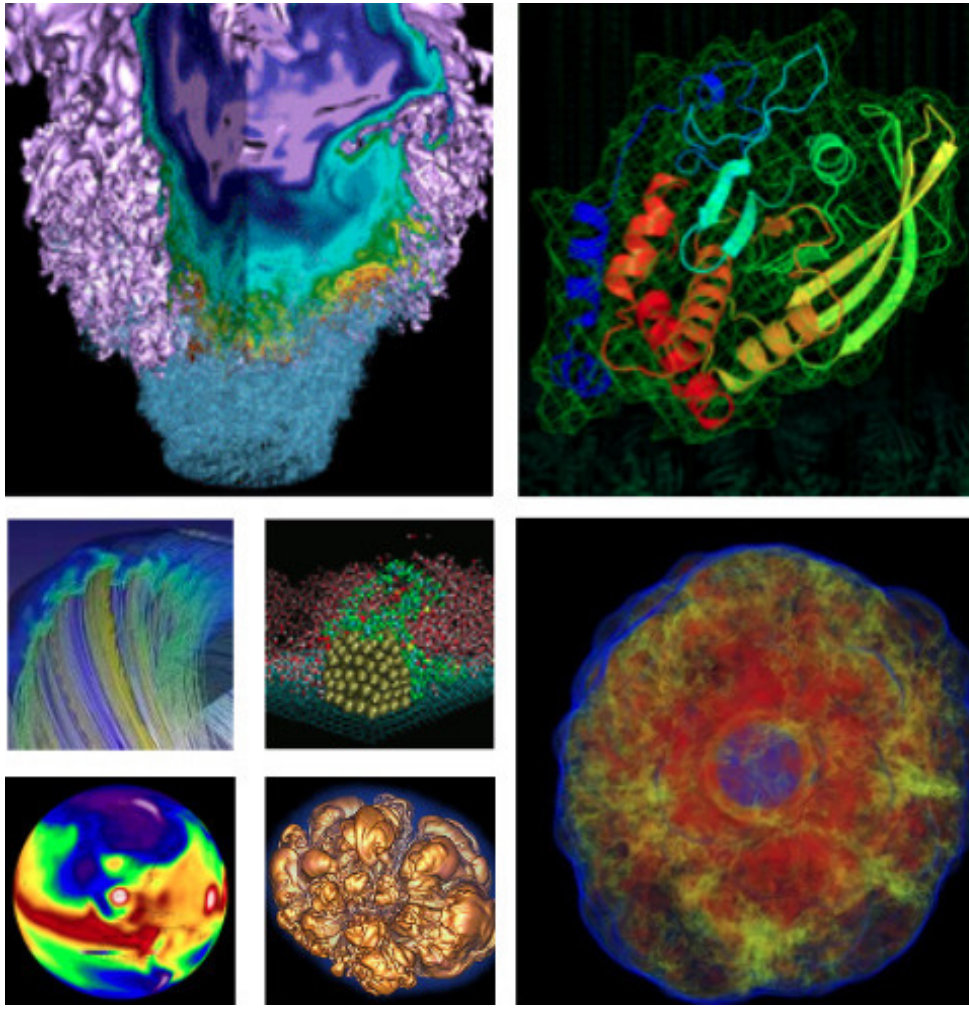
Geant4 + TiMemory

Automated Timing + Memory Analysis



National Energy Research Scientific Computing Center

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Jonathan R. Madsen

✉ jrmadsen@lbl.gov

National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory

August 28, 2018

- TiMemory is a lightweight "in-situ" package for consistent and automated performance reports
  - Supplements profilers – profilers have overhead and that overhead skews/misrepresents run-time
- Cross-language ⇒ C, C++ , and Python implementations can be used independently or simultaneously
- Designed to be very easy to use
  - Uses macros *e.g.*, `TIMEMORY_AUTO_TIMER("")` macro at beginning of function
    - When not compiled with `TiMemory` support this is an empty macro
  - insert "auto-timer" without parameter customization: function, file, and line-number automatically recorded
- Macro creates an object that starts measurements upon construction, stops measurements upon destruction, and accumulates results over multiple "laps"
- Records call-stack
- Support for PAPI hardware counters and CUDA event timers actively under development

