Introduction to profiling

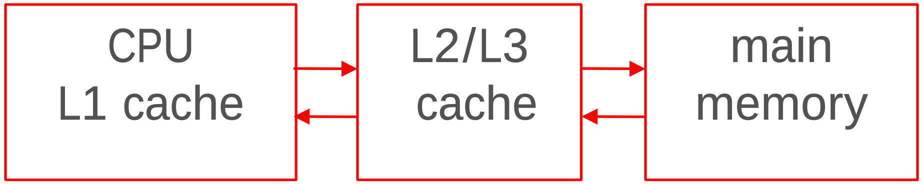
Martin Čuma
Center for High Performance Computing University of Utah
m.cuma@utah.edu

Overview

- Profiling basics
- Simple profiling
- Open source profiling tools
- Intel development tools
  - Advisor XE
  - Inspector XE
  - VTune Amplifier XE
  - Trace Analyzer and Collector
- Interpreted languages profiling
- GPU profilint
- https://www.surveymonkey.com/r/7PFVFCY

Why to profile

- Evaluate performance
- Find the performance bottlenecks
  - inefficient programming
  - memory, I/O bottlenecks
  - vectorization
  - parallel scaling


CPU L1 cache

L2/L3 cache

main memory

Tools categories

- Hardware counters
  - count events from CPU perspective (# of flops, memory loads, etc)
  - usually need Linux kernel module installed
- Statistical profilers (sampling)
  - interrupt program at given intervals to find what routine/line the program is in
- Event based profilers (tracing)
  - collect information on each function call

Simple profiling

- Time program runtime
  - get an idea on time to run and parallel scaling,
    - https://www.chpc.utah.edu/documentation/software/timing.php
- Serial profiling
  - discover inefficient programming
  - computer architecture slowdowns
  - compiler optimizations evaluation
  - gprof
    - Trick how to get gprof to work in parallel: http://shwina.github.io/2014/11/profiling-parallel

Open source tools

- Vendor based
  - – AMD CodeAnalyst
- Community based
  - – perf
    - hardware counter collection, part of Linux
  - – oprofile
    - profiler
  - – drawback – harder to analyze the profiling results

HPC OS tools

- HPC Toolkit
  - A few years old, did not find it as straightforward to use
- TAU
  - Lots of features, which makes the learning curve slow
- Scalasca
  - Developed by European consortium, did not try yet

Intel software development products

- We have a 2 concurrent users license
- Tools for all stages of development
  - Compilers and libraries
  - Verification tools
  - Profilers
- More info

`https://software.intel.com/en-us/intel-parallel-studio-xe`

`https://www.chpc.utah.edu/documentation/software/intel-parallelXE.php`

Intel tools

- Intel Parallel Studio XE 2020 Cluster Edition
  - Compilers (C/C++, Fortran)
  - Distribution for Python
  - Math library (MKL)
  - Data Analytics Acceleration Library (DAAL)
  - Threading library (TBB)
  - Vectorization or thread design and prototype (Advisor)
  - Memory and thread debugging (Inspector)
  - Profiler (VTune)
  - MPI library (Intel MPI)
  - MPI analyzer and profiler (ITAC)

Intel Vtune Profiler

- Serial and parallel profiler
  - multicore support for OpenMP and OpenCL on CPUs, GPUs and Xeon Phi
- Quick identification of performance bottlenecks
  - various analyses and points of view in the GUI
- GUI and command line use
- More info

