserts the compiler flag: -DBL_PROFILING)

- Provides only the most basic profiling functionality.
  - Tracks timers, but not MPI calls or the call stack.
- In most cases, want to turn on the full set of profiling options.

COMM_PROFILE & TRACE_PROFILE

AMReX Make flags:

COMM_PROFILE=TRUE

TRACE_PROFILE=TRUE

AMReX CMake flags:

AMReX_COMM_PROFILE=ON

AMReX_TRACE_PROFILE=ON

Trace

- Stores the call stack.
- Required for regions, time filtering and the all call times plot file.

Comm

- Stores information on MPI Calls.
- Required for the Timeline, Send/Recv information, etc.

★ Generally, unless fine-tuning (e.g. for a big run and you want to look a specific thing to reduce I/O), it is **good practice is turn all three on.**

If either COMM_PROFILE or TRACE_PROFILE are on, PROFILE will be turned on.

(This inserts the compiler flags: -DBL_COMM_PROFILING & -DBL_TRACE_PROFILING)

Tiny profiler vs. Full profiler

```
void
TinyProfiler::start ()
{
#ifdef _OPENMP
#pragma omp master
#endif
if (stats.empty())
{
    Real t = amrex::second();
    ttstack.push(std::make_pair(t, 0.0));

    global_depth = ttstack.size();
    for (auto const& region : regionstack)
    {
        Stats& st = statsmap[region][fname];
        ++st.depth;
        stats.push_back(&st);
    }
}
}
```

```
void BLProfiler::start() {
#ifdef _OPENMP
#pragma omp master
#endif
{
  bltelapsed = 0.0;
  bltstart = ParallelDescriptor::second();
  ++mProfStats[fname].nCalls;
  bRunning = true;
  nestedTimeStack.push(0.0);

#ifdef BL_TRACE_PROFILING
  int fnameNumber;
  std::map<std::string, int>::iterator it = BLProfiler::mFNameNumbers.find(fname);
  if(it == BLProfiler::mFNameNumbers.end()) {
    fnameNumber = BLProfiler::mFNameNumbers.size();
    BLProfiler::mFNameNumbers.insert(std::pair<std::string, int>(fname, fnameNumber));
  } else {
    fnameNumber = it->second;
  }
  ++callStackDepth;
  BL_ASSERT(vCallTrace.size() > 0);
  Real calltime(bltstart - startTime);
  vCallTrace.push_back(CallStats(callStackDepth, fnameNumber, 1, 0.0, 0.0, calltime));
  CallStats::minCallTime = std::min(CallStats::minCallTime, calltime);
  CallStats::maxCallTime = std::max(CallStats::maxCallTime, calltime);

  callIndexStack.push_back(CallStatsStack(vCallTrace.size() - 1));
  prevCallStackDepth = callStackDepth;
#endif
} }
```

How the database is written:

Output is hardwired to bl_prof.
If exists, moves old database to: “bl_prof.old.(unique#id)”.

Contains a single directory with up to three types of files:

bl_prof:	Basic Profiling
bl_comm_prof:	Comm Profiling
bl_call_stats:	Trace Profiling

Number of files written, when the data is flushed, etc. can be set similar to standard I/O. Details will be covered another week.

