Profiling and Understanding DL Workloads on Supercomputing Systems

Murali Emani,
Data Science group, ALCF
memani@anl.gov

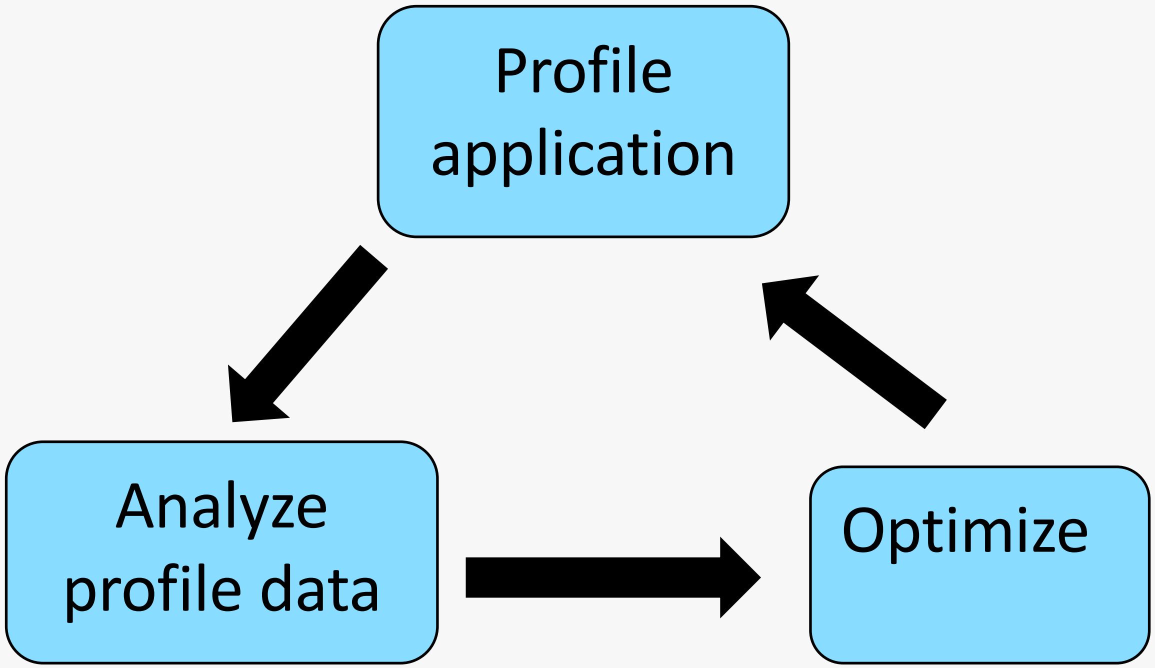
Introduction

- Profiling is an approach to measure application performance

- Simple Profiling:
  - How long does an application take
- Advanced Profiling:
  - Why does an operation take long time

- Goal: Find performance bottlenecks
  - inefficient programming
  - memory I/O bottlenecks
  - parallel scaling

Typical Optimization Workflow



Iterative workflow till desired performance is reached

Broad classification

- Hardware counters
  count events from CPU/GPU perspective (#flops, memory loads, etc.)
  usually needs Linux kernel module installed or root permission

- Statistical profilers (sampling)
  interrupt program at given intervals to find the state of a program

- Event based profilers (tracing)
  collect information on each function call

Plethora of Tools

- Cprofile
- Gprof
- Perf tool
- Intel Vtune
- HPCToolKit
- OpenSpeedShop
- TAU
- Nvidia Nvprof, Nsight

....

...

Profiling DNN workloads

- Critical to understand workload performance

- Machine learning and deep learning models are implemented on a variety of hardware

- Most applications are written in Python using standard ML frameworks

- The frameworks generate kernels based on hardware and customized installation and libraries (MKL-DNN, CuDNN etc.)

Challenges

- Profiling is hard, cumbersome and time-consuming
- Profiling tools generate lot of data and hard to understand
- The problem is further compounded with large, complex models with large volumes of data
- Need strategies to use right tools and detailed insights to how to analyze the profile data

Profiling on Nvidia GPUs

Profiling on Nvidia GPUs

Use Nvidia profiler ‘**Nvprof**’

- capture metrics from hardware counters
- invoked via command line or UI (Nvidia Visual Profiler NVVP)

See list of options using

`nvprof –h`

Some useful options:

`-o`: create output file to import into nvvp

`--metrics / -m` : collect metrics

`--events / -e` : collect events

`--log-file` : create human readable output file

`--analysis-metrics` : collect all metrics to import into nvvp

`--query-metrics/--query-events`: list of available metrics/events

Events and Metrics

- An **event** is a countable activity, action, or occurrence on a device. It corresponds to a single hardware counter value which is collected during kernel execution
- A **metric** is a characteristic of an application that is calculated from one or more event values

In general, events are only for experts, rarely used.

- Vary in number based on hardware family (P100, K80, V100 etc)
- For example, on V100, nvprof gives 175 metrics
- Event and metric values are aggregated across all units in the GPU.

Workflow

Option 1)

- Use ‘**nvprof’** to collect metrics in an output file (compute node)
- Use ‘**nvvp**’ to visualize the profile (login node)

Option 2)

- Directly launch **nvvp** on compute node and profile the code interactively

```
export PATH=/soft/compilers/cuda/cuda-9.1.85/bin:$PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/soft/compilers/cuda/cuda-
9.1.85/lib64
```

Profile Commands

– Kernel timing analysis:

```
nvprof --log-file timing.log <myapp>
nvprof --log-file timing.log python myapp.py args
```

– Traces (#threads, #warps, #registers)

```
nvprof --print-gpu-traces --log-file traces.log <myapp>
```

Profile Commands

– Kernel timing analysis:

```
nvprof --log-file timing.log <myapp>
nvprof --log-file timing.log python myapp.py args
```

– Traces (#threads, #warps, #registers)

```
nvprof --print-gpu-traces --log-file traces.log <myapp>
```

– Get all metrics for all kernels

```
nvprof --metrics all --log-file all-metrics.log <myapp>
```

– Get metrics for guided analysis

```
nvprof --analysis-metrics –o analysis.nvprof <myapp>
```

– Visual profile to use Nvidia Visual Profiler (nvvp)

```
nvprof –o analysis.nvprof <myapp>
```

Selective Profiling

- As profiling adds significant overhead, a better strategy is to profile only regions of interest (kernels and metrics)

- All metrics for kernels of interest:

```
nvprof --profile-from-start off --kernels <kernel-name> --metrics all
--log-file selective-profile.log <myapp>
```

- few metrics for kernels of interest

```
nvprof --profile-from-start off--kernels <kernel-name> --metrics ipc
--log-file selective-profile.log <myapp>
```

For example, if we want to profile heavy kernels only

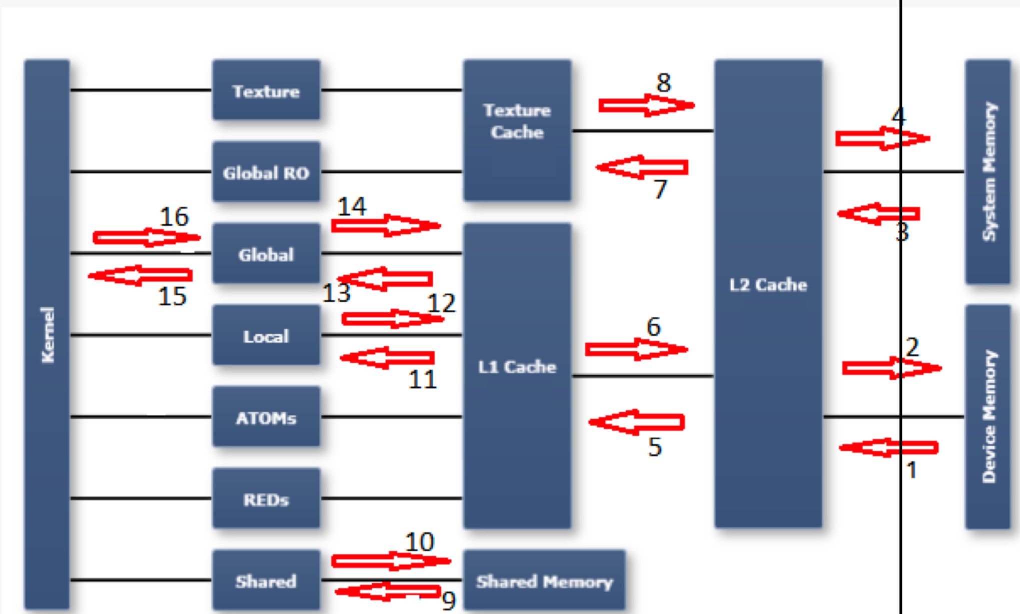
Step 1) use nvprof to list all kernels sorted by the time

Step 2) re-run nvprof in selective profiling mode

- Profile GEMM kernels

```
nvprof --profile-from-start off --kernels “::gemm:n” --metrics all
--log-file selective-profile.log <myapp>
```