https://software.intel.com/en-us/vtune

Intel VTune Amplifier

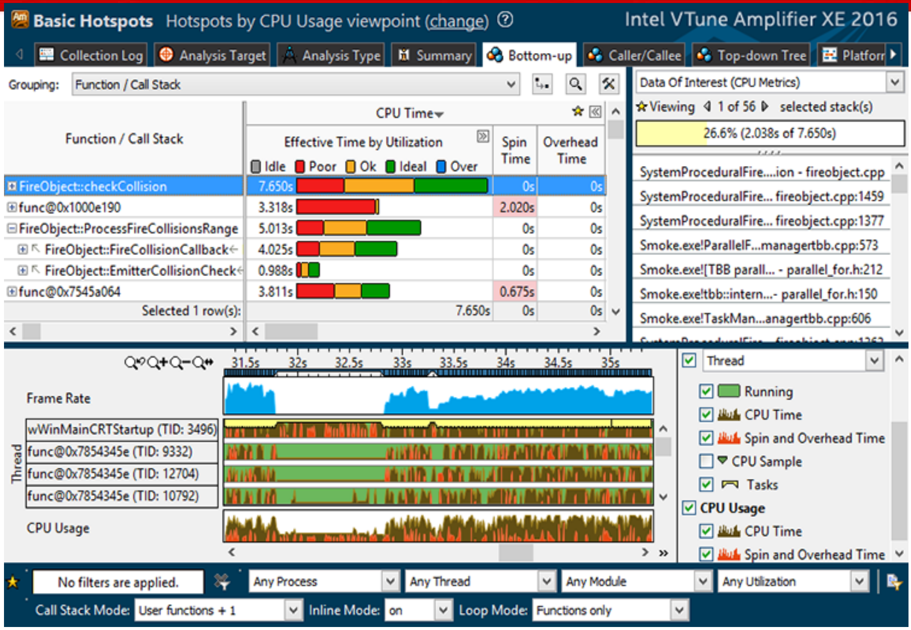
- Source the environment

  `module load vtune`

- Run VTune

  `amplxe-gui` – graphical user interface

  `amplxe-cl` – command line (best to get from the GUI)



Basic Hotspots   Hotspots by CPU Usage viewpoint (change)   Intel VTune Amplifier XE 2016

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time
FireObject::checkCollision	7.650s	0s	0s
func@0x1000e190	3.318s	2.020s	0s
FireObject::ProcessFireCollisionsRange	5.013s	0s	0s
FireObject::FireCollisionCallback	4.025s	0s	0s
FireObject::EmitterCollisionCheck	0.988s	0s	0s
func@0x7545a064	3.811s	0.675s	0s
Selected 1 row(s):	7.650s	0s	0s

Data Of Interest (CPU Metrics)

Viewing 1 of 56 selected stack(s)

26.6% (2.038s of 7.650s)

SystemProceduralFire....ion - fireobject.cpp
SystemProceduralFire... fireobject.cpp:1459
SystemProceduralFire... fireobject.cpp:1377
Smoke.exe!ParallelF...managertbb.cpp:573
Smoke.exe![TBB parall... - parallel_for.h:212
Smoke.exe!tbb::intern...- parallel_for.h:150
Smoke.exe!TaskMan...anagertbb.cpp:606

31.5s
32s
32.5s
33s
33.5s
34s
34.5s
35s
Frame Rate
Thread
wWinMainCRTStartup (TID: 3496)
func@0x7854345e (TID: 9332)
func@0x7854345e (TID: 12704)
func@0x7854345e (TID: 10792)
CPU Usage

☑ Thread
☑ Running
☑ CPU Time
☑ Spin and Overhead Time
☐ CPU Sample
☑ Tasks
☑ CPU Usage
☑ CPU Time
☑ Spin and Overhead Time

No filters are applied. | Any Process | Any Thread | Any Module | Any Utilization

Call Stack Mode: User functions + 1   Inline Mode: on   Loop Mode: Functions only

Can be used also for remote profiling (e.g. on Xeon Phi)

- Tuning guides for specific architectures

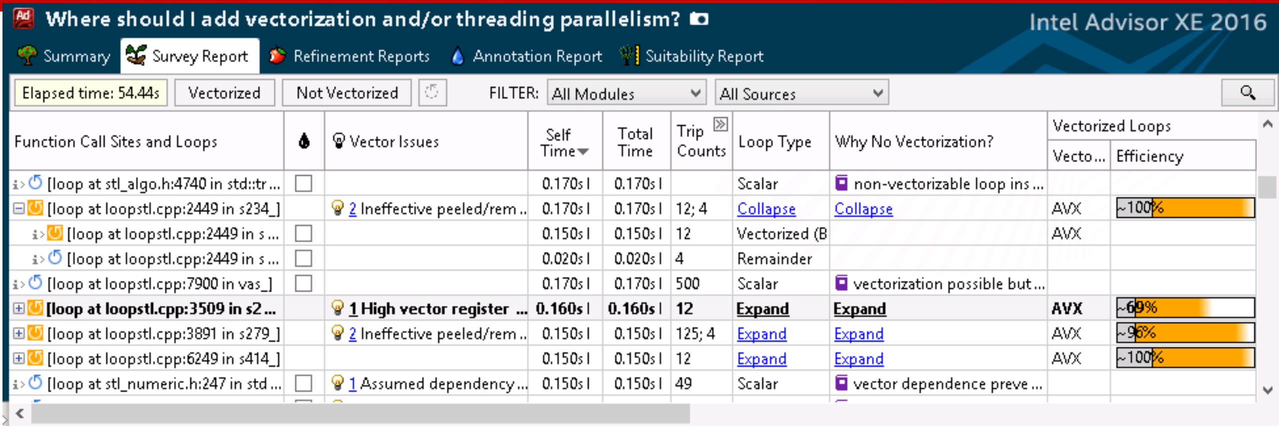
https://software.intel.com/en-us/articles/processor-specific-performance-analysis-papers

