Profiling Tutorial #3

Using ProfVis - GUI Features

Outline:

ProfVis GUI reference:

- All the features of the GUI and a reference for future.
- Examples of things to look out for.

ProfVis GUI Features

Looking at your results:

There are a few ways to alter your view of the data:

[Ctrl+Level #] to switch levels,
[Ctrl-D] to open the data.

- Scale (1x, 2x, 3x) (Found at: View… Scale)

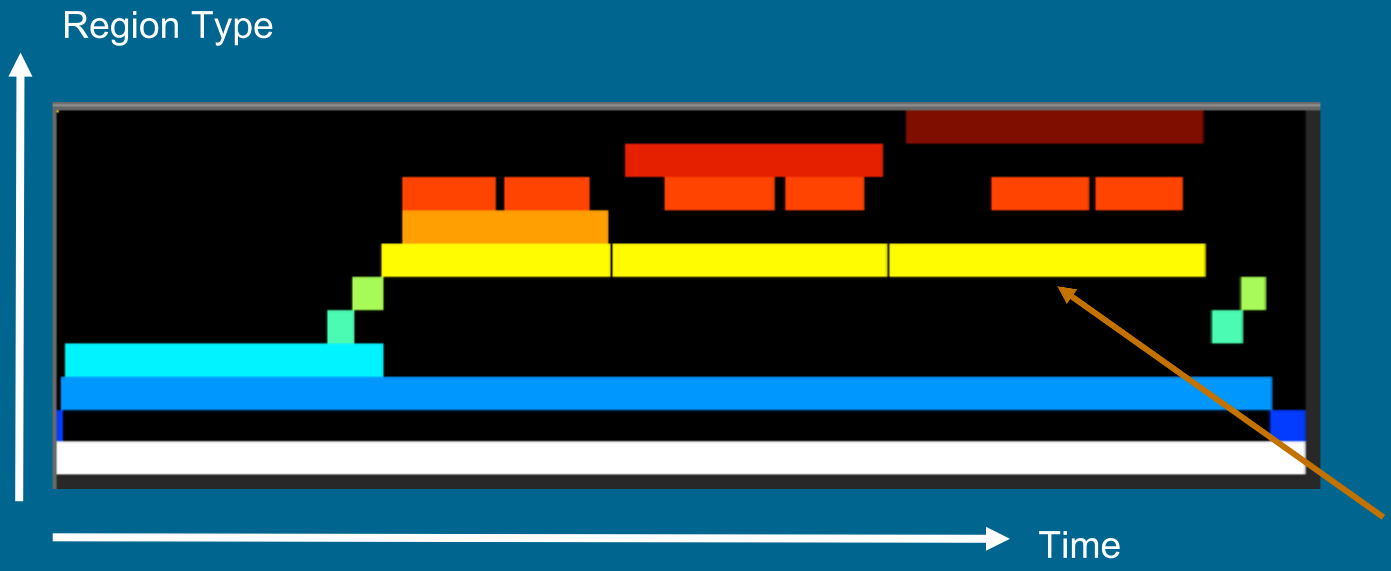
  Zoom. Especially useful when you are working with a small number of processors or a large number of quick regions over a long simulation time. “Long, skinny data set.”

- Levels - Used in Profiling PlotFiles (Found at: View… Level.)

  Profiling data has levels, same as AMRVis plotfiles.
  Can zoom into deeper levels to obtain finer results / the data set.

  For most profiling data, levels are visualized using `stride`.

ProfVis Window



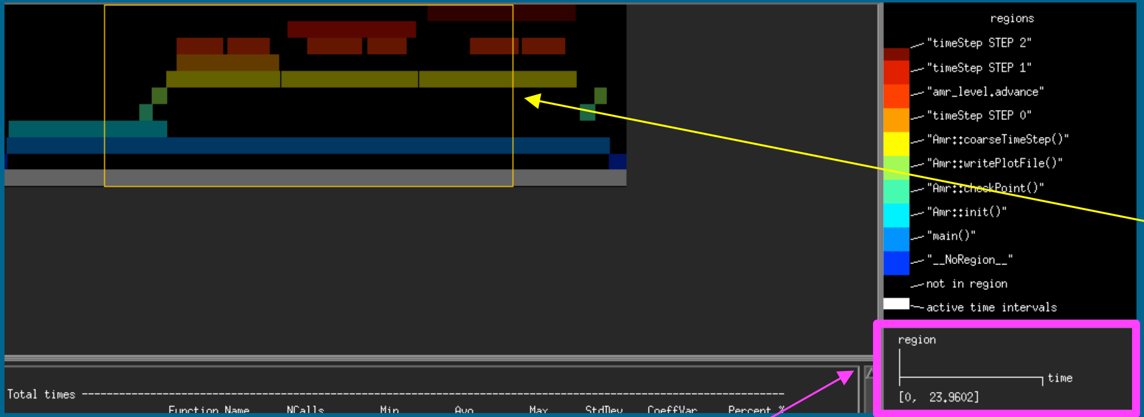
Each color corresponds to a different region.

Each region can appear multiple times, but will appear the same number of times on each rank.

Regions are identified by type and the instance of that type. (e.g. 3rd call of the 5th unique region called.)

Currently based on computational rank 0. (Will be variable in the future.)

Subregions Example: [Ctrl-S]



```
Total times -----------------------------------------------------------------------------------
             Function Name    NCalls    Min    Avg    Max    StdDev    CoeffVar    Percent %
```

Used in Profvis to filter your analysis time to a absolute minimum and maximum across all MPI ranks.

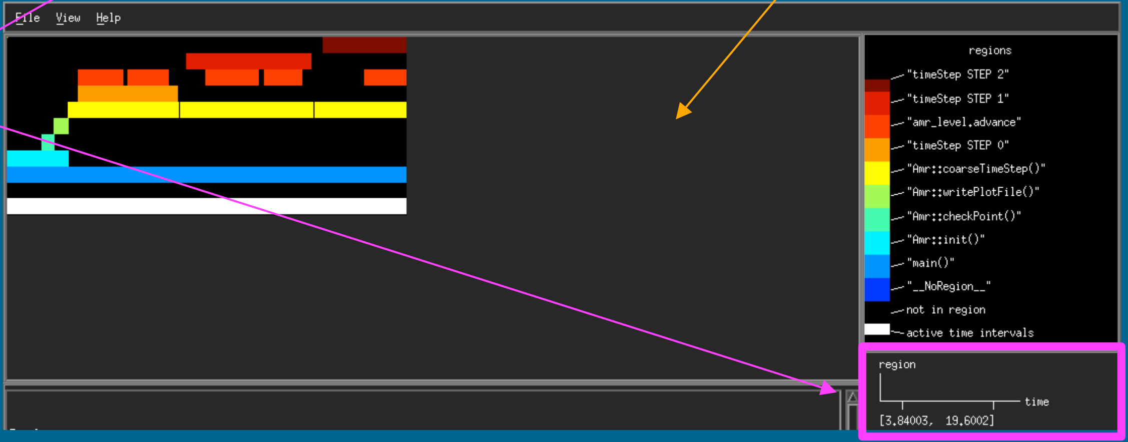
If you select this region (left click & drag), then subregion...