GPU Memory - metrics



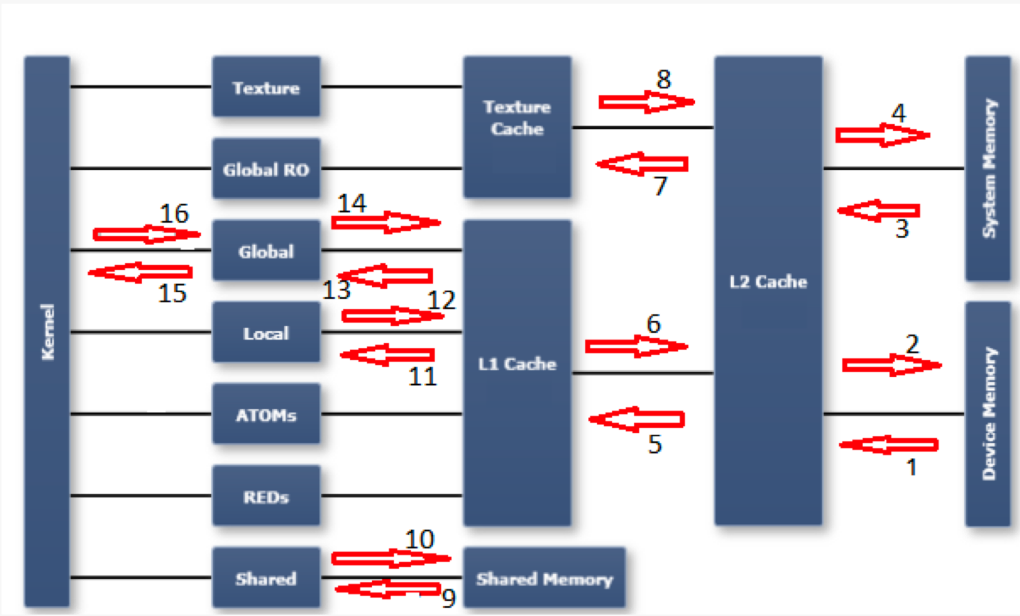
on-chip memory

off-chip memory

GPU Memory hierarchy

https://stackoverflow.com/questions/37732735/nvprof-option-for-bandwidth

GPU Memory - metrics

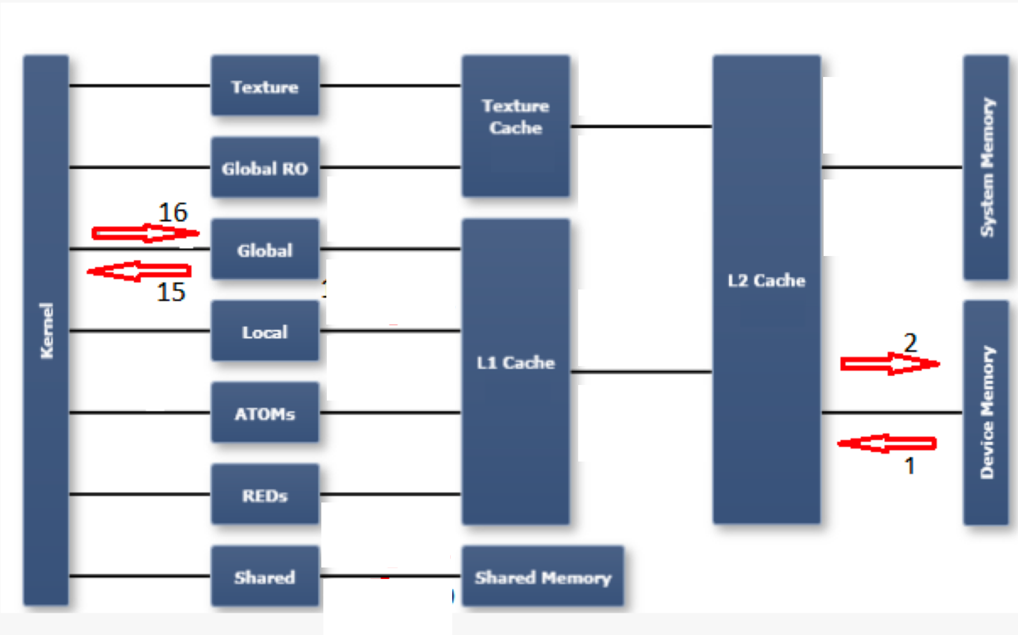


GPU Memory hierarchy

1.dram_read_throughput, dram_read_transactions
2.dram_write_throughput, dram_write_transactions
3.sysmem_read_throughput, sysmem_read_transactions
4.sysmem_write_throughput, sysmem_write_transaction
5.l2_l1_read_transactions, l2_l1_read_throughput
6.l2_l1_write_transactions, l2_l1_write_throughput
7.l2_tex_read_transactions, l2_texture_read_throughput
8.texture is read-only, there are no transactions possible o
this path
9.shared_load_throughput, shared_load_transactions
10.shared_store_throughput, shared_store_transactions
11.l1_cache_local_hit_rate
12.l1 is write-through cache, so there are no (independen
metrics for this path - refer to other local metrics
13.l1_cache_global_hit_rate
14.see note on 12
15.gld_efficiency, gld_throughput, gld_transactions
16.gst_efficiency, gst_throughput, gst_transactions

https://stackoverflow.com/questions/37732735/nvprof-option-for-bandwidth

GPU Memory - metrics



GPU Memory

1.dram_read_throughput, dram_read_transactions
2.dram_write_throughput, dram_write_transactions
3.sysmem_read_throughput, sysmem_read_transactions
4.sysmem_write_throughput, sysmem_write_transaction
5.l2_l1_read_transactions, l2_l1_read_throughput
6.l2_l1_write_transactions, l2_l1_write_throughput
7.l2_tex_read_transactions, l2_texture_read_throughput
8.texture is read-only, there are no transactions possible o this path
9.shared_load_throughput, shared_load_transactions
10.shared_store_throughput, shared_store_transactions
11.l1_cache_local_hit_rate
12.l1 is write-through cache, so there are no (independen metrics for this path - refer to other local metrics
13.l1_cache_global_hit_rate
14.see note on 12
15.gld_efficiency, gld_throughput, gld_transactions
16.gst_efficiency, gst_throughput, gst_transactions

https://stackoverflow.com/questions/37732735/nvprof-option-for-bandwidth

Metrics and Events

Metrics relevant to identify compute, memory, IO characteristics

achieved_occupancy	ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
ipc	Instructions executed per cycle
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage.
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage.
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10

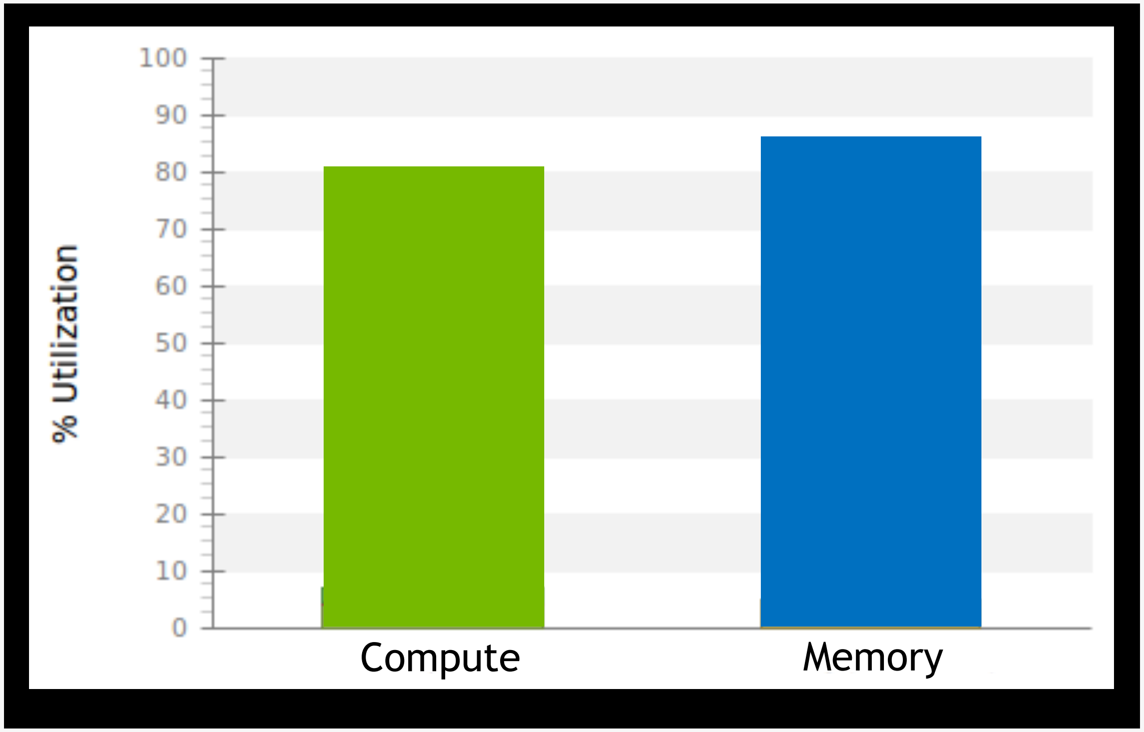
Metrics and Events

Metrics relevant to identify compute, memory, IO characteristics

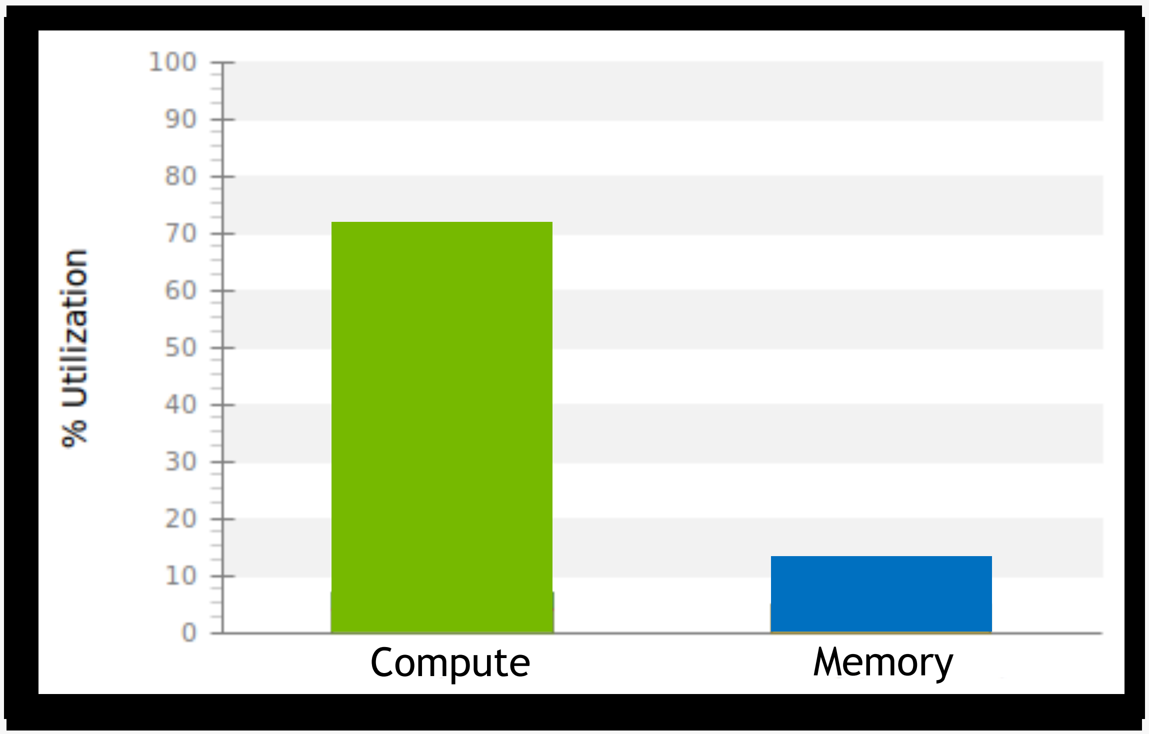
achieved_occupancy	ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
ipc	Instructions executed per cycle
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage.
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage.
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10