Intel Advisor

- Vectorization advisor
  - – Identify loops that benefit from vectorization, what is blocking efficient vectorization and explore benefit of data reorganization
- Thread design and prototyping
  - – Analyze, design, tune and check threading design without disrupting normal development
- More info

http://software.intel.com/en-us/advisor/

Intel Advisor



Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops	
								Vecto...	Efficiency
[loop at stl_algo.h:4740 in std::tr ...	☐		0.170s	0.170s		Scalar	non-vectorizable loop ins ...		
[loop at loopstl.cpp:2449 in s234_]		2 Ineffective peeled/rem ..	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%
[loop at loopstl.cpp:2449 in s ...	☐		0.150s	0.150s	12	Vectorized (B		AVX	
[loop at loopstl.cpp:2449 in s ...	☐		0.020s	0.020s	4	Remainder			
[loop at loopstl.cpp:7900 in vas_]	☐		0.170s	0.170s	500	Scalar	vectorization possible but ...		
[loop at loopstl.cpp:3509 in s2 ...		**1 High vector register ...**	**0.160s**	**0.160s**	**12**	**Expand**	**Expand**	**AVX**	~69%
[loop at loopstl.cpp:3891 in s279_]		2 Ineffective peeled/rem ..	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%
[loop at loopstl.cpp:6249 in s414_]			0.150s	0.150s	12	Expand	Expand	AVX	~100%
[loop at stl_numeric.h:247 in std ...	☐	1 Assumed dependency ...	0.150s	0.150s	49	Scalar	vector dependence preve ...		

- Source the environment

  `module load advisorxe`

- Run Advisor

  `advixe-gui` – graphical user interface

  `advixe-cl` – command line (best to get from the GUI)

- Create project and choose appropriate modeling
- Getting started guide

https://software.intel.com/en-us/get-started-with-advisor

Intel Trace Analyzer and Collector

- MPI profiler
  - traces MPI code
  - identifies communication inefficiencies
- Collector collects the data and Analyzer visualizes them
- More info

https://software.intel.com/en-us/trace-analyzer

Intel TAC

- Source the environment

```
module load itac
```

- Using Intel compilers, can compile with `–trace`

```
mpiifort -openmp –trace trap.f
```