...you will end up with this new ProfVis window.

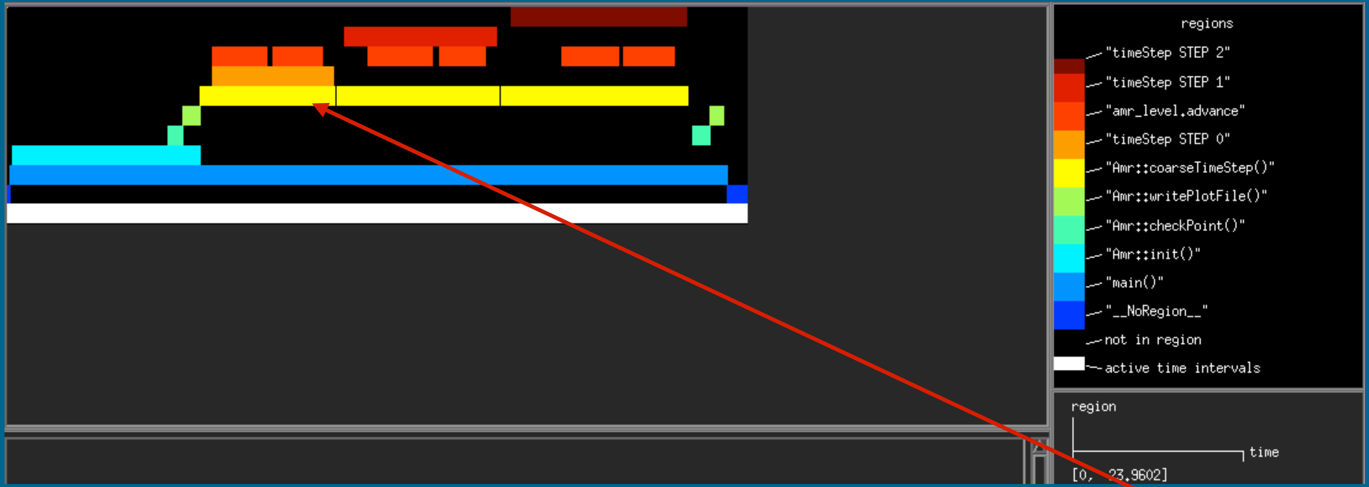
Note the 'legend' times are different.

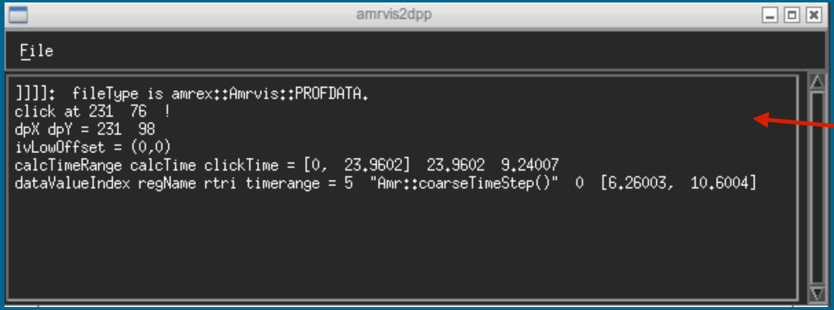
Analyses will now be limited to within this new time window for all MPI ranks.

(This is the 'highest' level parse.
Think of it as the "first" filter step.)



ProfVis Window: Select your regions





```
]]]]:  fileType is amrex::Amrvis::PROFDATA.
click at 231  76  !
dpX dpY = 231  98
ivLowOffset = (0,0)
calcTimeRange calcTime clickTime = [0,  23.9602]  23.9602  9.24007
dataValueIndex regName rtri timerange = 5  "Amr::coarseTimeStep()"  0  [6.26003,  10.6004]
```

Right Click:
Add region

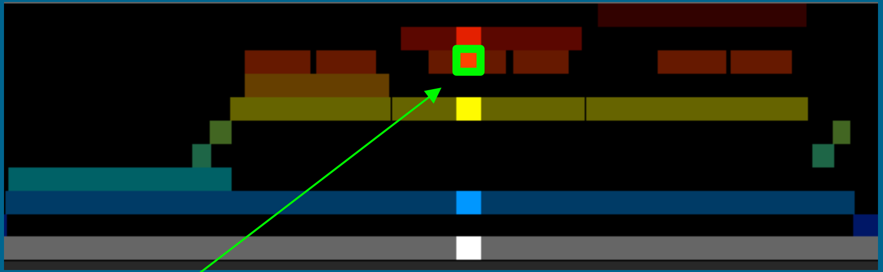
Middle Button:
Remove region

Left click and drag:
Select a time range.
(Slicing).

Left click:
Get info about the selected region in the amrvis command window.

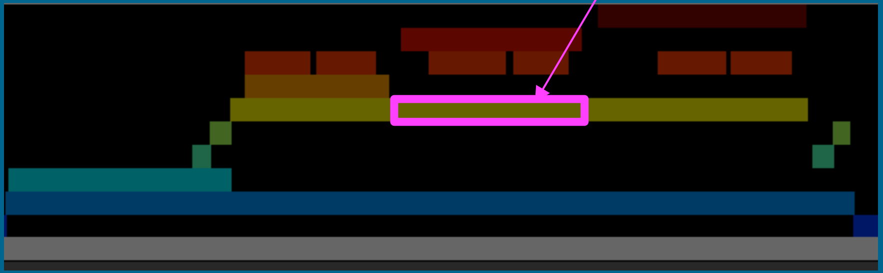
Regions:

Turning regions on and off turns them off relative to the time of the selected region, (Entire matching vertical block).



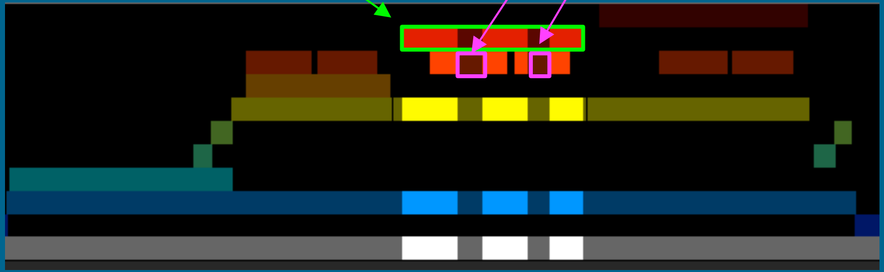
Turn this region on ….

…followed by turning this off.



Turn these regions on ….