Warps efficiency/active cycles

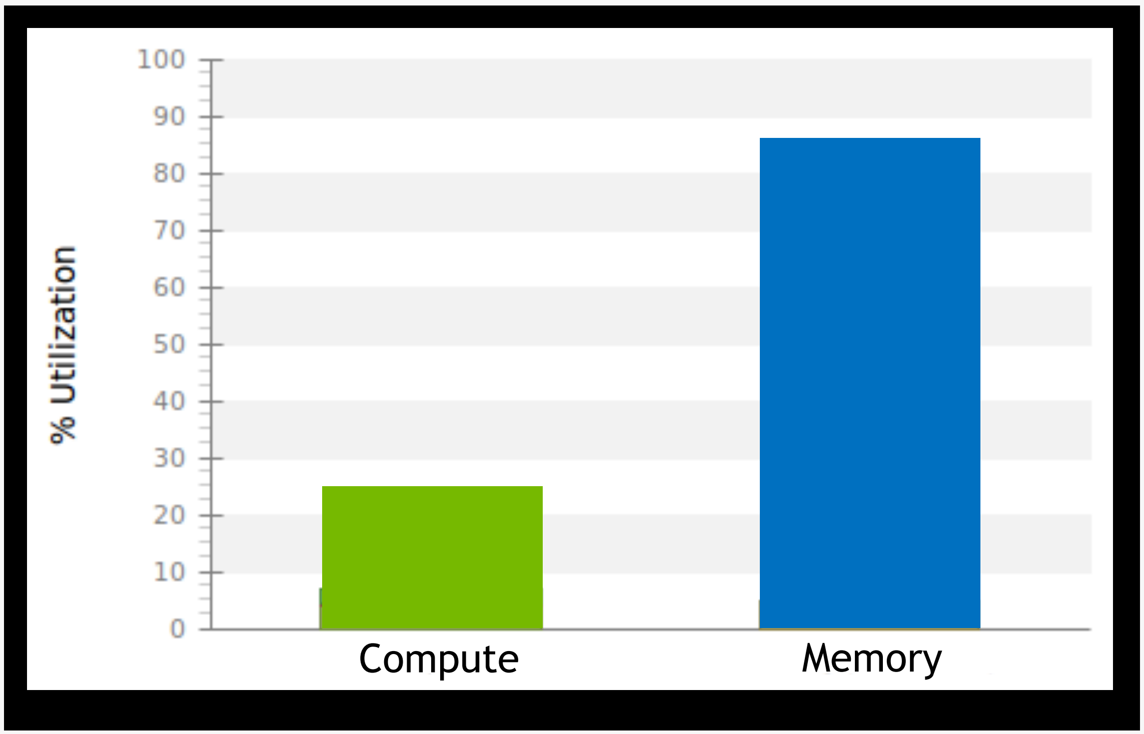
Both high compute and memory highly utilized



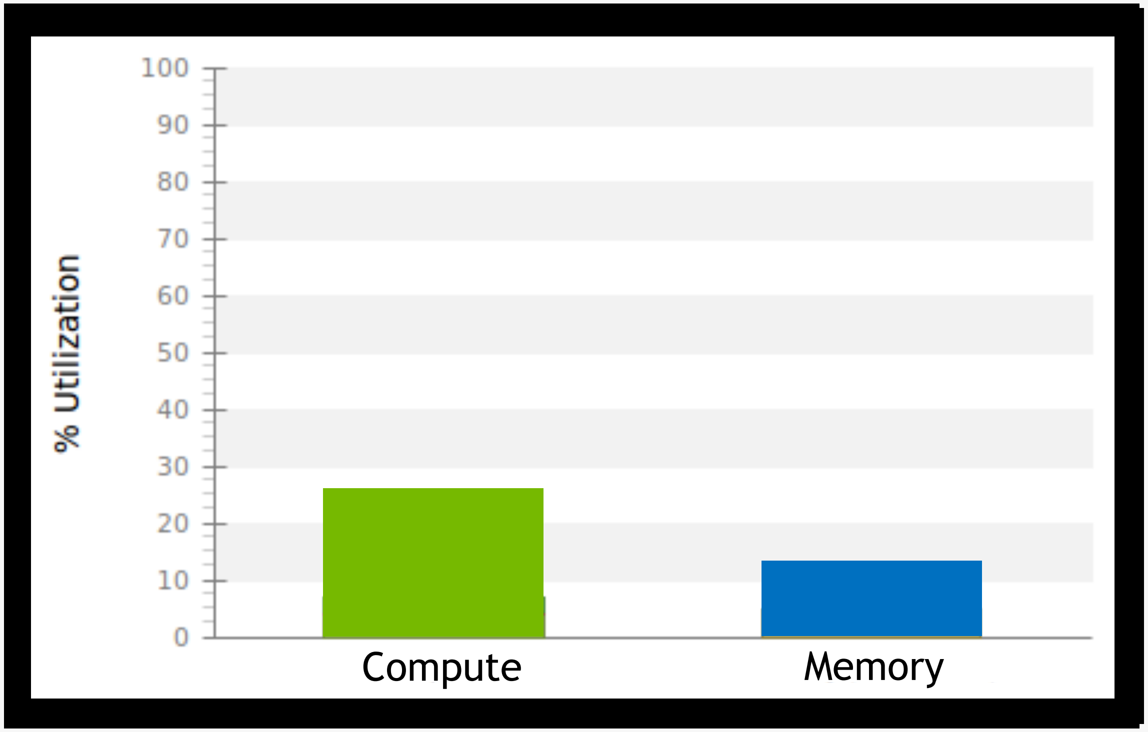
High compute, low memory utilization => compute bound



Low compute, high memory utilization => memory bound

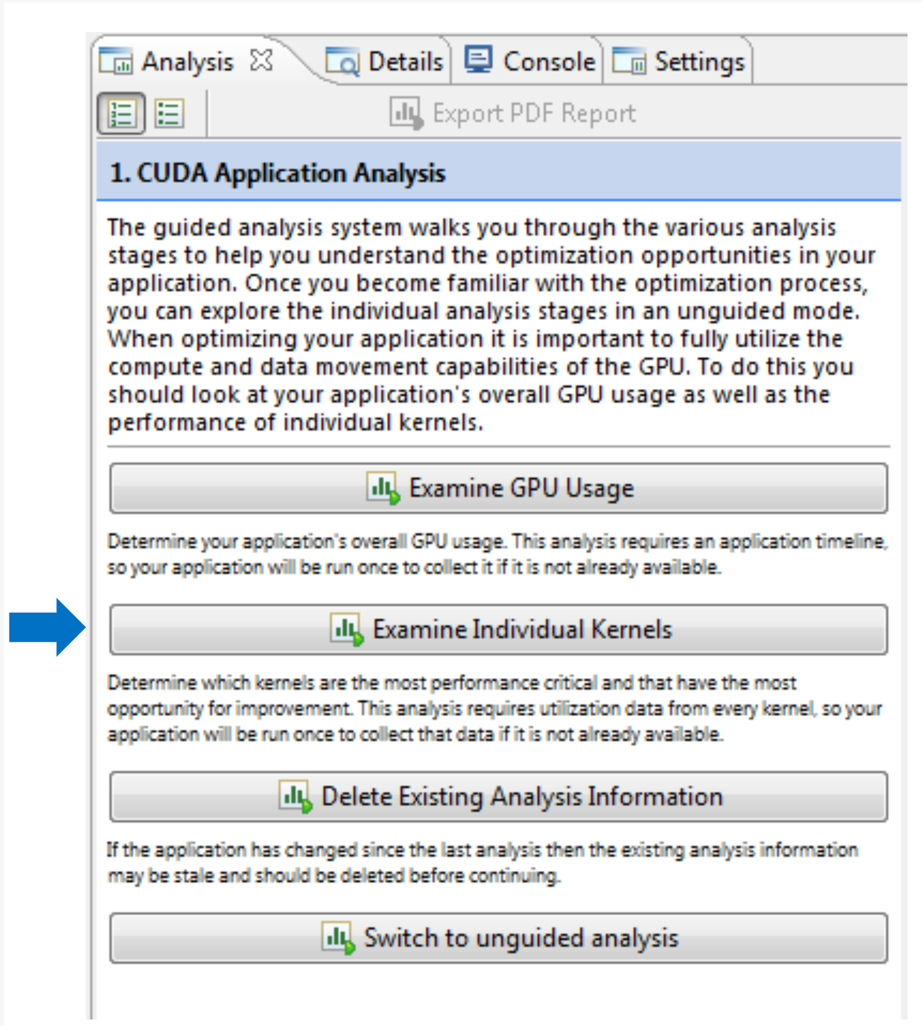


Both low => latency bound



Detailed Analysis

Use visual profiler nvvp



i Kernel Optimization Priorities

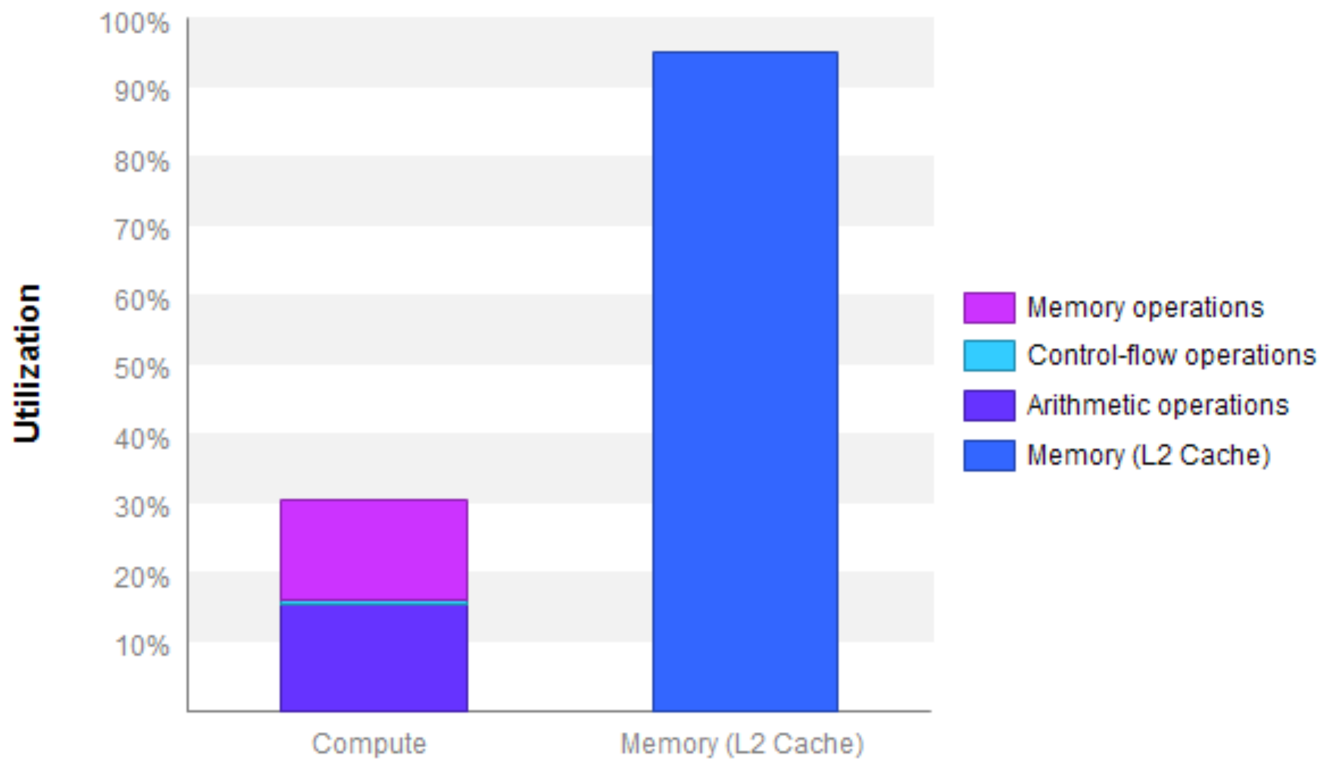
The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[2 kernel instances] maxwell_sgemm_128x64_tn
1	[1 kernel instances] elementWise(float*, float*, float*, float*, float*, float*)

Perform Kernel Analysis

i Kernel Performance Is Bound By Memory Bandwidth

For device "Quadro M6000" the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the L2 Cache memory.