- JSON serialization
- Built-in features (via Python)
  - Timing and memory plotting
  - Support for attaching analysis as `CTEST_NOTES` and attaching images as `<DartMeasurementFile>` to CDash dashboard
- MPI-compatible
  - No data merging occurs other than one combined report, each MPI process has independent data
- Thread-safe
  - Automated merging of thread-local data when report is requested or when thread exits
- Termination signal handling (SIGHUP, SIGQUIT, etc.) to dump "profile" when tracking down memory and/or timing issues

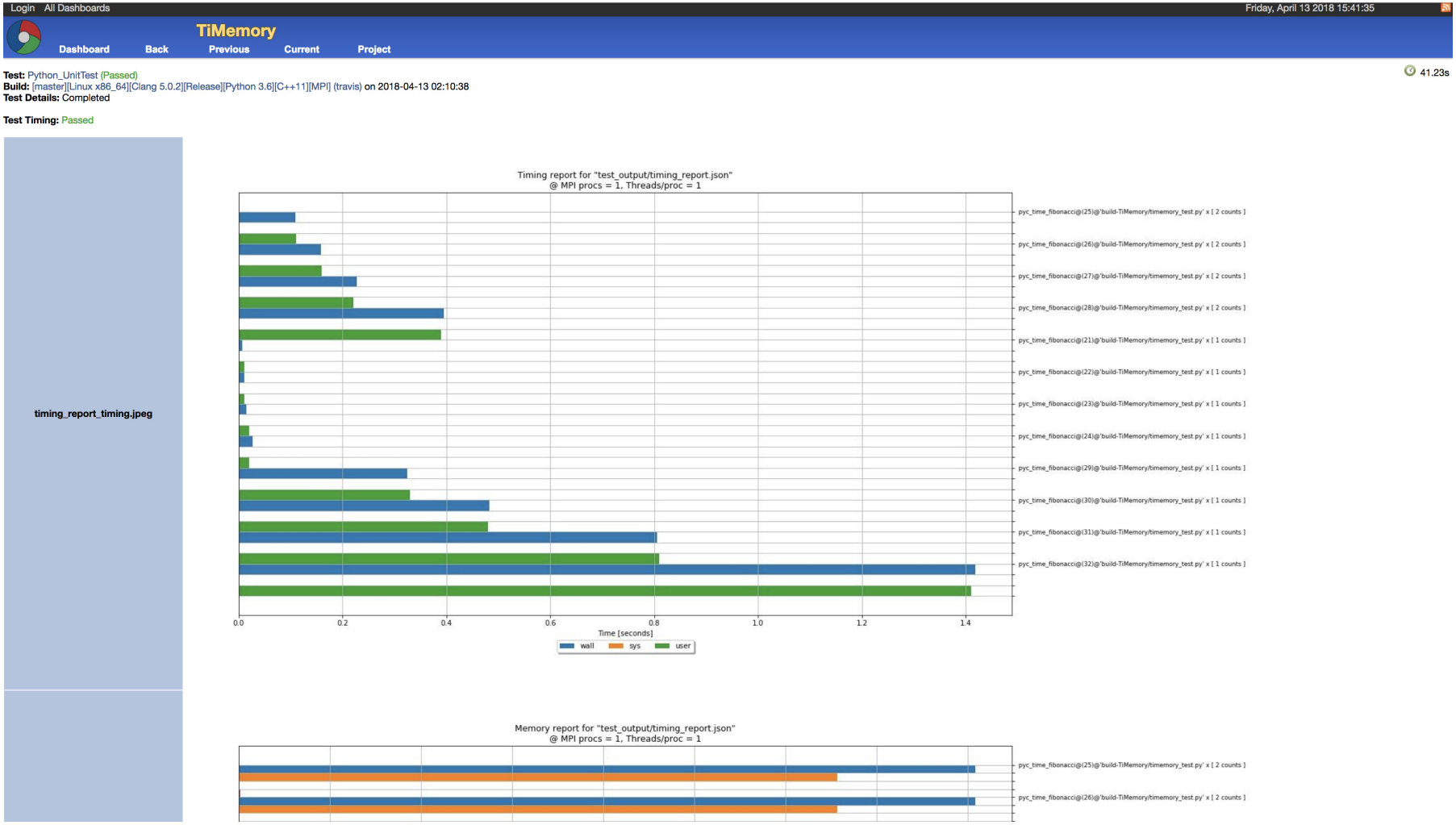


Figure 1: Plot of timing/memory in CDash

Login All Dashboards Friday, April 13 2018 15:43:19

TiMemory

Dashboard Back Previous Current Project

Site: travis
Build Name: [master][Linux x86_64][Clang 5.0.2][Release][Python 3.6][C++11][MPI]
Stamp: 20180413-0910-Continuous

- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_array_test.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/nested_report.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/./timing_report.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_report.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_decorator.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_depth.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_toggle.out
- 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_context_manager.out

2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/timing_array_test.out

```
        > [pyc] main@'array_test.py':205                                                  :  4.142 wall,  1.170 user +  2.970 system =  4.140 CPU [sec] (100.0%) : RSS {tot,self}_{curr,peak} : (  0.1|572.1) | (  0.1|572.0) [MB] (total # of laps: 3)
> [pyc] |_run@'array_test.py':174                                                  :  3.924 wall,  1.160 user +  2.760 system =  3.920 CPU [sec] ( 99.9%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (190.7|381.3) [MB] (total # of laps: 15)