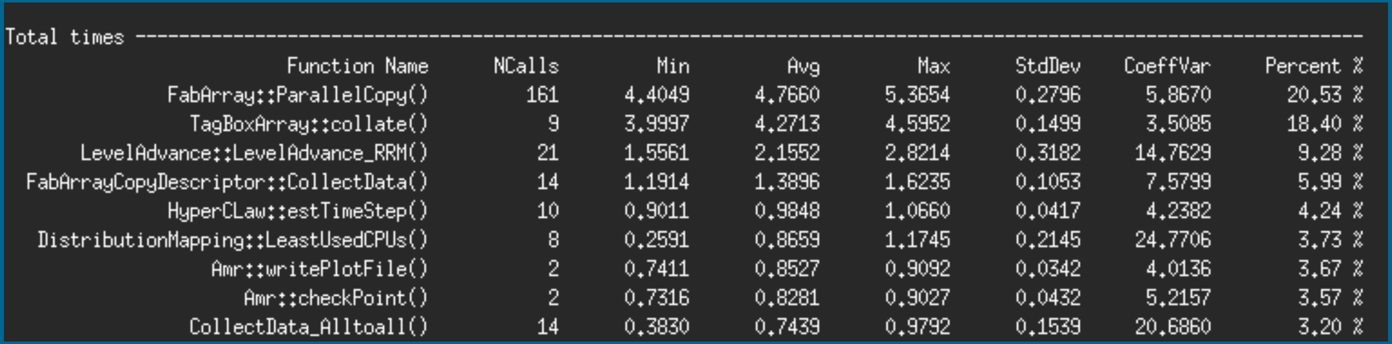
…followed by turning these off.



Although not displayed, the selected time region for each MPI rank will be based on the local time of the matching region. Can be seen in stdout when “Generate Func List” is clicked if interested. Look for:
“filterTimeRanges[rank][time block] = [##, ##]”.

The exact time for each rank **should not match**. It will include the time for the selected regions on all ranks.

Function List: Breakdown of timers



```
Total times ------------------------------------------------------------------------------------------------
                   Function Name   NCalls      Min      Avg      Max   StdDev   CoeffVar   Percent %
          FabArray::ParallelCopy()    161   4.4049   4.7660   5.3654   0.2796     5.8670    20.53 %
            TagBoxArray::collate()      9   3.9997   4.2713   4.5952   0.1499     3.5085    18.40 %
     LevelAdvance::LevelAdvance_RRM()  21   1.5561   2.1552   2.8214   0.3182    14.7629     9.28 %
FabArrayCopyDescriptor::CollectData()  14   1.1914   1.3896   1.6235   0.1053     7.5799     5.99 %
          HyperCLaw::estTimeStep()     10   0.9011   0.9848   1.0660   0.0417     4.2382     4.24 %
  DistributionMapping::LeastUsedCPUs()  8   0.2591   0.8659   1.1745   0.2145    24.7706     3.73 %
               Amr::writePlotFile()     2   0.7411   0.8527   0.9092   0.0342     4.0136     3.67 %
                 Amr::checkPoint()      2   0.7316   0.8281   0.9027   0.0432     5.2157     3.57 %
            CollectData_Alltoall()     14   0.3830   0.7439   0.9792   0.1539    20.6860     3.20 %
```

Function Name

Number of calls on MPI rank 0.

Minimum, average, and maximum time spent in region across all MPI ranks.

Standard deviation and coefficient of variation across all MPI ranks.

Percent of total selected time.

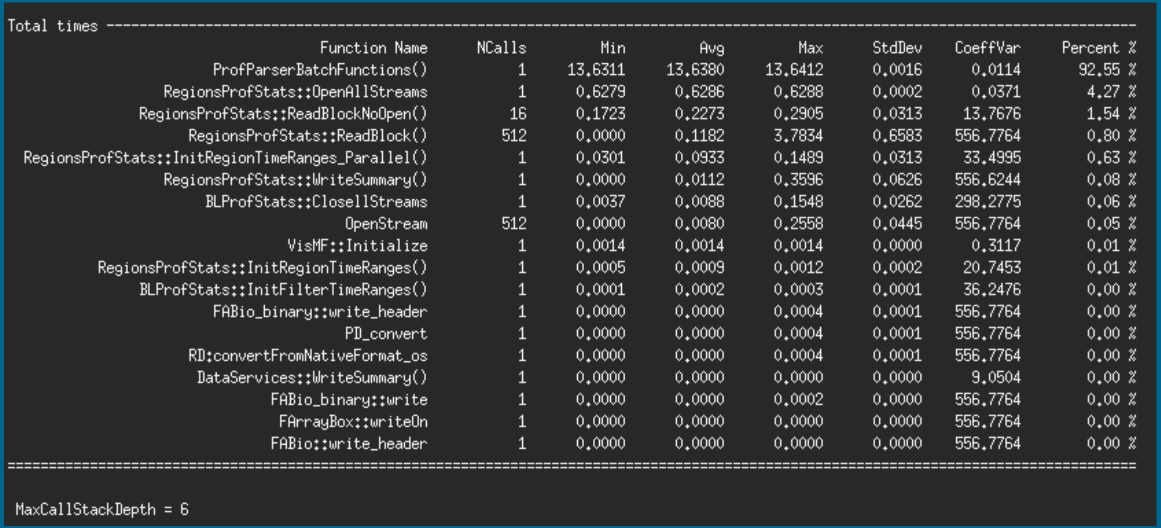
3 Lists, in order:

- Alphabetical, exclusive.
  - ➢ Total times & percent selected.
- Percent time, exclusive.
- Total time, inclusive.

- Prior to clicking on "Generate Function List", it provides a simple overview similar to that provided by TINYPROFILER.
  - (Does not require parsing through the profiling database. VERY quick, even for large databases.)
- After clicking on "Generate Function List", it is more exact data from the full database information.

★ The function list results will filter based on both the subregion and the selected regions.