Tips

- Start with the nvprof output
- Perform deeper analysis only if a kernel takes significant amount of execution time.
- Know your hardware:
  - If your GPU can do 6 TFLOPs, and you’re already doing 5.5 TFLOPs, you won’t go much faster!
- Sometimes quite simple changes can lead to big improvements in performance

Example

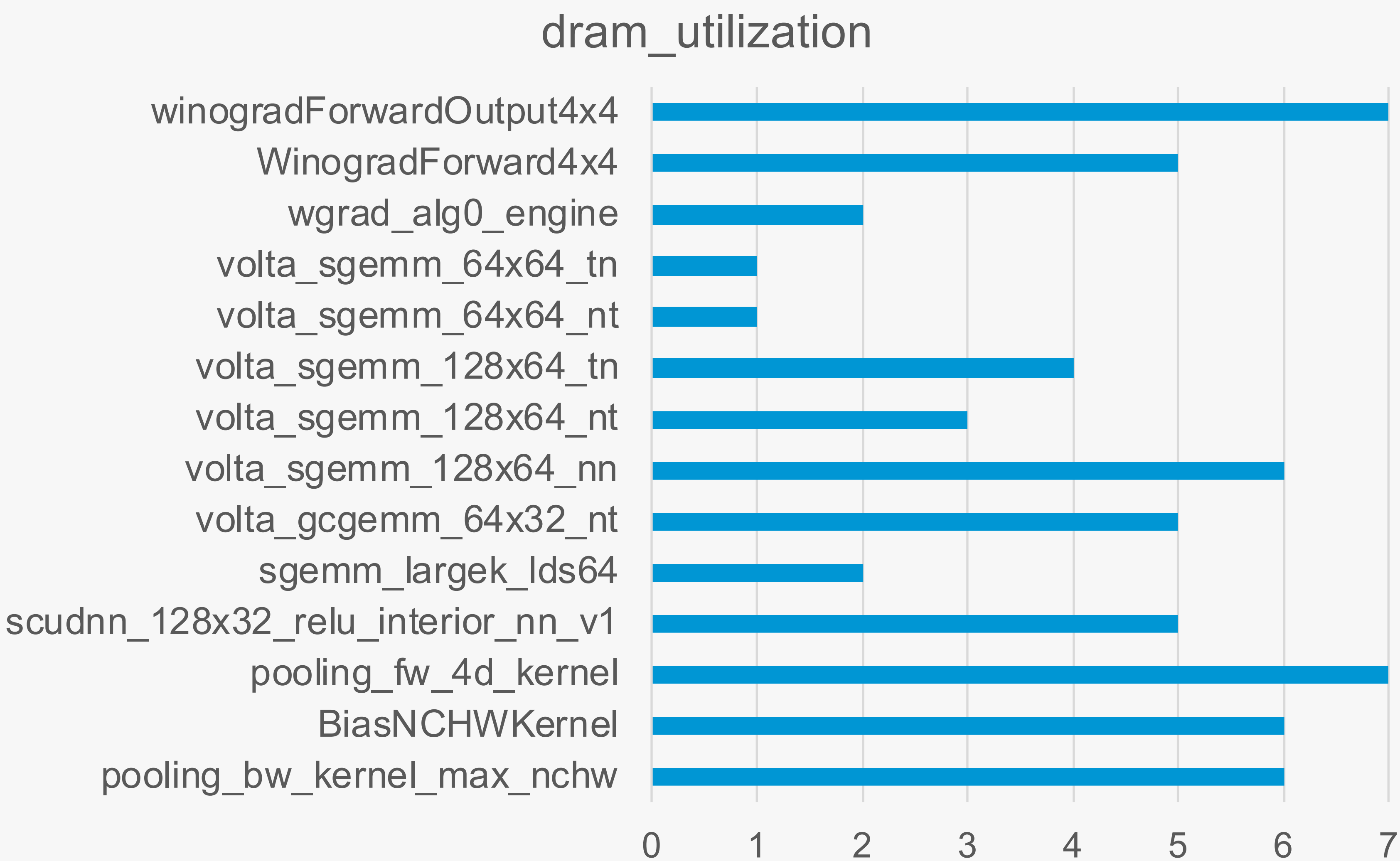
Simple CNN in Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu’))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax’))

model.compile(…....)

model.fit(.....)

model.evaluate(.....)
```

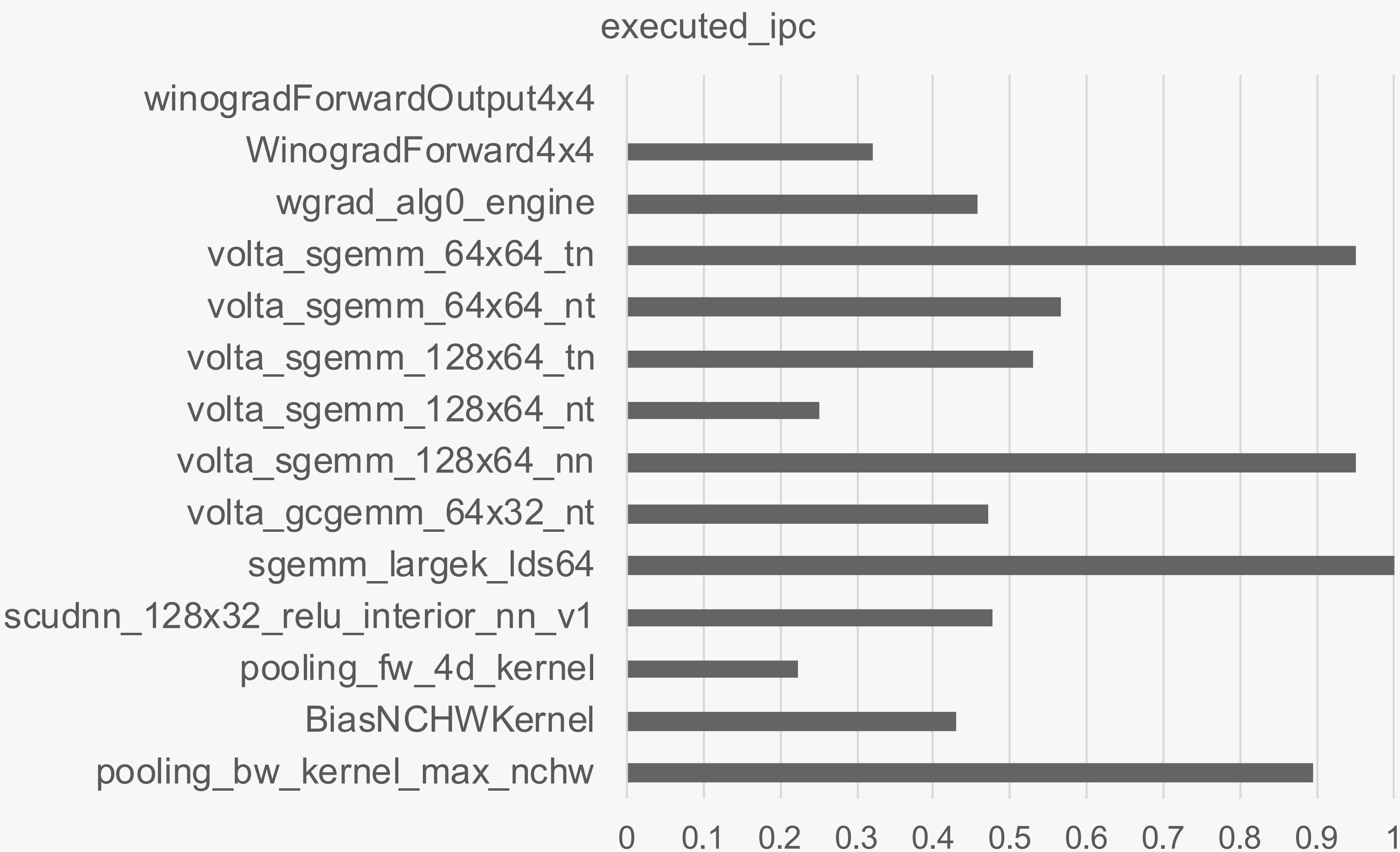
Example

Simple CNN in Keras

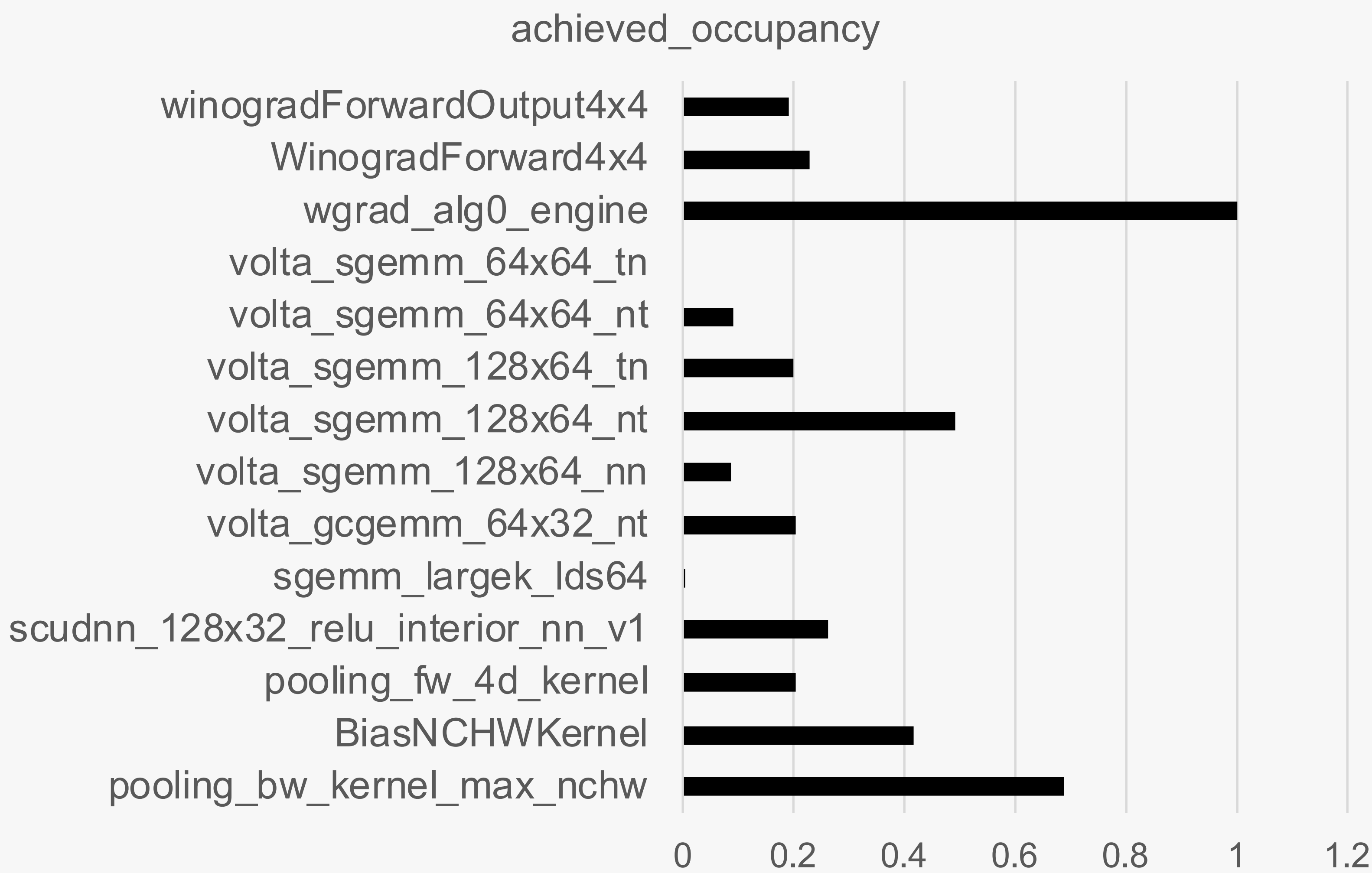
```
import numba.cuda