> [pyc]   |_create@'array_test.py':177                                              :  1.569 wall,  0.320 user +  1.230 system =  1.550 CPU [sec] ( 98.8%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (190.7|190.7) [MB] (total # of laps: 15)
> [pyc]     |___array_finalize__[auto_disk_array]@'array_test.py':156               :  0.001 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.2|  0.0) [MB] (total # of laps: 15)
> [pyc]     |___init__[auto_array_weakref]@'array_test.py':125                      :  0.004 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc]       |_enable_auto_rollback[auto_array_weakref]@'array_test.py':102        :  0.003 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc]         |_bind_finalizer[auto_array_weakref]@'array_test.py':68             :  0.001 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc]   |___call__[auto_disk_array]@'array_test.py':167                           :  0.001 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc]   |___finalize__[auto_array_weakref]@'array_test.py':120                    :  0.003 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc]     |_do_rollback[auto_array_weakref]@'array_test.py':134                   :  0.001 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc] |___del__[auto_disk_array]@'array_test.py':163                              :  0.001 wall,  0.000 user +  0.010 system =  0.010 CPU [sec] (1261.0%) : RSS {tot,self}_{curr,peak} : (381.6|572.1) | (  0.0|  0.0) [MB] (total # of laps: 15)
> [pyc] dummy@'array_test.py':217                                                   :  1.002 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (  0.1|572.1) | (  0.0|  0.0) [MB]
> [pyc] |_do_sleep@'array_test.py':220                                              :  1.001 wall,  0.000 user +  0.000 system =  0.000 CPU [sec] (  0.0%) : RSS {tot,self}_{curr,peak} : (  0.1|572.1) | (  0.0|  0.0) [MB]
```