Function List: What to Look Out For



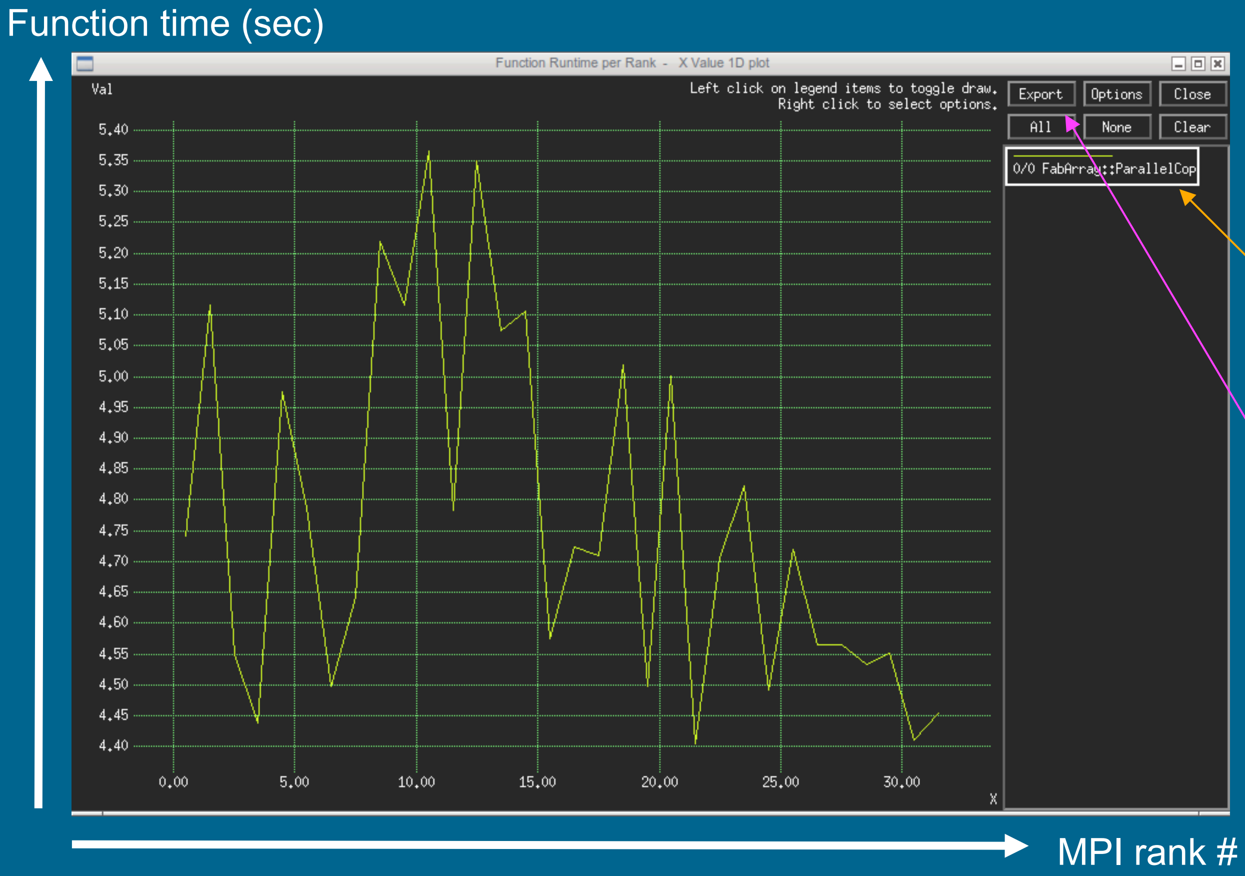
```
Total times -------------------------------------------------------------------------------------------------------------------------------
                         Function Name     NCalls        Min        Avg        Max     StdDev   CoeffVar   Percent %
              ProfParserBatchFunctions()        1    13.6311    13.6380    13.6412     0.0016     0.0114    92.55 %
         RegionsProfStats::OpenAllStreams        1     0.6279     0.6286     0.6288     0.0002     0.0371     4.27 %
      RegionsProfStats::ReadBlockNoOpen()       16     0.1723     0.2273     0.2905     0.0313    13.7676     1.54 %
            RegionsProfStats::ReadBlock()      512     0.0000     0.1182     3.7834     0.6583   556.7764     0.80 %
RegionsProfStats::InitRegionTimeRanges_Parallel()   1     0.0301     0.0933     0.1489     0.0313    33.4995     0.63 %
         RegionsProfStats::WriteSummary()        1     0.0000     0.0112     0.3596     0.0626   556.6244     0.08 %
              BLProfStats::CloseAllStreams        1     0.0037     0.0088     0.1548     0.0262   298.2775     0.06 %
                              OpenStream      512     0.0000     0.0080     0.2558     0.0445   556.7764     0.05 %
                        VisMF::Initialize        1     0.0014     0.0014     0.0014     0.0000     0.3117     0.01 %
     RegionsProfStats::InitRegionTimeRanges()    1     0.0005     0.0009     0.0012     0.0002    20.7453     0.01 %
         BLProfStats::InitFilterTimeRanges()     1     0.0001     0.0002     0.0003     0.0001    36.2476     0.00 %
               FABio_binary::write_header        1     0.0000     0.0000     0.0004     0.0001   556.7764     0.00 %
                               PD_convert        1     0.0000     0.0000     0.0004     0.0001   556.7764     0.00 %
             RD:convertFromNativeFormat_os        1     0.0000     0.0000     0.0004     0.0001   556.7764     0.00 %
              DataServices::WriteSummary()        1     0.0000     0.0000     0.0000     0.0000     9.0504     0.00 %
                      FABio_binary::write        1     0.0000     0.0000     0.0002     0.0000   556.7764     0.00 %
                       FArrayBox::writeOn        1     0.0000     0.0000     0.0000     0.0000   556.7764     0.00 %
                     FABio::write_header        1     0.0000     0.0000     0.0000     0.0000   556.7764     0.00 %
===========================================================================================================================================

 MaxCallStackDepth = 6
```

- Top time consuming functions and total percent spent in them.
- Functions that you wouldn’t expect to be called.
- Very high number of calls.
- Very high standard deviation or coefficient of variation.

Function Plot:

Click on a function in the function list to get a plot of the amount of time each MPI rank spent in that function.



Left click to recenter plot.

Left click & drag to zoom to box.

Right click to reset the view.

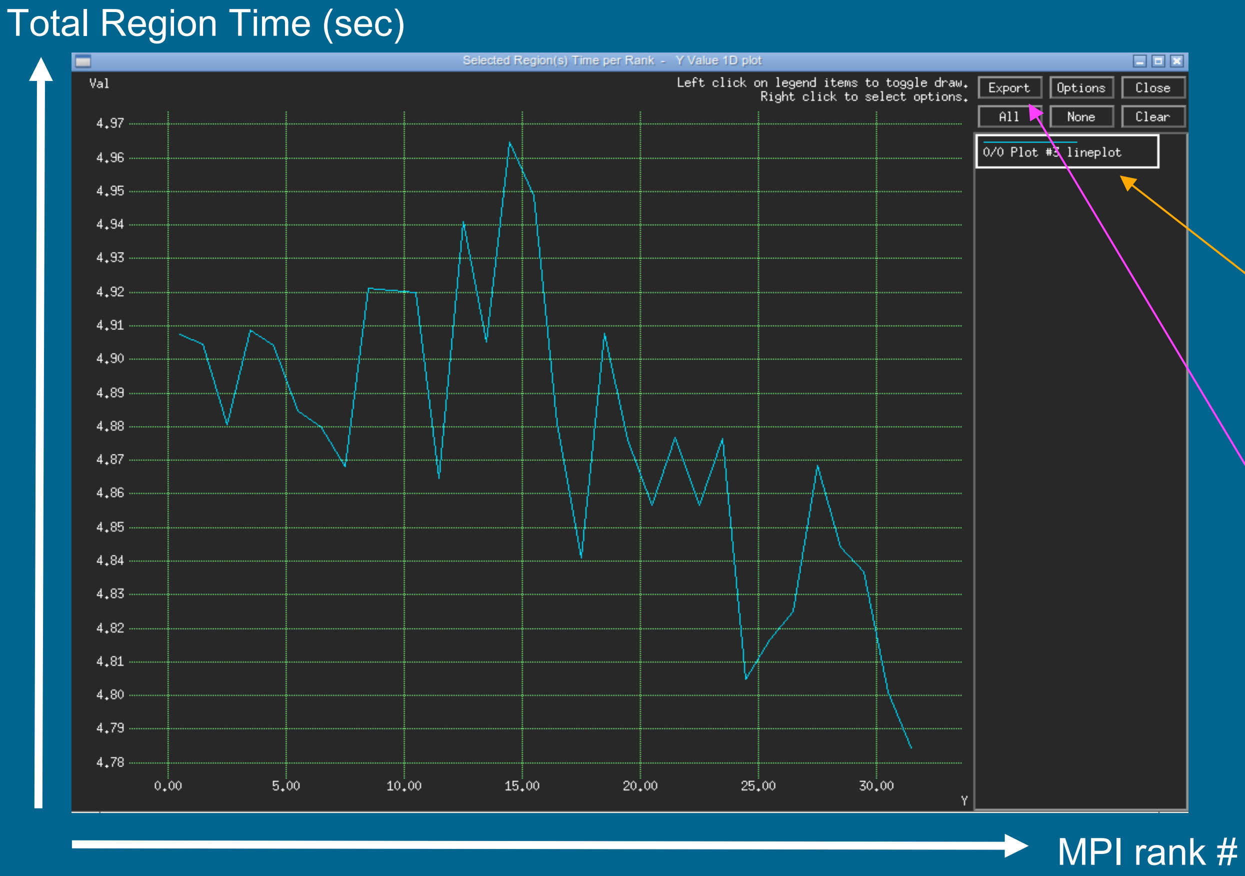
Can plot multiple functions on a single plot. Turn on and off by left clicking on the function.