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu’))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax’))

model.compile(…....)

## begin cuda profile
cuda.profile_start()

model.fit(.....)

## stop cuda profile
cuda.profile_stop()

model.evaluate(.....)
```

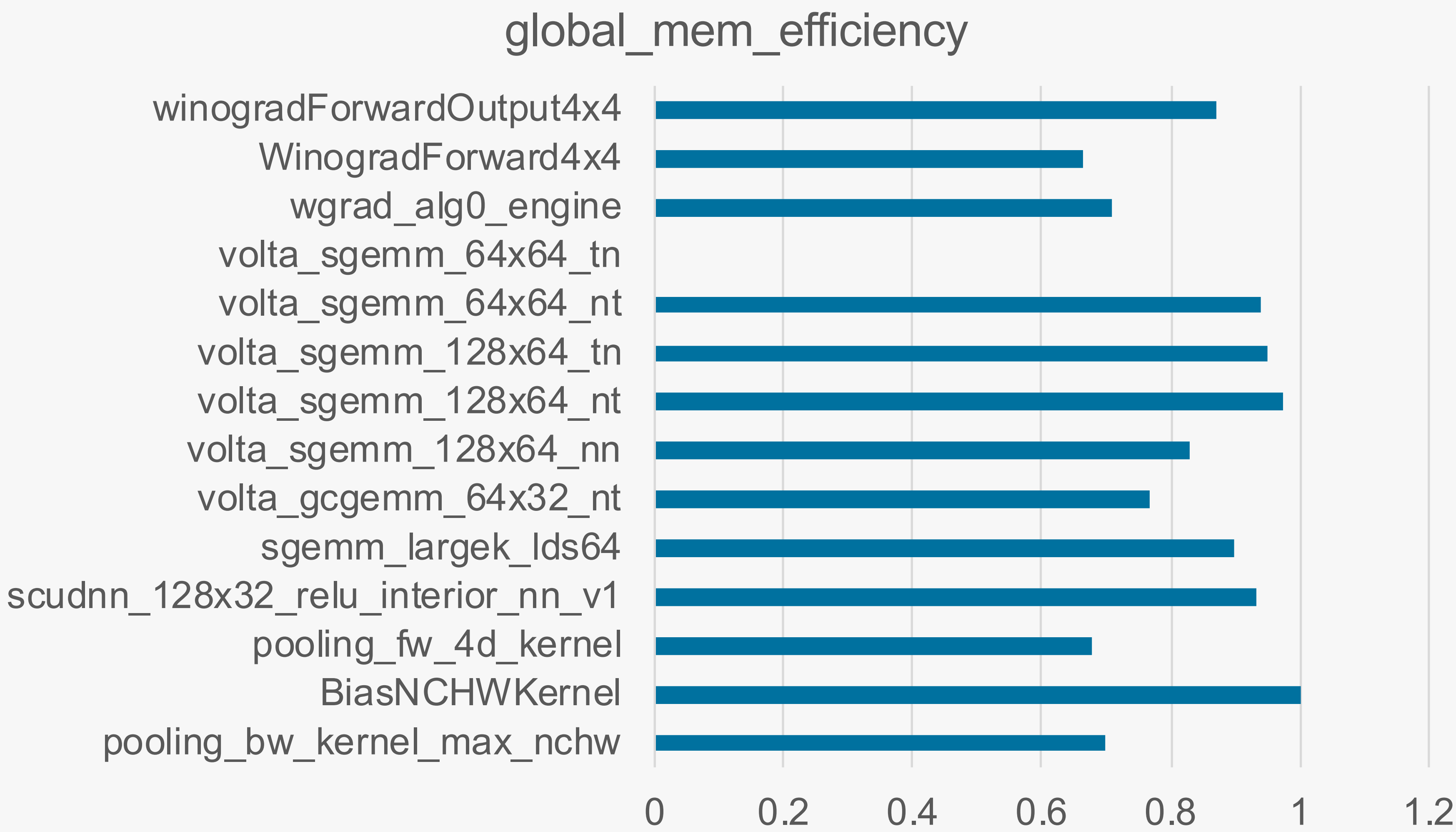


dram_utilization
winogradForwardOutput4x4
WinogradForward4x4
wgrad_alg0_engine
volta_sgemm_64x64_tn
volta_sgemm_64x64_nt
volta_sgemm_128x64_tn
volta_sgemm_128x64_nt
volta_sgemm_128x64_nn
volta_gcgemm_64x32_nt
sgemm_largek_lds64
scudnn_128x32_relu_interior_nn_v1
pooling_fw_4d_kernel
BiasNCHWKernel
pooling_bw_kernel_max_nchw
0
1
2
3
4
5
6
7

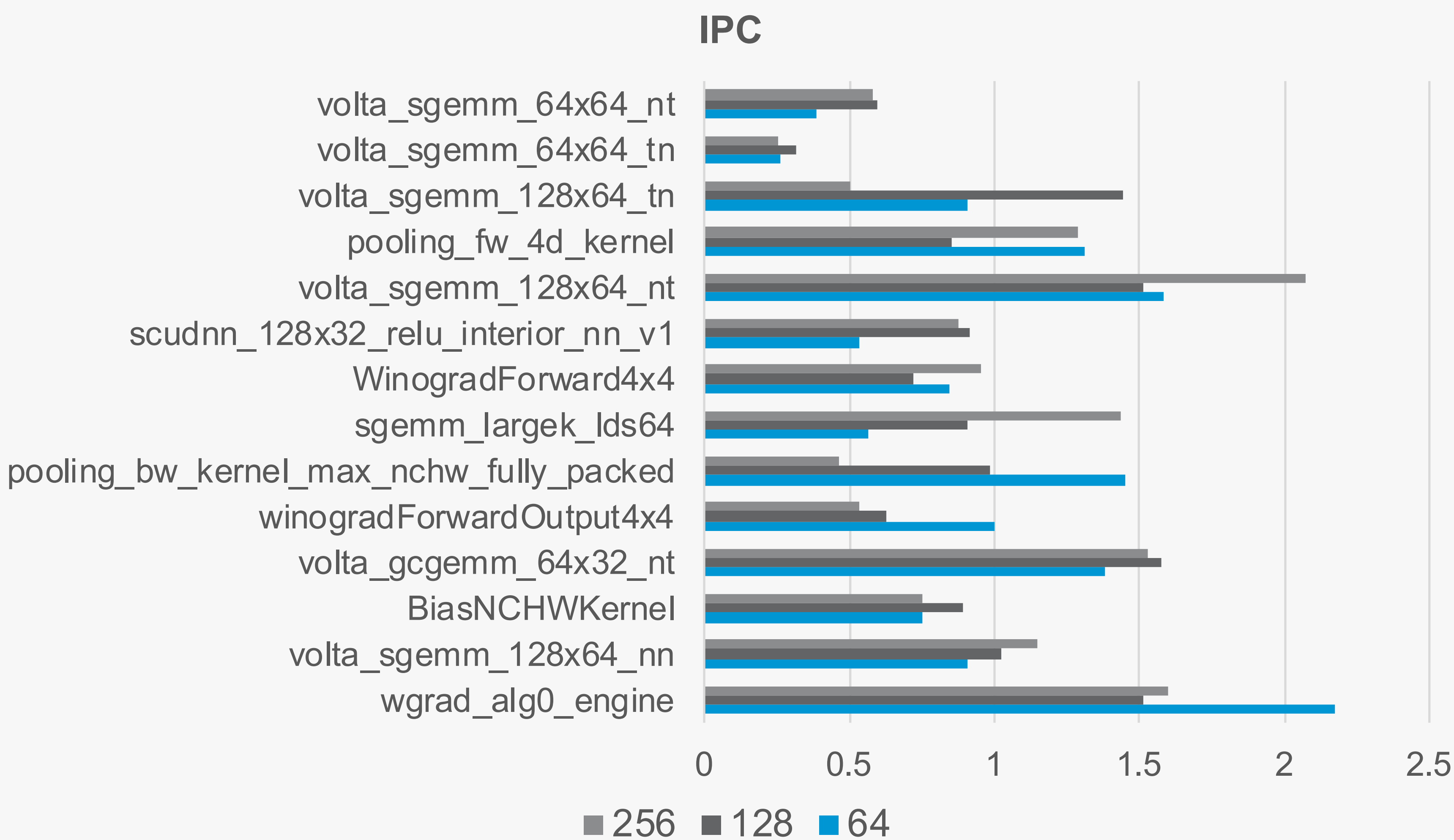


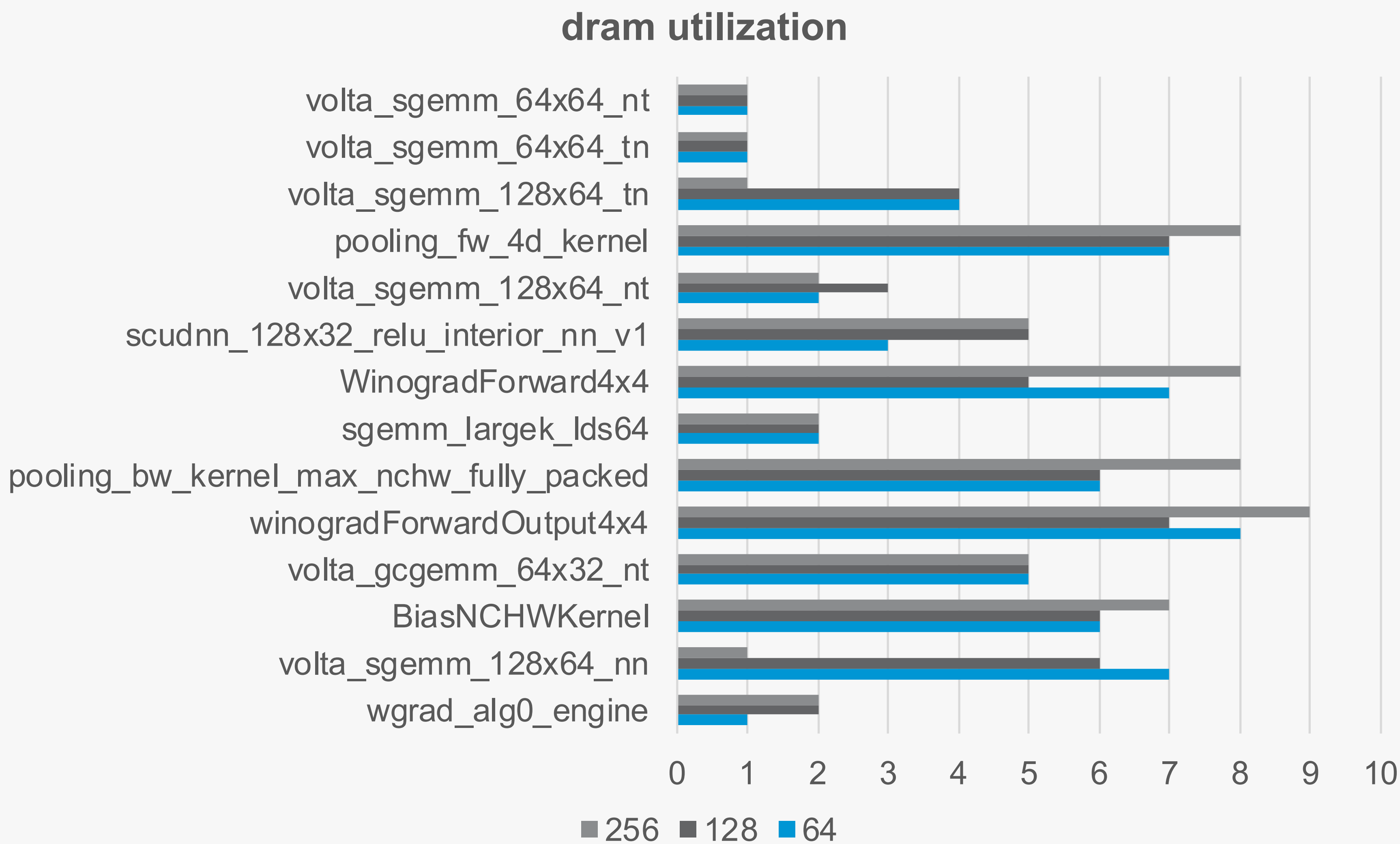
executed_ipc
winogradForwardOutput4x4
WinogradForward4x4
wgrad_alg0_engine
volta_sgemm_64x64_tn
volta_sgemm_64x64_nt
volta_sgemm_128x64_tn
volta_sgemm_128x64_nt