2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test_output/nested_report.out

```
        > [pyc] main'AUTO_TIMER_FOR_NESTED_TEST':104[int]@'nested_test.py':104  :  8.640 wall,  2.410 user +  0.220 system =  2.630 CPU [sec] ( 30.4%) : RSS {tot,self}_{curr,peak} : ( 5.2|10.7) | ( 5.2|10.7) [MB]
> [pyc] |_nested_func_1[int](15)@'nested_test.py':69                              :  0.453 wall,  0.400 user +  0.040 system =  0.440 CPU [sec] ( 97.2%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.4| 0.4) [MB] (total # of laps: 2)
> [pyc]   |_fibonacci(15)@'nested_test.py':64                                      :  0.441 wall,  0.390 user +  0.040 system =  0.430 CPU [sec] ( 97.4%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.4| 0.4) [MB] (total # of laps: 2)
> [pyc]     |_fibonacci(14)@'nested_test.py':64                                    :  0.283 wall,  0.250 user +  0.040 system =  0.290 CPU [sec] (102.3%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.2| 0.2) [MB] (total # of laps: 2)
> [pyc]       |_fibonacci(13)@'nested_test.py':64                                  :  0.174 wall,  0.160 user +  0.010 system =  0.170 CPU [sec] ( 97.7%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.1| 0.1) [MB] (total # of laps: 2)
> [pyc]         |_fibonacci(12)@'nested_test.py':64                                :  0.106 wall,  0.100 user +  0.000 system =  0.100 CPU [sec] ( 94.7%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.1| 0.1) [MB] (total # of laps: 2)
> [pyc]           |_fibonacci(11)@'nested_test.py':64                              :  0.065 wall,  0.060 user +  0.000 system =  0.060 CPU [sec] ( 92.7%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]             |_fibonacci(10)@'nested_test.py':64                            :  0.039 wall,  0.040 user +  0.000 system =  0.040 CPU [sec] (101.9%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]               |_fibonacci(9)@'nested_test.py':64                           :  0.023 wall,  0.020 user +  0.000 system =  0.020 CPU [sec] ( 86.9%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                 |_fibonacci(8)@'nested_test.py':64                         :  0.014 wall,  0.020 user +  0.000 system =  0.020 CPU [sec] (147.3%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                   |_fibonacci(7)@'nested_test.py':64                       :  0.008 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (130.1%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                     |_fibonacci(6)@'nested_test.py':64                     :  0.004 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (249.2%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                   |_fibonacci(6)@'nested_test.py':64                       :  0.003 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (297.7%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                 |_fibonacci(8)@'nested_test.py':64                         :  0.014 wall,  0.020 user +  0.000 system =  0.020 CPU [sec] (145.3%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                   |_fibonacci(7)@'nested_test.py':64                       :  0.007 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (139.5%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                     |_fibonacci(6)@'nested_test.py':64                     :  0.003 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (291.7%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                   |_fibonacci(6)@'nested_test.py':64                       :  0.004 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (243.1%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                     |_fibonacci(5)@'nested_test.py':64                     :  0.001 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (691.1%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]             |_fibonacci(9)@'nested_test.py':64                             :  0.023 wall,  0.020 user +  0.000 system =  0.020 CPU [sec] ( 87.9%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]               |_fibonacci(8)@'nested_test.py':64                           :  0.014 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] ( 72.8%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                 |_fibonacci(6)@'nested_test.py':64                         :  0.004 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (257.1%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]               |_fibonacci(7)@'nested_test.py':64                           :  0.007 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (150.2%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
> [pyc]                 |_fibonacci(6)@'nested_test.py':64                         :  0.003 wall,  0.010 user +  0.000 system =  0.010 CPU [sec] (299.1%) : RSS {tot,self}_{curr,peak} : ( 0.4| 0.4) | ( 0.0| 0.0) [MB] (total # of laps: 2)
```

Figure 2: ASCII output of timing + memory in CDash

```
#ifdef GEANT4_USE_TIMEMORY // compiled with TiMemory

#include <timemory/timemory.hpp>

typedef tim::auto_timer G4AutoTimer;

inline void InitializeTiMemory()
{
    tim::manager* instance = tim::manager::instance();
    instance->enable(true);
}

#else   // not compiled with TiMemory

#define TIMEMORY_AUTO_TIMER(str)
#define TIMEMORY_AUTO_TIMER_OBJ(str) {}
#define TIMEMORY_BASIC_AUTO_TIMER(str)
#define TIMEMORY_BASIC_AUTO_TIMER_OBJ(str) {}
#define TIMEMORY_DEBUG_BASIC_AUTO_TIMER(str)
#define TIMEMORY_DEBUG_AUTO_TIMER(str)

inline void InitializeTiMemory() { }

#endif
```