In the future, will be able to export the time/rank data for your own processing.
(No output formatting options setup yet for Profvis.)

BEWARE the time axes!!! Will scale based on the min/max time of the selected functions, so can be misleading.

Region Time Plot:

Select regions of interest and click on “Generate Region Time Plot” to make a plot of total time in the selected regions for each MPI.



Left click to recenter plot.

Left click & drag to zoom to box.

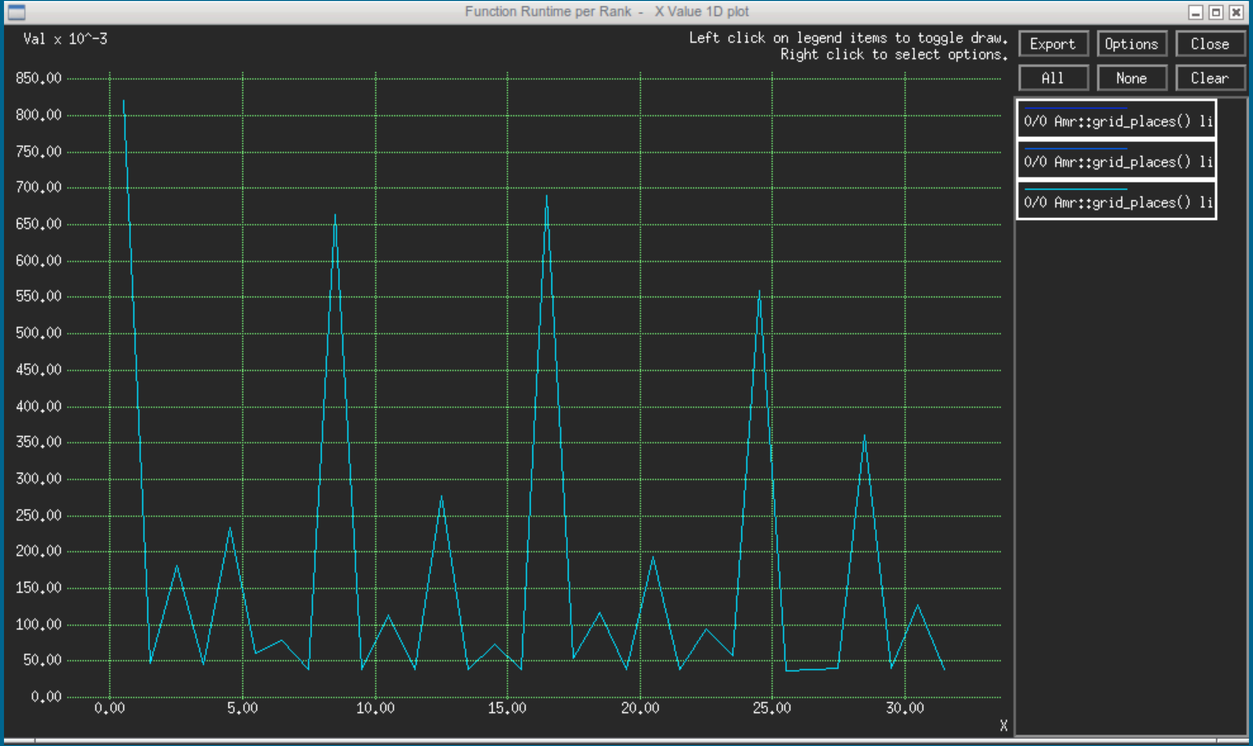
Right click to reset the view.

Can plot multiple functions on a single plot. Turn on and off by left clicking on the function.

In the future, will be able to export the time/rank data for your own processing. (No output formatting options setup yet for Profvis.)

BEWARE the time axes!!! Will scale based on the min/max time of the selected functions, so can be misleading.

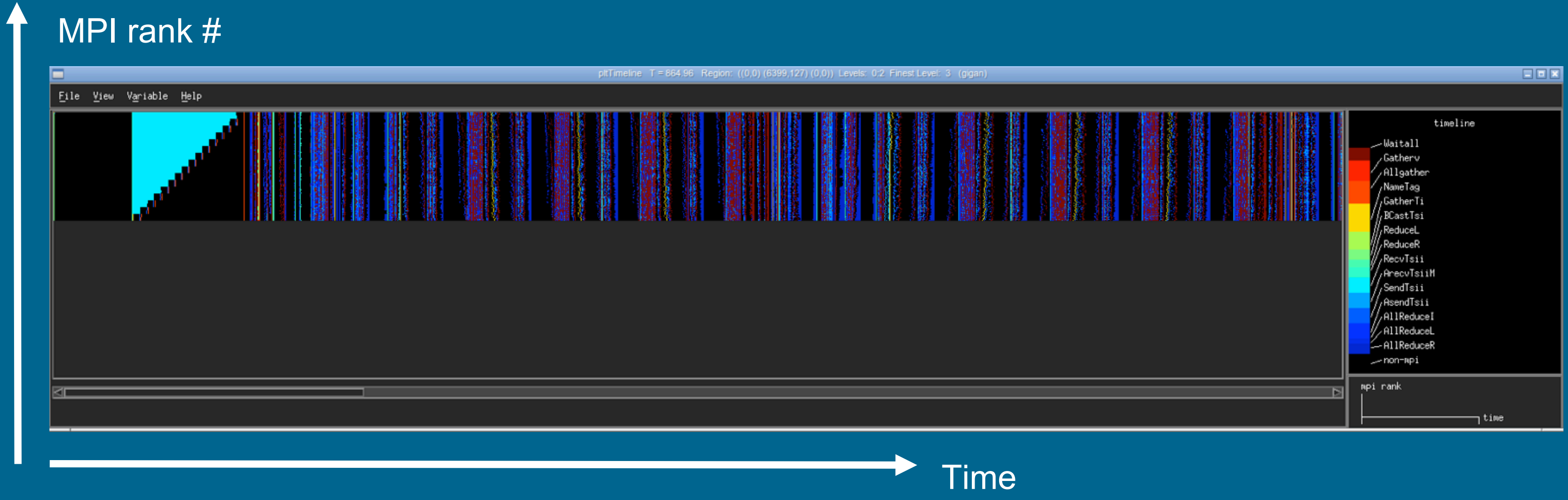
Function Plot & Region Time Plot: What to look out for