But: if you need to flush early, use BL_PROFILE_FLUSH().

```
kngott@gigan:/scratch/kngott/profData/bl_prof.Hyper$ ls
bl_call_stats_D_00000  bl_call_stats_H_00022  bl_comm_prof_H_00012
bl_call_stats_D_00001  bl_call_stats_H_00023  bl_comm_prof_H_00013
bl_call_stats_D_00002  bl_call_stats_H_00024  bl_comm_prof_H_00014
bl_call_stats_D_00003  bl_call_stats_H_00025  bl_comm_prof_H_00015
bl_call_stats_D_00004  bl_call_stats_H_00026  bl_comm_prof_H_00016
bl_call_stats_D_00005  bl_call_stats_H_00027  bl_comm_prof_H_00017
bl_call_stats_D_00006  bl_call_stats_H_00028  bl_comm_prof_H_00018
bl_call_stats_D_00007  bl_call_stats_H_00029  bl_comm_prof_H_00019
bl_call_stats_D_00008  bl_call_stats_H_00030  bl_comm_prof_H_00020
bl_call_stats_D_00009  bl_call_stats_H_00031  bl_comm_prof_H_00021
bl_call_stats_D_00010  bl_comm_prof_D_00000   bl_comm_prof_H_00022
bl_call_stats_D_00011  bl_comm_prof_D_00001   bl_comm_prof_H_00023
bl_call_stats_D_00012  bl_comm_prof_D_00002   bl_comm_prof_H_00024
bl_call_stats_D_00013  bl_comm_prof_D_00003   bl_comm_prof_H_00025
bl_call_stats_D_00014  bl_comm_prof_D_00004   bl_comm_prof_H_00026
bl_call_stats_D_00015  bl_comm_prof_D_00005   bl_comm_prof_H_00027
bl_call_stats_D_00016  bl_comm_prof_D_00006   bl_comm_prof_H_00028
bl_call_stats_D_00017  bl_comm_prof_D_00007   bl_comm_prof_H_00029
bl_call_stats_D_00018  bl_comm_prof_D_00008   bl_comm_prof_H_00030
bl_call_stats_D_00019  bl_comm_prof_D_00009   bl_comm_prof_H_00031
bl_call_stats_D_00020  bl_comm_prof_D_00010   bl_prof_D_00000
bl_call_stats_D_00021  bl_comm_prof_D_00011   bl_prof_D_00001
bl_call_stats_D_00022  bl_comm_prof_D_00012   bl_prof_D_00002
bl_call_stats_D_00023  bl_comm_prof_D_00013   bl_prof_D_00003
bl_call_stats_D_00024  bl_comm_prof_D_00014   bl_prof_D_00004
bl_call_stats_D_00025  bl_comm_prof_D_00015   bl_prof_D_00005
bl_call_stats_D_00026  bl_comm_prof_D_00016   bl_prof_D_00006
bl_call_stats_D_00027  bl_comm_prof_D_00017   bl_prof_D_00007
bl_call_stats_D_00028  bl_comm_prof_D_00018   bl_prof_D_00008
bl_call_stats_D_00029  bl_comm_prof_D_00019   bl_prof_D_00009
bl_call_stats_D_00030  bl_comm_prof_D_00020   bl_prof_D_00010
bl_call_stats_D_00031  bl_comm_prof_D_00021   bl_prof_D_00011
bl_call_stats_H        bl_comm_prof_D_00022   bl_prof_D_00012
bl_call_stats_H_00000  bl_comm_prof_D_00023   bl_prof_D_00013
bl_call_stats_H_00001  bl_comm_prof_D_00024   bl_prof_D_00014
bl_call_stats_H_00002  bl_comm_prof_D_00025   bl_prof_D_00015
bl_call_stats_H_00003  bl_comm_prof_D_00026   bl_prof_D_00016
bl_call_stats_H_00004  bl_comm_prof_D_00027   bl_prof_D_00017
bl_call_stats_H_00005  bl_comm_prof_D_00028   bl_prof_D_00018
bl_call_stats_H_00006  bl_comm_prof_D_00029   bl_prof_D_00019
bl_call_stats_H_00007  bl_comm_prof_D_00030   bl_prof_D_00020
bl_call_stats_H_00008  bl_comm_prof_D_00031   bl_prof_D_00021
bl_call_stats_H_00009  bl_comm_prof_H         bl_prof_D_00022
bl_call_stats_H_00010  bl_comm_prof_H_00000   bl_prof_D_00023
bl_call_stats_H_00011  bl_comm_prof_H_00001   bl_prof_D_00024
bl_call_stats_H_00012  bl_comm_prof_H_00002   bl_prof_D_00025
bl_call_stats_H_00013  bl_comm_prof_H_00003   bl_prof_D_00026
bl_call_stats_H_00014  bl_comm_prof_H_00004   bl_prof_D_00027
bl_call_stats_H_00015  bl_comm_prof_H_00005   bl_prof_D_00028
bl_call_stats_H_00016  bl_comm_prof_H_00006   bl_prof_D_00029
bl_call_stats_H_00017  bl_comm_prof_H_00007   bl_prof_D_00030
bl_call_stats_H_00018  bl_comm_prof_H_00008   bl_prof_D_00031
bl_call_stats_H_00019  bl_comm_prof_H_00009   bl_prof_H
bl_call_stats_H_00020  bl_comm_prof_H_00010
bl_call_stats_H_00021  bl_comm_prof_H_00011
```

Controlling when the bl_prof database is written

The bl_prof database will be written:

1) When amrex::Finalize() is called.
2) When the stored data becomes larger than the default flush size.
3) When BL_PROFILE_FLUSH() is called.

Whenever written, the database stores the timer information collected up to that point in the simulation. The database is viable and accessible as long as the writing step is not interrupted. (The analysis doesn't require a final time and can be written in multiple sub steps or sporadically as needed. The data is just limited to the last successful write.)

ProfVis

Building a ProfVis Executable

Pull the AMRVis repo:

```
git clone https://github.com/AMReX-Codes/Amrvis.git
```

Master branch:	Current stable branch of AMRVis.
Profvis branch:	Development of profiling features. (UAOR)

AMRVis Make Flags

DIM = 2 (Currently, all profiling info is in 2d.)

USE_PROFPARSER = TRUE (Turns on PROFILE and TRACE_PROFILE.)

PROFILE = TRUE

TRACE_PROFILE = TRUE (Recommended to add these explicitly as well.)

COMM_PROFILE = TRUE (Again, adds, doesn't restrict, so turn it on.)

USE_MPI = ?? (Probably want one of both. MPI for larger databases.)

This will still allow opening and parsing of regular plotfiles. Just adds reading of profiling databases.