volta_sgemm_128x64_nn
volta_gcgemm_64x32_nt
sgemm_largek_lds64
scudnn_128x32_relu_interior_nn_v1
pooling_fw_4d_kernel
BiasNCHWKernel
pooling_bw_kernel_max_nchw
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


achieved_occupancy
winogradForwardOutput4x4
WinogradForward4x4
wgrad_alg0_engine
volta_sgemm_64x64_tn
volta_sgemm_64x64_nt
volta_sgemm_128x64_tn
volta_sgemm_128x64_nt
volta_sgemm_128x64_nn
volta_gcgemm_64x32_nt
sgemm_largek_lds64
scudnn_128x32_relu_interior_nn_v1
pooling_fw_4d_kernel
BiasNCHWKernel
pooling_bw_kernel_max_nchw
0
0.2
0.4
0.6
0.8
1
1.2

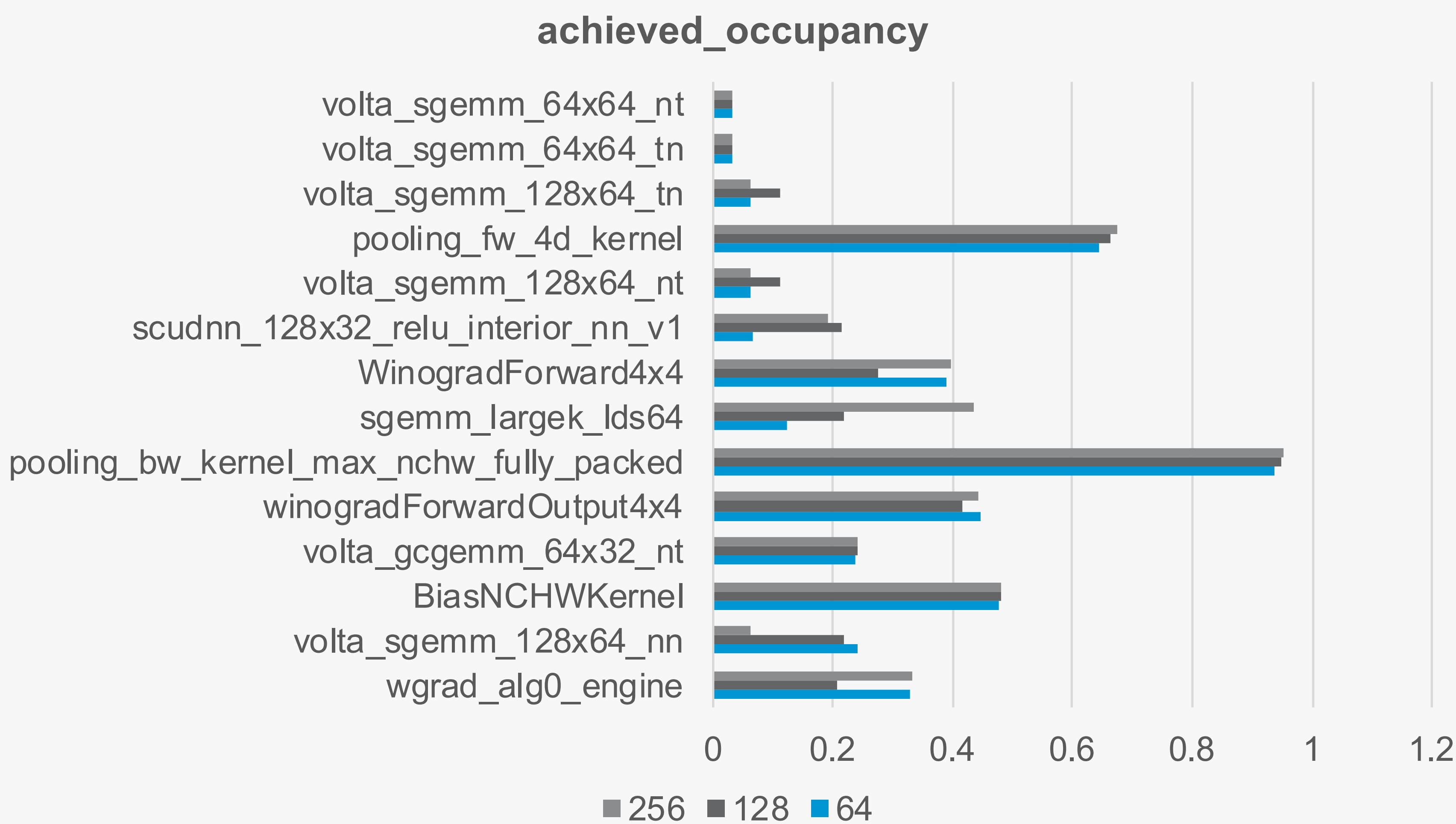

global_mem_efficiency
winogradForwardOutput4x4
WinogradForward4x4
wgrad_alg0_engine
volta_sgemm_64x64_tn
volta_sgemm_64x64_nt
volta_sgemm_128x64_tn
volta_sgemm_128x64_nt
volta_sgemm_128x64_nn
volta_gcgemm_64x32_nt
sgemm_largek_lds64
scudnn_128x32_relu_interior_nn_v1