- Run MPI code

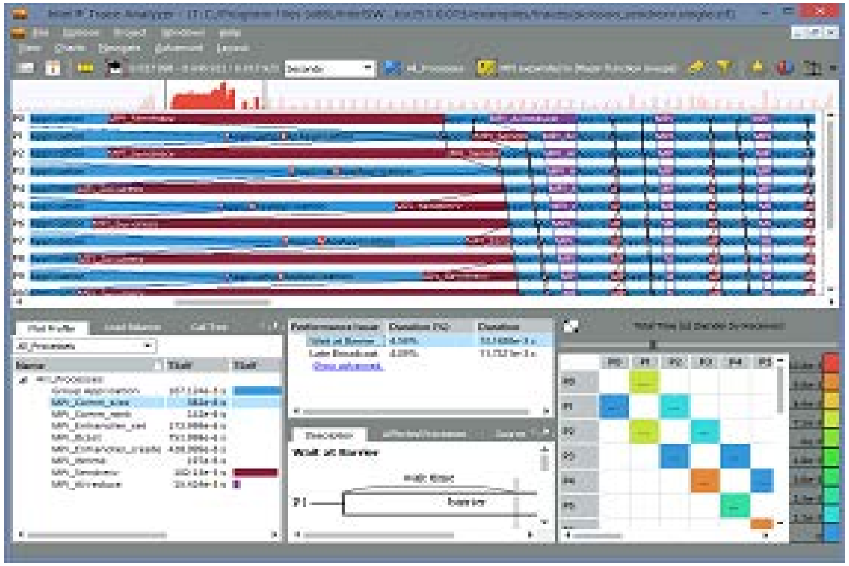
```
mpirun –trace –n 4 ./a.out
```

- Run visualizer

```
traceanalyzer a.out.stf &
```

- CHPC site

https://software.intel.com/en-us/get-started-with-itac-for-linux

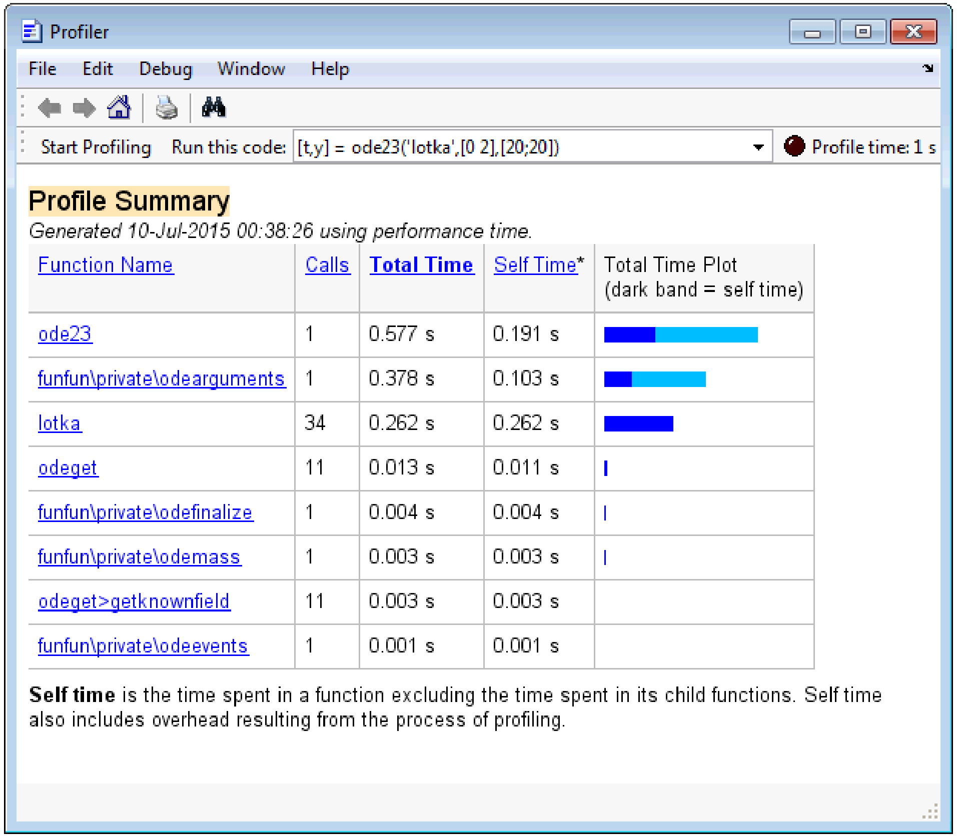


Interpreted languages profiling

- With increased use of interpreted languages, their performance is becoming important
- Matlab
  - Profiling ecosystem in the IDE
- Python
  - Python modules or IDEs
- R
  - Profiling libraries or RStudio

Matlab

- `profile` command turns on/off profiling
- Profile is then displayed in the IDE
- Click on each function to show line-by-line profile



```
[t,y] = ode23('lotka',[0 2],[20;20])
```

Profile Summary

Generated 10-Jul-2015 00:38:26 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
ode23	1	0.577 s	0.191 s	
funfun\private\odearguments	1	0.378 s	0.103 s	
lotka	34	0.262 s	0.262 s	
odeget	11	0.013 s	0.011 s	
funfun\private\odefinalize	1	0.004 s	0.004 s	
funfun\private\odemass	1	0.003 s	0.003 s	
odeget>getknownfield	11	0.003 s	0.003 s	
funfun\private\odeevents	1	0.001 s	0.001 s	