- Insert "auto-timer"

```
void G4RunManager::BeamOn(G4int, const char*, G4int)
{
    TIMEMORY_AUTO_TIMER("");
    // ...
}
```

- Result

```
> [exe] total execution time
                :  25.622 wall, 163.840 user +   0.340 system
                = 164.180 CPU [sec] (640.8%)
                : RSS {tot,self}_{curr,peak}
                : (94.9|94.9) | (71.5|71.5) [MB] (x1 laps)
> [cxx] |_BeamOn@'G4RunManager.cc':327
                :  25.591 wall, 163.820 user +   0.340 system
                = 164.160 CPU [sec] (641.5%)
                : RSS {tot,self}_{curr,peak}
                : (94.9|94.9) | (58.2|58.2) [MB] (x2 laps)
> [cxx]   |_RunInitialization@'G4RunManagerKernel.cc':666
                :   1.634 wall,   1.600 user +   0.030 system
                =   1.630 CPU [sec] ( 99.7%)
                : RSS {tot,self}_{curr,peak}
                : (92.6|92.6) | (22.5|22.5) [MB] (x2 laps)
```

```
|0> [pyc] main@'toast_ground_sim_simple.py'
        : 41.10 wall, 69.15 user + 4.69 system = 73.84 CPU [sec] (179.6%)
```

- Timing fields:
  - Real time *e.g.*, wall clock time
  - User time *e.g.*, time spent executing user code
  - System time *e.g.*, time spent executing system code
  - CPU time *e.g.*, user + system time
  - % CPU utilization *e.g.*, cpu / real * 100
- Total time from all the "laps"