Palette & Defaults

Found in the AMRVis directory: “Palette” & “amrvis.defaults”

AMRVis looks for the defaults file at:

1) ./amrvis.defaults (Priority for specific cases.)
2) /home/username/amrvis.defaults (Pick whichever of these you prefer,)
3) /home/username/.amrvis.defaults (and move amrvis.defaults there.)

To get color, set the palette entry in amrvis.defaults to point to the Palette file, e.g.:

palette ~/Amrvis/Palette

Flex and Bison

Flex and Bison are required to parse a bl_prof directory.

Should have the required version already on your computers/NERSC systems.

Flex: flex --version (On Cori: 2.5.37)

Bison: bison --version (On Cori: 2.7)

If it’s not on your system and you would like help installing it, let me know.

Set .Xdefaults and call xrdb

Save visual defaults for Xwindow that make the function list readable.

Edit the ~/.Xdefaults file on the computer you are working from (e.g. your workstation)

Add these lines (or your own, if you know X window):

```
<executable name>*Background:  #280028002800
<executable name>*Foreground:  white
<executable name>*fontList:  6x13
```

Note: You cannot use dots (.) in executable names in your .Xdefaults file.

Rename your compiled amrvis executables if they contain a dot (or create a link with 'ls').

Once added, load the new default file: > xrdb .Xdefaults

Using AMRVis with NX (NoMachine)

There is a **untested** fix for running AMRVis with NX on Cori.

Add:

DEFINES += -DAV_NX_FIX

to the AMRVis GNUmakefile.

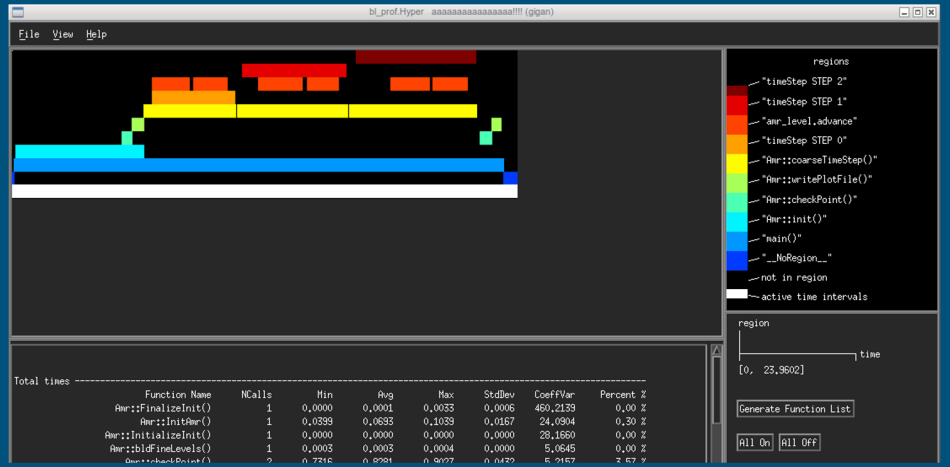
It has **not** been fully tested yet. If it fails, let me know the details and use a regular terminal with “ssh -Y”.

Checking your build

Easiest to create a link to your executable in your bin.

Will work as long as executable has same name.

That name can be used in your .Xdefaults file.



> cd ~/bin (or mkdir if you don't have one)

> ln -s <executable> <link name>

> "amrvis2d" <bl_prof_dir>/bl_prof

If your AMRVis opens the window, has color and the function list has appropriate alignment, your build is successful and complete.

BL_PROFILE_REGION

What are Regions?

- User-defined blocks of code that allow deeper filtering of profiling data.
- Everything within a region can be easily filtered into/out of your profiling analysis.
- Describes how the overview of your code is structured.

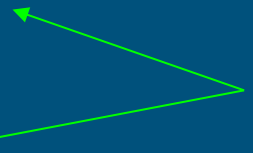
Region requirement: Each MPI rank calls each profiling region the same number of times.

Adding Regions to your profiling

BL_PROFILE_REGION_START(“Region Name String”);

… Code with (BL_PROFILE_VAR objects)

BL_PROFILE_REGION_STOP(“Region Name String”);



- Does not create a timer. Uniquely identifies all timers inside the region to allow filtering through the database.
- If you want to also time the region, place a BL_PROFILE_VAR( ) inside/outside it. (I suggest inside, so you see % of un-profiled time.)

Your First Filtering

Loading the database

Profiling directories need to start with bl_prof to identify it is a profiling database.

Can add to the directory name to make your own identifications: bl_prof.Hyper

Load a database with:

<amrvis> bl_prof

or from the GUI: File… Open.

GUI Clicks & Buttons:

Left click:	Print info on location to the AMRVis window.
	Boxing of regions on the plot.
Right click:	Add region to the selected list.
Middle click:	Remove region from the selected list.
All On, All Off:	Select/de-select all regions.
Generate Function List:	Generate a new function list of the selected regions.
Click on a Function:	Generate a rank vs. runtime for the clicked function.

For Profiling Help Contact:

Kevin Gott:

kngott@lbl.gov