Function Runtime per Rank - X Value 1D plot
Val x 10^-3
Left click on legend items to toggle draw.
Right click to select options.
Export
Options
Close
All
None
Clear
0/0 Amr::grid_places() li
0/0 Amr::grid_places() li
0/0 Amr::grid_places() li
850.00
800.00
750.00
700.00
650.00
600.00
550.00
500.00
450.00
400.00
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
0.00
5.00
10.00
15.00
20.00
25.00
30.00
X

- ★ Very large range of runtimes.
- ★ Single / small number of MPI ranks that are doing all the work.
- ★ Unexpected patterns of work.

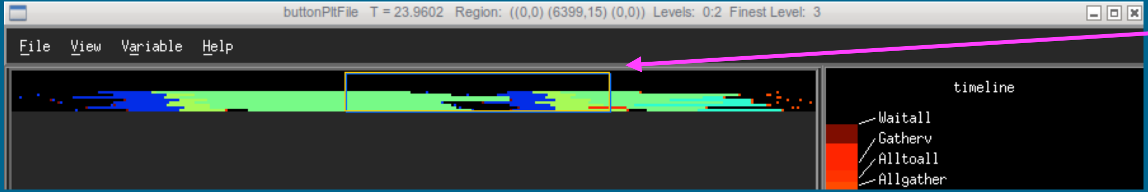
Timeline Plotfile:

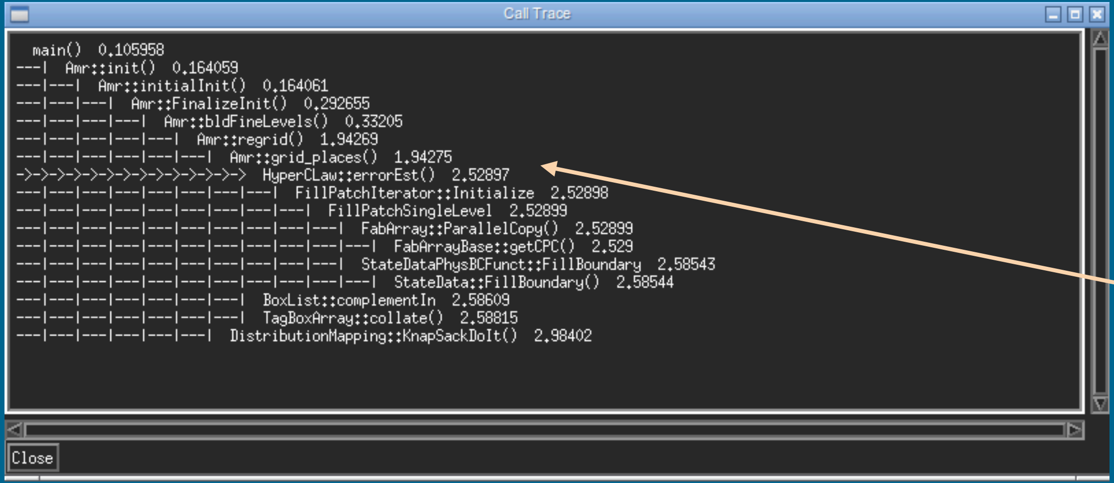
“Generate Timeline” creates a plot of MPI calls for each process over time, giving a picture of the communications across your application.



- Extremely useful to visualize your communication and find bottlenecks.
- Parses based on slicing but not regions (would leave giant blank areas between regions.)
- Currently, the time is incorrect in sliced timelines, but the chart is still accurate.

Timeline Plotfile: Call trace


buttonPltFile T = 23.9602 Region: ((0,0) (6399,15) (0,0)) Levels: 0:2 Finest Level: 3
File
View
Variable
Help
timeline
Waitall
Gatherv
Alltoall
Allgather



```
 main()  0.105958
---|  Amr::init()  0.164059
---|---|  Amr::initialInit()  0.164061
---|---|---|  Amr::FinalizeInit()  0.292655
---|---|---|---|  Amr::bldFineLevels()  0.33205
---|---|---|---|---|  Amr::regrid()  1.94269
---|---|---|---|---|---|  Amr::grid_places()  1.94275
->->->->->->->->->->->->->->  HyperCLaw::errorEst()  2.52897
---|---|---|---|---|---|---|---|  FillPatchIterator::Initialize  2.52898
---|---|---|---|---|---|---|---|---|  FillPatchSingleLevel  2.52899
---|---|---|---|---|---|---|---|---|---|  FabArray::ParallelCopy()  2.52899
---|---|---|---|---|---|---|---|---|---|---|  FabArrayBase::getCPC()  2.529
---|---|---|---|---|---|---|---|---|---|  StateDataPhysBCFunct::FillBoundary  2.58543
---|---|---|---|---|---|---|---|---|---|---|  StateData::FillBoundary()  2.58544
---|---|---|---|---|---|---|  BoxList::complementIn  2.58609
---|---|---|---|---|---|---|  TagBoxArray::collate()  2.58815
---|---|---|---|---|---|  DistributionMapping::KnapSackDoIt()  2.98402
```

Open with "View...Call Trace".

Click and drag to select a region.

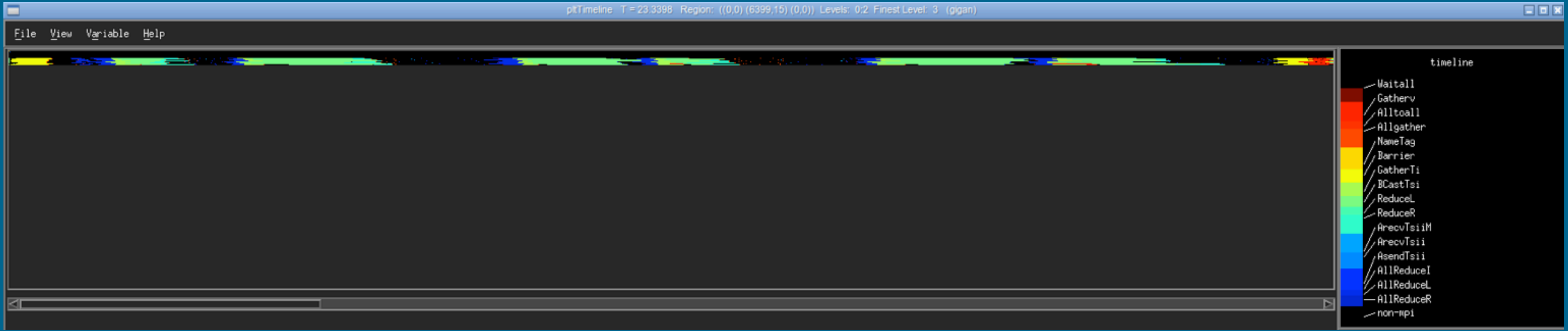
Ctrl - Left Click and drag to select a box that covers all processors (vertical box).