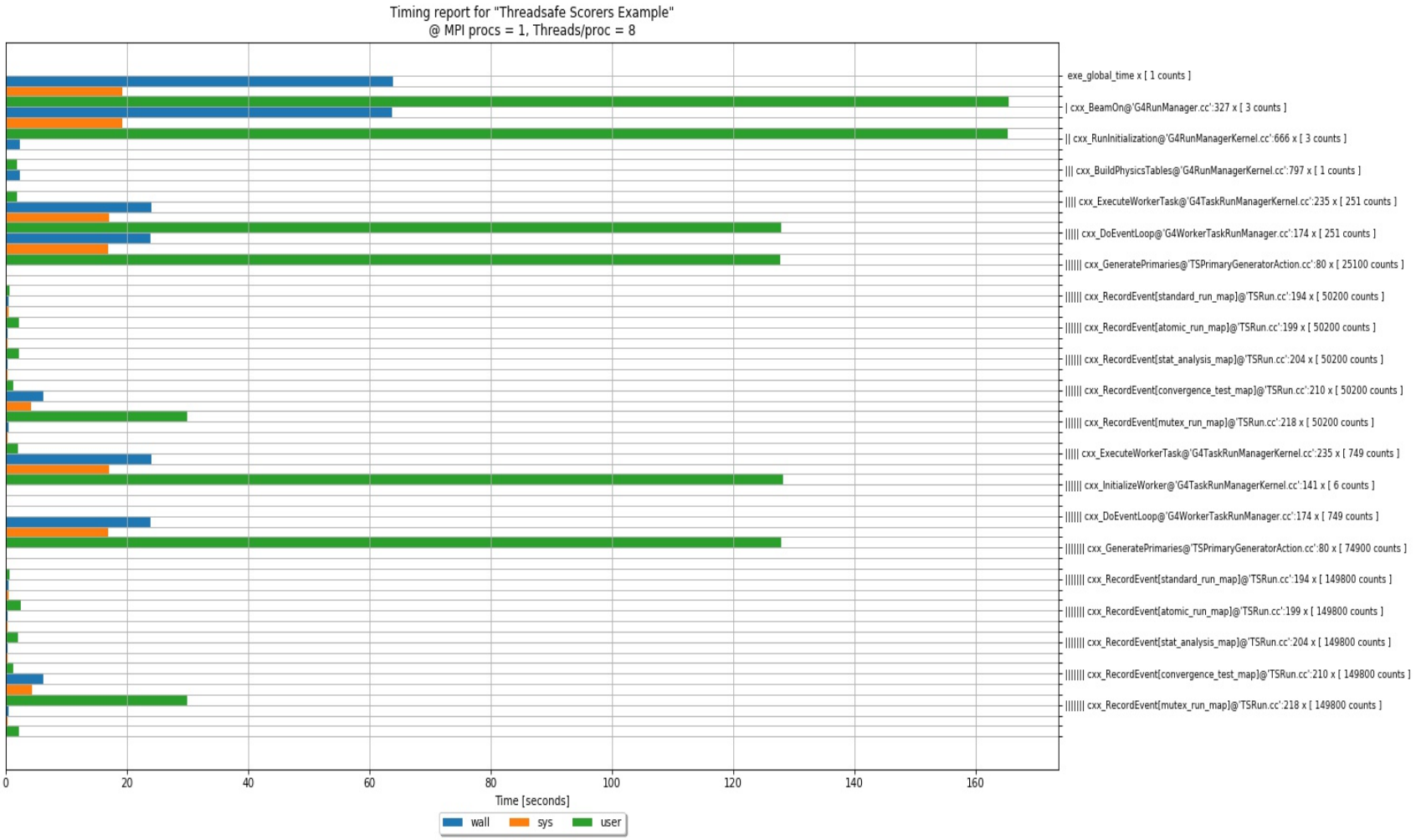


Figure 3: Timing Plot from Geant4 ThreadsafeScorers example using 8 threads

```
|0> [pyc] main@'toast_ground_sim_simple.py'
        : RSS {tot,self}_{curr,peak} : (114.5|223.7) | (107.4|215.6) [MB]
```

- Resident set size (RSS) fields: total-current, total-peak, self-current, self-peak
  - Total fields are measured when the timer was stopped
  - Self RSS fields are difference between timer start and timer stop
  - Current RSS is the measurement of the allocated pages in memory
  - Peak RSS is the peak RSS usage for the process by the OS
- Temporary memory allocation can be calculated via $peak - current$
  - This calculation is valid when the timing scope is setting the "high-water mark" of memory allocation – but this is where the temporary memory usage is of primary interest
  - Relevant example: application involving matrix calculations is throwing runtime OOM/bad_alloc error due to intermediate solution matrices within the matrix library.