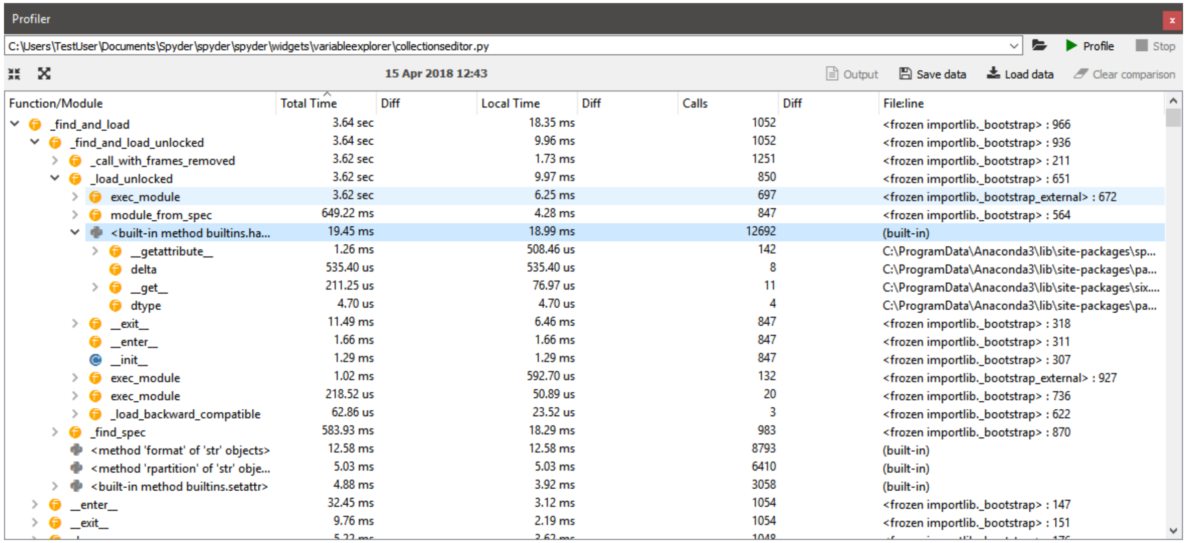
Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

- Performance improvement strategies

https://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

Python

- `profile` and `cProfile` modules
  - Text based output, optional format with `pstats` , analysis with `Stats`
- Plethora of other tools
  - E.g. line profiling with line_profiler
- Some IDEs display profiles
  - Spyder



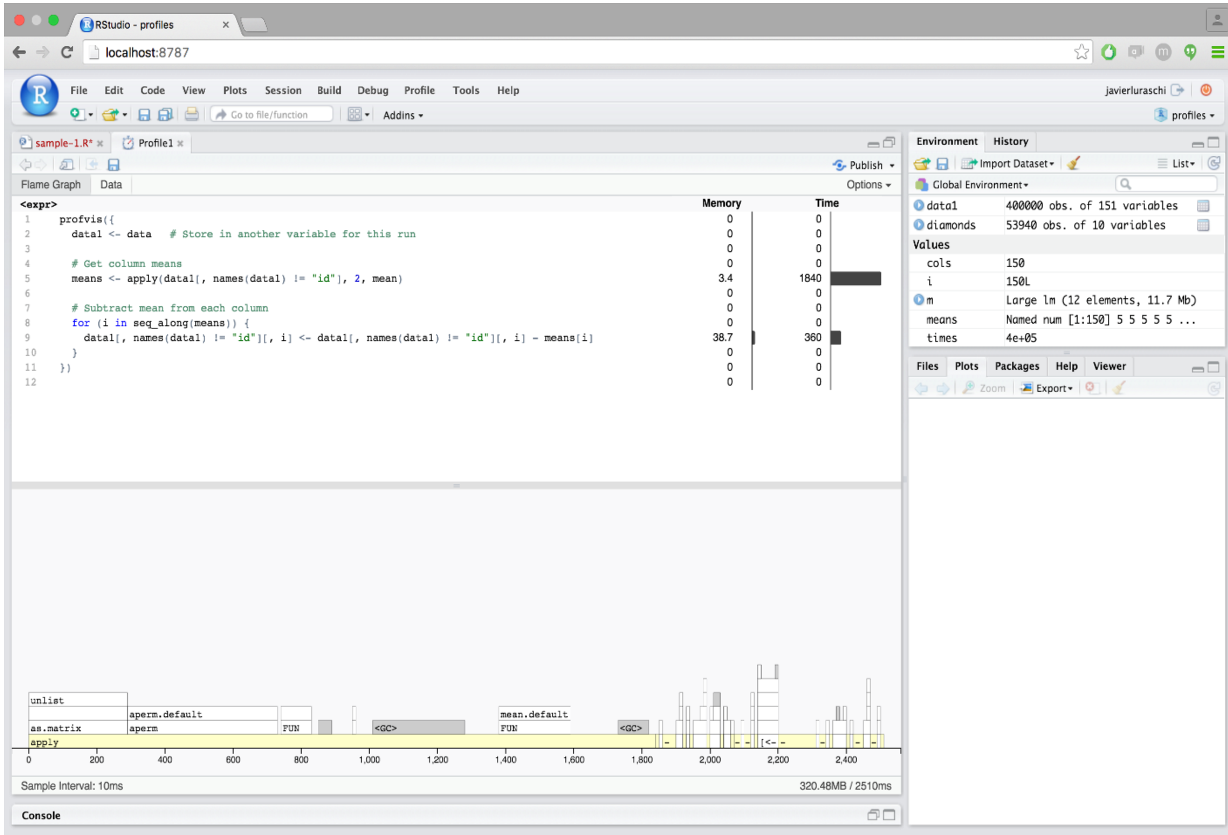
C:\Users\TestUser\Documents\Spyder\spyder\spyder\widgets\variableexplorer\collectionseditor.py