pooling_fw_4d_kernel
BiasNCHWKernel
pooling_bw_kernel_max_nchw
0
0.2
0.4
0.6
0.8
1
1.2

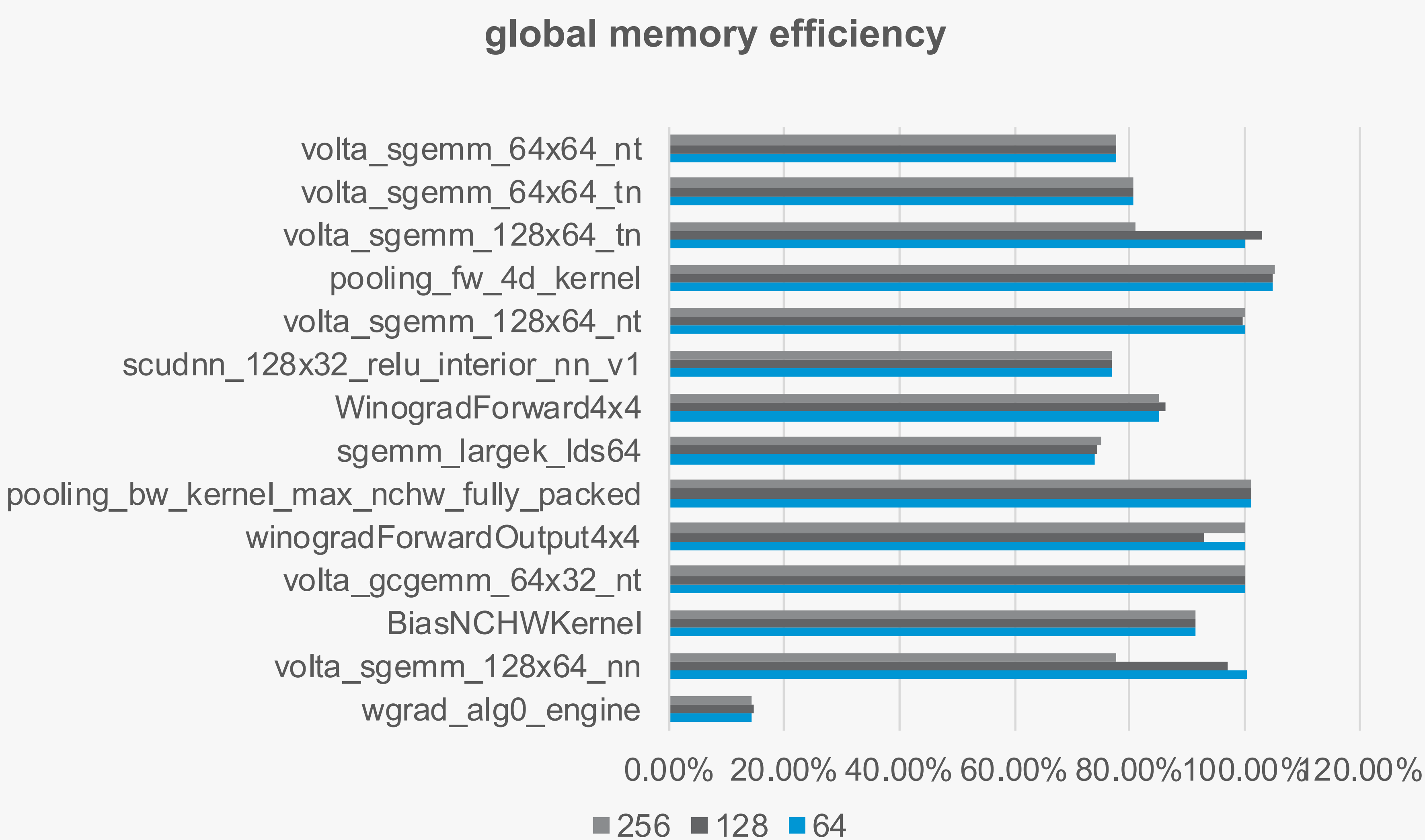
Impact of batch size





Argonne NATIONAL LABORATORY


achieved_occupancy
volta_sgemm_64x64_nt
volta_sgemm_64x64_tn
volta_sgemm_128x64_tn
pooling_fw_4d_kernel
volta_sgemm_128x64_nt
scudnn_128x32_relu_interior_nn_v1
WinogradForward4x4
sgemm_largek_lds64
pooling_bw_kernel_max_nchw_fully_packed
winogradForwardOutput4x4
volta_gcgemm_64x32_nt
BiasNCHWKernel
volta_sgemm_128x64_nn
wgrad_alg0_engine
0
0.2
0.4
0.6
0.8
1
1.2
256
128
64

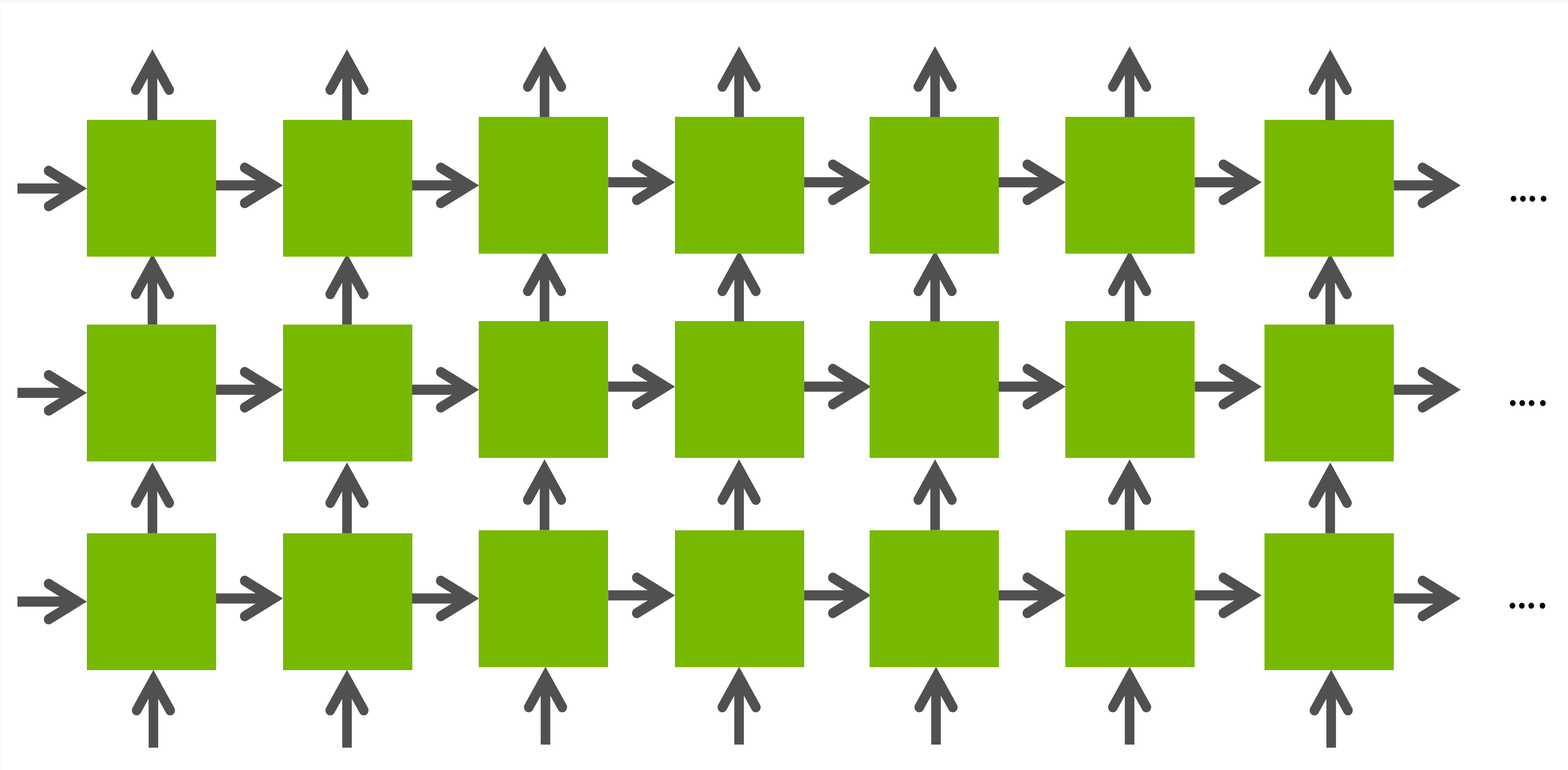


Argonne NATIONAL LABORATORY

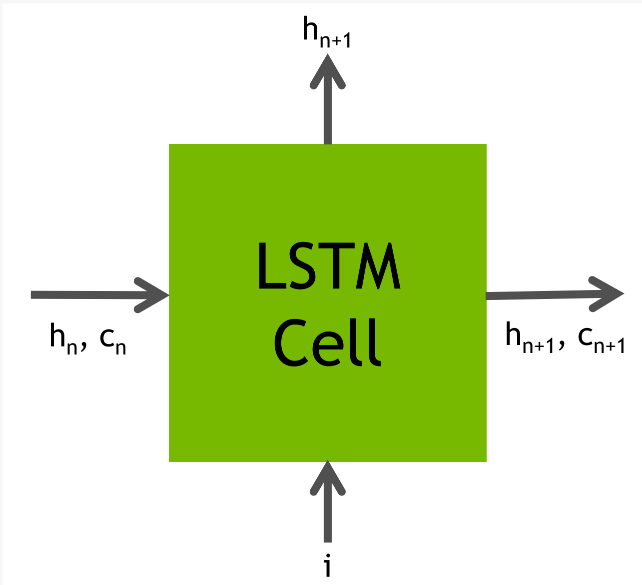
Example

LSTM – Long Short Term Memory

Recurrent Neural Network with potential for long-term memory



https://www.robots.ox.ac.uk/seminars/Extra/2015_10_08_JeremyAppleyard.pdf



hidden layer size = 512
minibatch size = 64