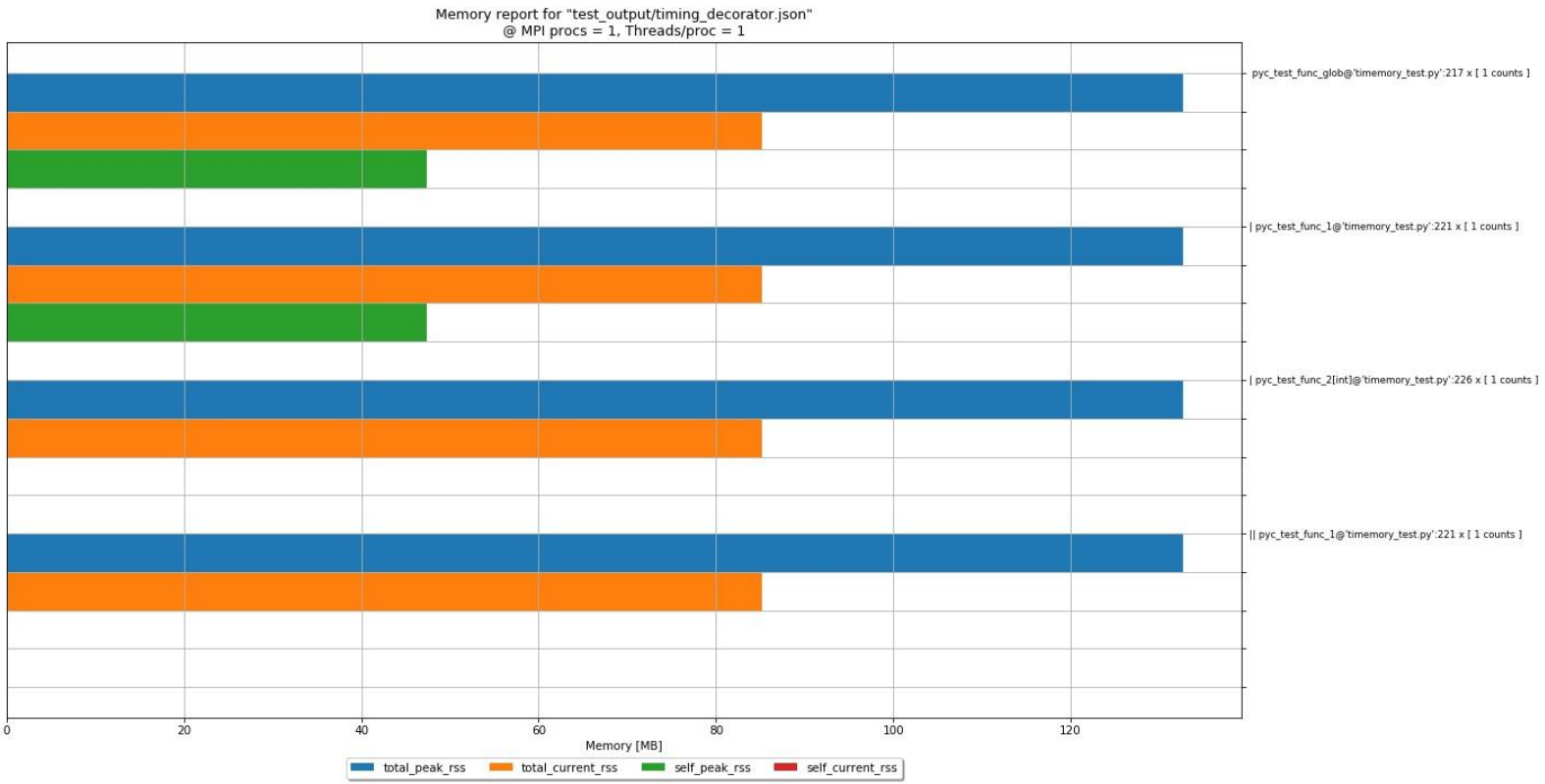


Figure 4: Memory Plot from TiMemory unit test

- TiMemory depends on two header-only libraries (automatically included):
  - PyBind11 – creates Python bindings
  - cereal – produces JSON serialization
- Installation from Python package managers includes the C++ development files
- Install from source: github.com/jrmadsen/TiMemory
- Install from PyPi: `pip install -v timemory`
- Install from Anaconda: `conda install -c jrmadsen timemory`
- Documentation: jrmadsen.github.io/TiMemory
- System requirements
  - Operating system: Windows 10, Linux, macOS
  - Compilers: GCC 4.9+, Clang 4.0+, AppleClang, Intel, MSVC 14, MSVC 15
  - Packages: CMake (>= 2.8.12)
  - Python: 2.7, >= 3.4

- All `rusage` fields
- GPU timers
  - CUDA: `cudaEvent_t` and `cudaEventRecord`
  - OpenCL: `clGetEventProfilingInfo`
- PAPI – hardware counters (Linux only)
  - PAPI_TOT_INS – total instructions issued
  - PAPI_VEC_DP – double precision vector/SIMD instructions
  - PAPI_L1_DCM – Level 1 data cache misses
  - PAPI_FP_OPS – floating-point operations executed
  - ... etc.