15 Apr 2018 12:43

Function/Module	Total Time	Diff	Local Time	Diff	Calls	Diff	File:line
_find_and_load	3.64 sec		18.35 ms		1052		<frozen importlib._bootstrap> : 966
_find_and_load_unlocked	3.64 sec		9.96 ms		1052		<frozen importlib._bootstrap> : 936
_call_with_frames_removed	3.62 sec		1.73 ms		1251		<frozen importlib._bootstrap> : 211
_load_unlocked	3.62 sec		9.97 ms		850		<frozen importlib._bootstrap> : 651
exec_module	3.62 sec		6.25 ms		697		<frozen importlib._bootstrap_external> : 672
module_from_spec	649.22 ms		4.28 ms		847		<frozen importlib._bootstrap> : 564
<built-in method builtins.ha...	19.45 ms		18.99 ms		12692		(built-in)
__getattribute__	1.26 ms		508.46 us		142		C:\ProgramData\Anaconda3\lib\site-packages\sp...
delta	535.40 us		535.40 us		8		C:\ProgramData\Anaconda3\lib\site-packages\pa...
__get__	211.25 us		76.97 us		11		C:\ProgramData\Anaconda3\lib\site-packages\six....
dtype	4.70 us		4.70 us		4		C:\ProgramData\Anaconda3\lib\site-packages\pa...
__exit__	11.49 ms		6.46 ms		847		<frozen importlib._bootstrap> : 318
__enter__	1.66 ms		1.66 ms		847		<frozen importlib._bootstrap> : 311
__init__	1.29 ms		1.29 ms		847		<frozen importlib._bootstrap> : 307
exec_module	1.02 ms		592.70 us		132		<frozen importlib._bootstrap_external> : 927
exec_module	218.52 us		50.89 us		20		<frozen importlib._bootstrap> : 736
_load_backward_compatible	62.86 us		23.52 us		3		<frozen importlib._bootstrap> : 622
_find_spec	583.93 ms		18.29 ms		983		<frozen importlib._bootstrap> : 870
<method 'format' of 'str' objects>	12.58 ms		12.58 ms		8793		(built-in)
<method 'rpartition' of 'str' obje...	5.03 ms		5.03 ms		6410		(built-in)
<built-in method builtins.setattr>	4.88 ms		3.92 ms		3058		(built-in)
__enter__	32.45 ms		3.12 ms		1054		<frozen importlib._bootstrap> : 147
__exit__	9.76 ms		2.19 ms		1054		<frozen importlib._bootstrap> : 151

R

- `Rprof` function to profile
- `summaryRprof` to display
- RStudio has a profile interface called profviz



<expr>	Memory	Time
`profvis({`	0	0
`  data1 <- data   # Store in another variable for this run`	0	0
	0	0
`  # Get column means`	0	0
`  means <- apply(data1[, names(data1) != "id"], 2, mean)`	3.4	1840
	0	0
`  # Subtract mean from each column`	0	0
`  for (i in seq_along(means)) {`	0	0
`    data1[, names(data1) != "id"][, i] <- data1[, names(data1) != "id"][, i] - means[i]`	38.7	360
`  }`	0	0
`})`	0	0
	0	0