Useless for a call trace, but: Shift - Left Click and drag to select all time (horizontal box).

Displays entire call trace history in the box and start time of that call.

Arrows denote location at the end of the box. (Everything below was ran and exited.)

Timeline Plotfile: What to Look Out For



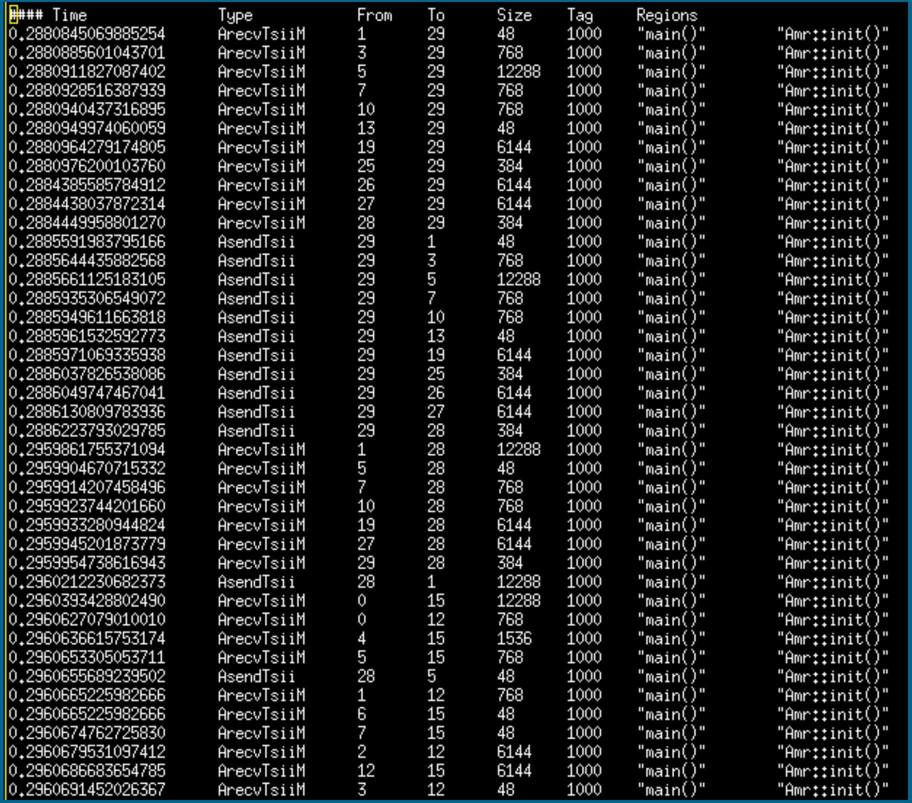
- Large barriers or MPI calls waiting for results.
- Blocks of MPI calls (suggesting processors are waiting.)
- Non-structured communication patterns that can be improved.

Colors / legend can be confusing.

Be sure to double check what's what by looking at the data set.

Send/Recv List:

Detailed list of all point-to-point MPI calls, sorted chronologically.



```
#### Time               Type         From   To    Size   Tag    Regions
0.2880845069885254     ArecvTsiiM   1      29    48     1000   "main()"        "Amr::init()"
0.2880885601043701     ArecvTsiiM   3      29    768    1000   "main()"        "Amr::init()"
0.2880911827087402     ArecvTsiiM   5      29    12288  1000   "main()"        "Amr::init()"
0.2880928516387939     ArecvTsiiM   7      29    768    1000   "main()"        "Amr::init()"
0.2880940437316895     ArecvTsiiM   10     29    768    1000   "main()"        "Amr::init()"
0.2880949974060059     ArecvTsiiM   13     29    48     1000   "main()"        "Amr::init()"
0.2880964279174805     ArecvTsiiM   19     29    6144   1000   "main()"        "Amr::init()"
0.2880976200103760     ArecvTsiiM   25     29    384    1000   "main()"        "Amr::init()"
0.2884385585784912     ArecvTsiiM   26     29    6144   1000   "main()"        "Amr::init()"
0.2884438037872314     ArecvTsiiM   27     29    6144   1000   "main()"        "Amr::init()"
0.2884449958801270     ArecvTsiiM   28     29    384    1000   "main()"        "Amr::init()"
0.2885591983795166     AsendTsii    29     1     48     1000   "main()"        "Amr::init()"
0.2885644435882568     AsendTsii    29     3     768    1000   "main()"        "Amr::init()"
0.2885661125183105     AsendTsii    29     5     12288  1000   "main()"        "Amr::init()"
0.2885935306549072     AsendTsii    29     7     768    1000   "main()"        "Amr::init()"
0.2885949611663818     AsendTsii    29     10    768    1000   "main()"        "Amr::init()"
0.2885961532592773     AsendTsii    29     13    48     1000   "main()"        "Amr::init()"
0.2885971069335938     AsendTsii    29     19    6144   1000   "main()"        "Amr::init()"
0.2886037826538086     AsendTsii    29     25    384    1000   "main()"        "Amr::init()"
0.2886049747467041     AsendTsii    29     26    6144   1000   "main()"        "Amr::init()"
0.2886130809783936     AsendTsii    29     27    6144   1000   "main()"        "Amr::init()"
0.2886223793029785     AsendTsii    29     28    384    1000   "main()"        "Amr::init()"
0.2959861755371094     ArecvTsiiM   1      28    12288  1000   "main()"        "Amr::init()"
0.2959904670715332     ArecvTsiiM   5      28    48     1000   "main()"        "Amr::init()"
0.2959914207458496     ArecvTsiiM   7      28    768    1000   "main()"        "Amr::init()"
0.2959923744201660     ArecvTsiiM   10     28    768    1000   "main()"        "Amr::init()"
0.2959933280944824     ArecvTsiiM   19     28    6144   1000   "main()"        "Amr::init()"
0.2959945201873779     ArecvTsiiM   27     28    6144   1000   "main()"        "Amr::init()"
0.2959954738616943     ArecvTsiiM   29     28    384    1000   "main()"        "Amr::init()"
0.2960212230682373     AsendTsii    28     1     12288  1000   "main()"        "Amr::init()"
0.2960393428802490     ArecvTsiiM   0      15    12288  1000   "main()"        "Amr::init()"
0.2960627079010010     ArecvTsiiM   0      12    768    1000   "main()"        "Amr::init()"
0.2960636615753174     ArecvTsiiM   4      15    1536   1000   "main()"        "Amr::init()"
0.2960653305053711     ArecvTsiiM   5      15    768    1000   "main()"        "Amr::init()"
0.2960655689239502     AsendTsii    28     5     48     1000   "main()"        "Amr::init()"
0.2960665225982666     ArecvTsiiM   1      12    768    1000   "main()"        "Amr::init()"
0.2960665225982666     ArecvTsiiM   6      15    48     1000   "main()"        "Amr::init()"
0.2960674762725830     ArecvTsiiM   7      15    48     1000   "main()"        "Amr::init()"
0.2960679531097412     ArecvTsiiM   2      12    6144   1000   "main()"        "Amr::init()"
0.2960686683654785     ArecvTsiiM   12     15    6144   1000   "main()"        "Amr::init()"
0.2960691452026367     ArecvTsiiM   3      12    48     1000   "main()"        "Amr::init()"
```