Sample Interval: 10ms

320.48MB / 2510ms

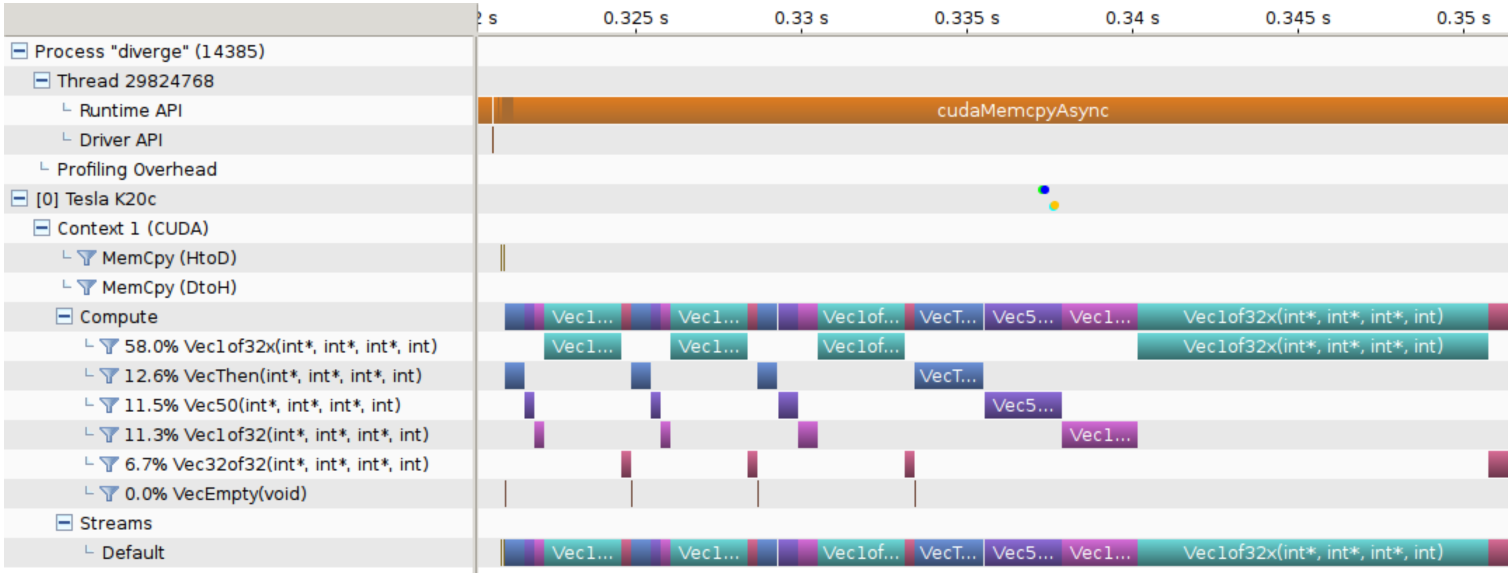
data1	400000 obs. of 151 variables
diamonds	53940 obs. of 10 variables
Values	
cols	150
i	150L
m	Large lm (12 elements, 11.7 Mb)
means	Named num [1:150] 5 5 5 5 5 ...
times	4e+05

- Performance improvement strategies

http://adv-r.had.co.nz/Profiling.html

GPU profiling

- Nvidia provides several tools
- Profilers shipping with CUDA (deprecated)
  - nvprof - text/line based
  - nvvp - visual profiler