Lists in chronological order:

- Time of function call
- Type of MPI function call
- From rank
- To rank
- Size of the message
- Message tag
- Regions that the call occurred within.

Filtered by both subregions and region selection.

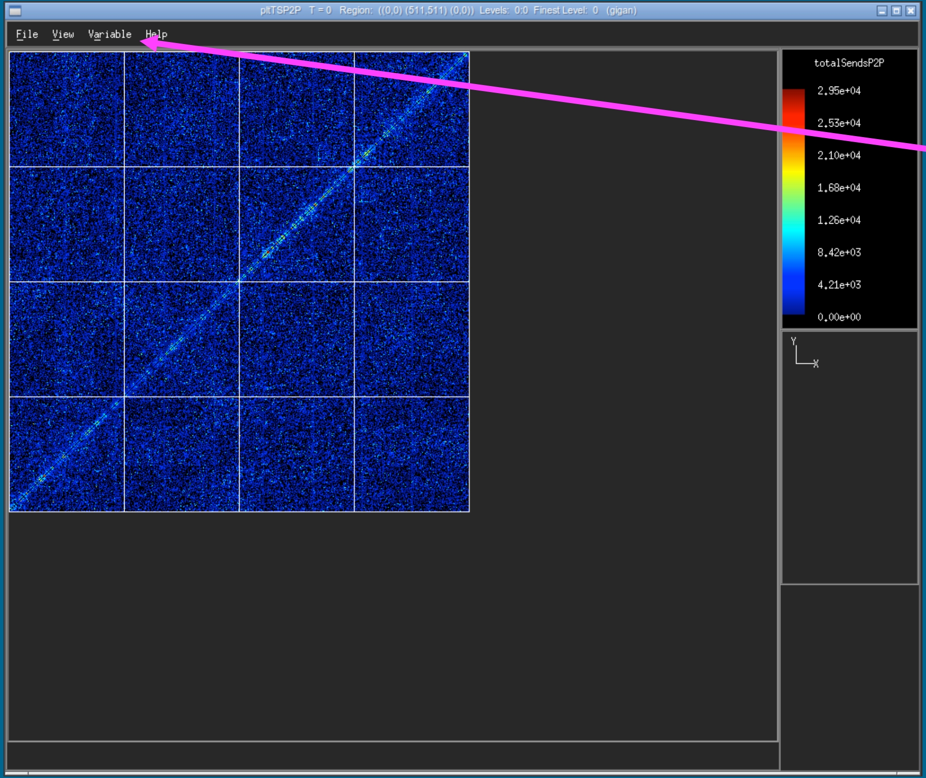
Specially made for MPI analysis for Cy Chan's research group.

If you can parse the data yourself, feel free to use it in your own studies. If you need a different version of something similar, let me know.

Sends Plotfile:

Creates plot file describing the total number of MPI point-to-point sends and the total size of messages between each pair or MPI ranks.

To MPI rank #



From MPI rank #

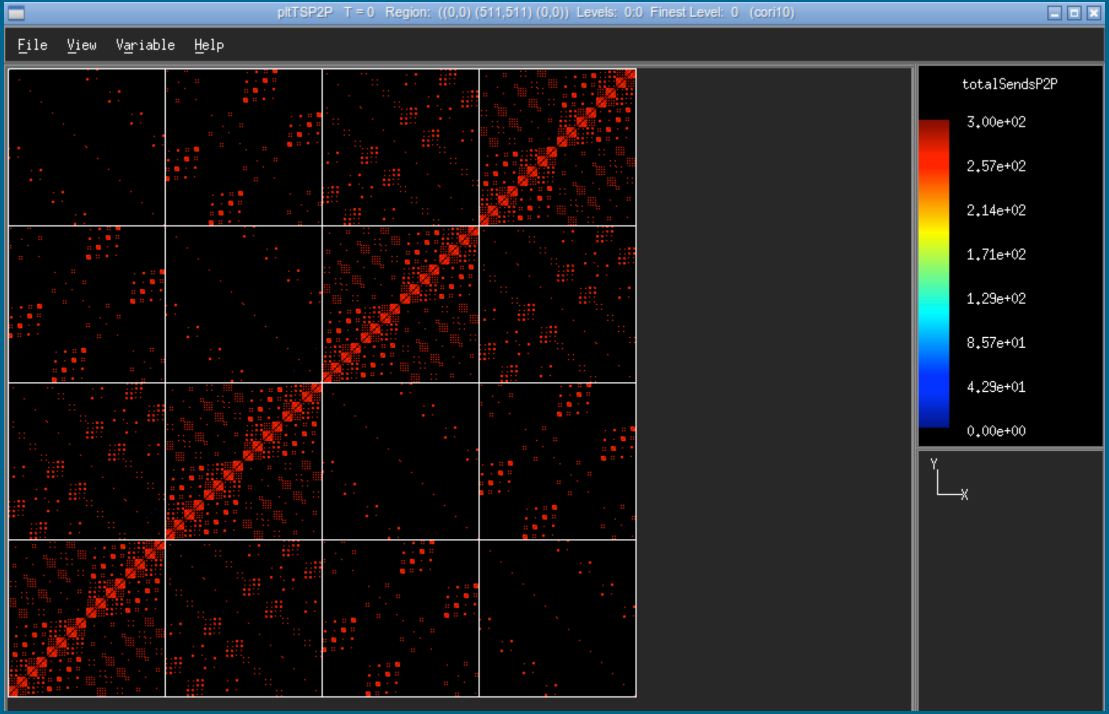
Use the "Variable" drop-down menu to switch between:

- number of calls (totalSendsP2P)
- total message size in bytes (totalSentDataP2P)

Currently, building a Sends plotfile in the GUI only works in a serial build of AMRVis.

However, the serial version will properly filter over subregions and regions.

Sends Plotfile: What to look out for



- Hotspots in data sent or number of calls.
- Extremely asymmetric plots; all the data is going one-way.
- Procs with no communication at all; black columns or rows.
- More ‘far away’ communication than ‘nearby’ communication; biggest contributions are away from the diagonal.

For Profiling Help Contact:

Kevin Gott:

kngott@lbl.gov