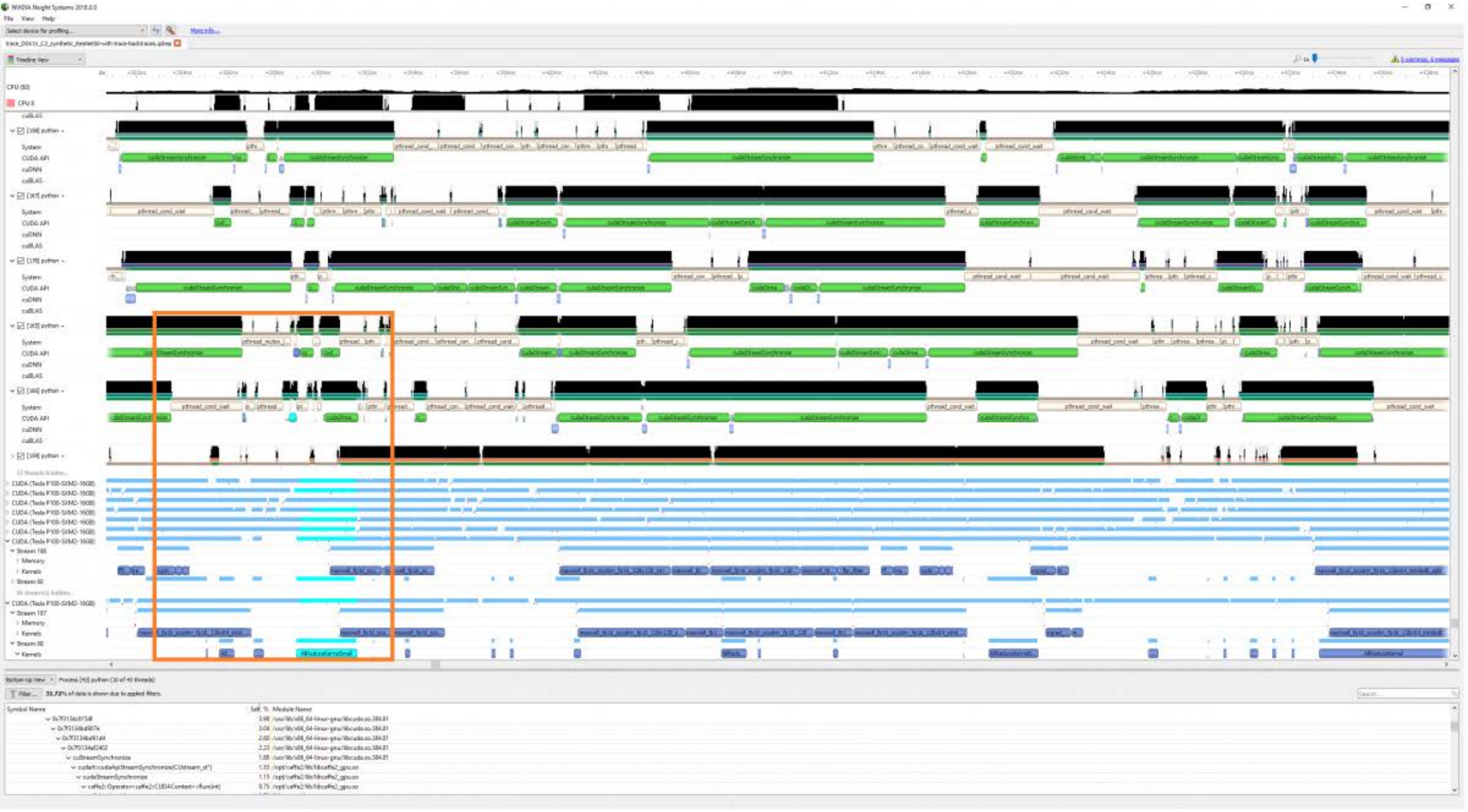

2 s
0.325 s
0.33 s
0.335 s
0.34 s
0.345 s
0.35 s
Process "diverge" (14385)
Thread 29824768
Runtime API
Driver API
Profiling Overhead
[0] Tesla K20c
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
58.0% Vec1of32x(int*, int*, int*, int)
12.6% VecThen(int*, int*, int*, int)
11.5% Vec50(int*, int*, int*, int)
11.3% Vec1of32(int*, int*, int*, int)
6.7% Vec32of32(int*, int*, int*, int)
0.0% VecEmpty(void)
Streams
Default
cudaMemcpyAsync
Vec1...
Vec1of...
VecT...
Vec5...
Vec1of32x(int*, int*, int*, int)

GPU profiling

- Using GPU hardware counters requires us to set up a SLURM reservation
  - there is a security issue with the hardware counters enabled
  - our admins will turn the counters on for the reservation only
  - nvprof -m all ./myprogram
  - more details at https://developer.nvidia.com/nvidia-development-tools-solutions-ERR_NVGPUCTRPERM-permission-issue-performance-counters

GPU profiling

- Nvidia Nsight Systems
  - nsight-sys, profiles CUDA, OpenGL, NVTX, pthreads
  - ships with CUDA but newer version available



31.72% of data is shown due to applied filters.

Symbol Name	Self, %	Module Name
0x7f3134c8f54f	3.98	/usr/lib/x86_64-linux-gnu/libcuda.so.384.81
0x7f3134bd907e	3.04	/usr/lib/x86_64-linux-gnu/libcuda.so.384.81
0x7f3134bd91d4	2.60	/usr/lib/x86_64-linux-gnu/libcuda.so.384.81
0x7f3134af2402	2.23	/usr/lib/x86_64-linux-gnu/libcuda.so.384.81
cuStreamSynchronize	1.88	/usr/lib/x86_64-linux-gnu/libcuda.so.384.81
cudart::cudaApiStreamSynchronize(CUstream_st*)	1.33	/opt/caffe2/lib/libcaffe2_gpu.so
cudaStreamSynchronize	1.15	/opt/caffe2/lib/libcaffe2_gpu.so
caffe2::Operator<caffe2::CUDAContext>::Run(int)	0.75	/opt/caffe2/lib/libcaffe2_gpu.so

Summary

- Serial profilers
  - gprof, perf
- Intel tools
  - VTune, AdvisorXE, ITAC
- Interpreted languages profiling
  - Matlab profile
  - Python profile, Cprofile
  - R Rprof, profviz
- GPU profiling
  - nvprof, nvcc - older
  - nsight-sys - current