```
nvprof ./LSTM 512 64
```

```
==22964== NVPROF is profiling process 22964, command: ./LSTM 512 64
==22964== Profiling application: ./LSTM 512 64
==22964== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 93.93%  575.72us         8  71.964us  70.241us  78.945us  maxwell_sgemm_128x64_tn
  3.60%  22.080us         8  2.7600us  2.3360us  3.5840us  addBias(float*, float*)
  1.43%  8.7680us         4  2.1920us  2.0800us  2.4640us  vecAdd(float*, float*, float*)
  1.04%  6.3680us         1  6.3680us  6.3680us  6.3680us  nonLin(float*, float*, float*, float*)

==28493== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 97.04%  103.55ms        21  4.9308ms  10.606us  103.30ms  cudaLaunch
  2.08%  2.2189ms       249  8.9110us     202ns  350.88us  cuDeviceGetAttribute
  0.53%  568.27us        21  27.060us     286ns  557.64us  cudaConfigureCall
  0.17%  176.23us         3  58.741us  57.818us  59.862us  cuDeviceTotalMem
  0.14%  147.11us         3  49.036us  46.468us  52.966us  cuDeviceGetName
  0.04%  42.216us       128     329ns     240ns  5.1400us  cudaSetupArgument
  0.00%  4.3100us         8     538ns     354ns  1.7220us  cudaGetLastError
  0.00%  3.7640us         2  1.8820us     308ns  3.4560us  cuDeviceGetCount
  0.00%  1.8660us         6     311ns     204ns     648ns  cuDeviceGet
```

Optimization

$$[A_1][h] = [x_1]$$
$$[A_2][h] = [x_2]$$
$$[A_3][h] = [x_3]$$
$$[A_4][h] = [x_4]$$

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} [h] = \begin{bmatrix} \\ x \\ \\ \end{bmatrix}$$

Combine many small data transfers to few large data transfers

Optimization

$$[A_1][h] = [x_1]$$
$$[A_2][h] = [x_2]$$
$$[A_3][h] = [x_3]$$
$$[A_4][h] = [x_4]$$

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} [h] = \begin{bmatrix} \\ x \\ \\ \end{bmatrix}$$

Time(%)	Time	Calls	Avg	Min	Max	Name
93.93%	**575.72us**	8	71.964us	70.241us	78.945us	maxwell_sgemm_128x64_tn

Time(%)	Time	Calls	Avg	Min	Max	Name
84.40%	**198.11us**	2	99.057us	98.177us	99.937us	maxwell_sgemm_128x64_tn

2.5x performance gain

Profiling on CPUs using Intel Vtune

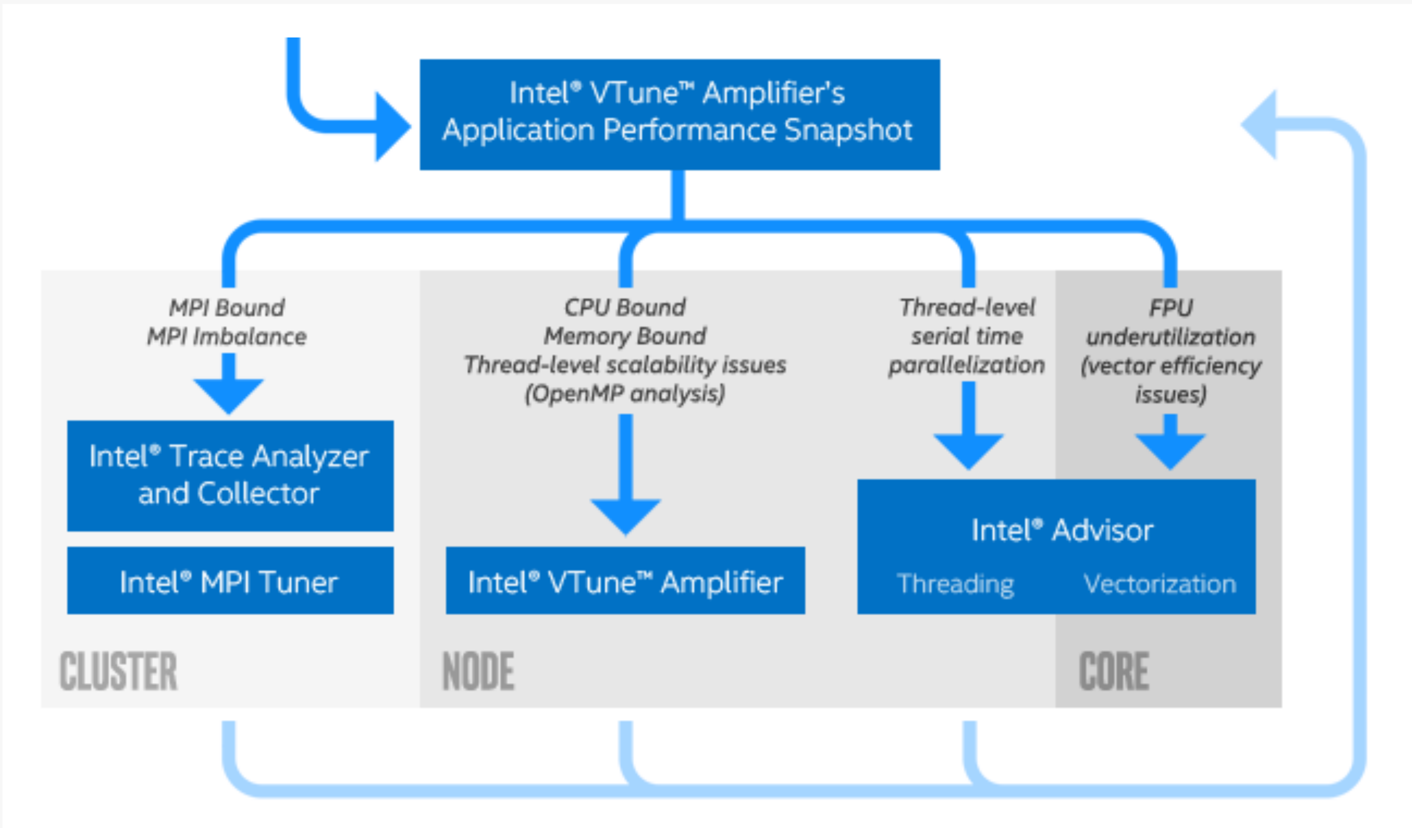
Intel Vtune

- Performance profiling tool to identify where in the code time is being spent in both serial and threaded applications.

- For threaded applications, it can also determine the amount of concurrency and identify bottlenecks created by synchronization primitive

- Different analysis groups
  - Hotspots (Advanced-hotspots is integrated here)
  - Memory consumption
  - Microarchitectural exploration
    - Hardware issues
    - Memory access analysis and high bandwidth issues

Intel Vtune



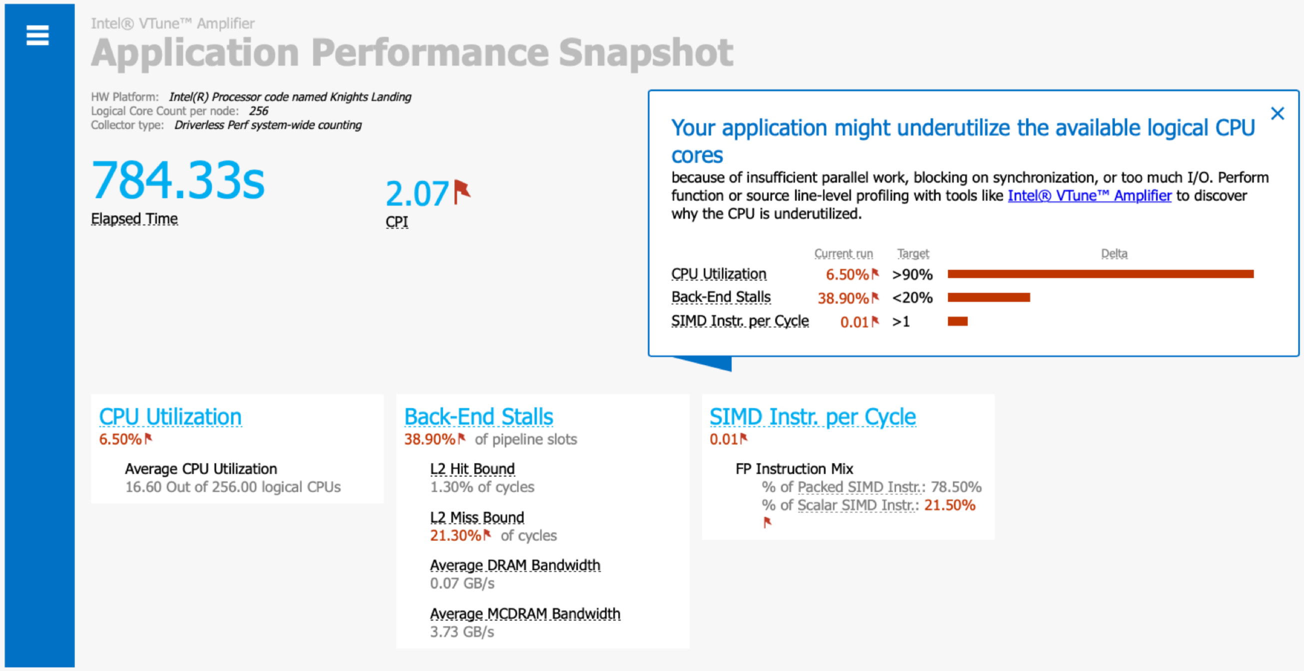
https://software.intel.com/sites/products/snapshots/application-snapshot/

Application Performance Snapshot (APS)

APS generates a high level performance snapshot of your application.

```
source /soft/compilers/intel-2019/vtune_amplifier_2019/apsvars.sh
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/soft/compilers/intel-2019/vtune_amplifier_2019/lib64
export PMI_NO_FORK=1

aps --result-dir=aps_results/ -- python /full/path/to/script.py
```

Application Performance Snapshot (APS)



Pros

- Very easy to use
- Tracks important hardware metrics:
  - Thread Load Balancing
  - Vectorization
  - CPU Usage

Cons

- Only high level information – but then again, that is the design of this tool.

Application Performance Snapshot (APS)

APS generates a highlevel performance snapshot of your application.

```
source /soft/compilers/intel-2019/vtune_amplifier_2019/apsvars.sh
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/soft/compilers/intel-2019/vtune_amplifier_2019/lib64
export PMI_NO_FORK=1

aps --result-dir=aps_results/ -- python /full/path/to/script.py
```

```
| Summary information
|--------------------------------------------------------------------
  HW Platform                : Intel(R) Processor code named Knights Landing
  Logical core count per node: 256
  Collector type             : Driverless Perf system-wide counting
  Used statistics            : aps_results
|
| Your application might underutilize the available logical CPU cores
| because of insufficient parallel work, blocking on synchronization, or too much I/O.
Perform function or source line-level profiling with tools like Intel(R) VTune(TM)
Amplifier to discover why the CPU is underutilized.
  CPU Utilization:                                     6.50%
| Your application might underutilize the available logical CPU cores because of
| insufficient parallel work, blocking on synchronization, or too much I/O.
| Perform function or source line-level profiling with tools like Intel(R)
```

Intel Vtune – Hotspots

Provides list of functions in an application ordered by the amount of time spent in each function.

```
source /soft/compilers/intel-2019/vtune_amplifier_2019/amplxe-vars.sh
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/soft/compilers/intel-2019/vtune_amplifier_2019/lib64
export PMI_NO_FORK=1

amplxe-cl -collect hotspots -finalization-mode=none -r vtune-result-dir_hotspots/ --
python /full/path/to/script.py
```

Pros

- **Can track activity from python code**
- Quickly identify heavy functions

Cons

- Will **not** run with more than a few threads, making it impossible to profile the “real” application.

Intel Vtune – Hotspots

`sampling-mode=sw` - User-Mode Sampling (default) used for profiling:

- Targets running longer than a few seconds
- A single process or a process-tree
- Python and Intel runtimes

`sampling-mode=hw` - (Advanced hotspots) Hardware Event-Based Sampling used for profiling:

- Targets running less than a few seconds
- All processes on a system, including the kernel

Intel Vtune – Advanced Hotspots

Advanced Hotspots analysis

- Detailed report of how effective the computation is on CPUs
- uses the OS kernel support or VTune Amplifier kernel driver
- extends the hotspots analysis by collecting call stacks, context switch and statistical call count data and analyzing the CPI (Cycles Per Instruction) metric.
- By default, this analysis uses higher frequency sampling at lower overhead compared to the Basic Hotspots analysis.

```
amplxe-cl -collect hotspots -knob sampling-mode=hw -finalization-mode=none -r vtune-
result-dir_advancedhotspots/ -- python /full/path/to/script.py
```

Intel Vtune – Advanced Hotspots

Advanced Hotspots analysis

- Detailed report of how effective the computation is on CPUs
- uses the OS kernel support or VTune Amplifier kernel driver
- extends the hotspots analysis by collecting call stacks, context switch and statistical call count data and analyzing the CPI (Cycles Per Instruction) metric.
- By default, this analysis uses higher frequency sampling at lower overhead compared to the Basic Hotspots analysis.

```
amplxe-cl -collect hotspots -knob sampling-mode=hw -finalization-mode=none -r vtune-result-dir_advancedhotspots/ -- python /full/path/to/script.py
```

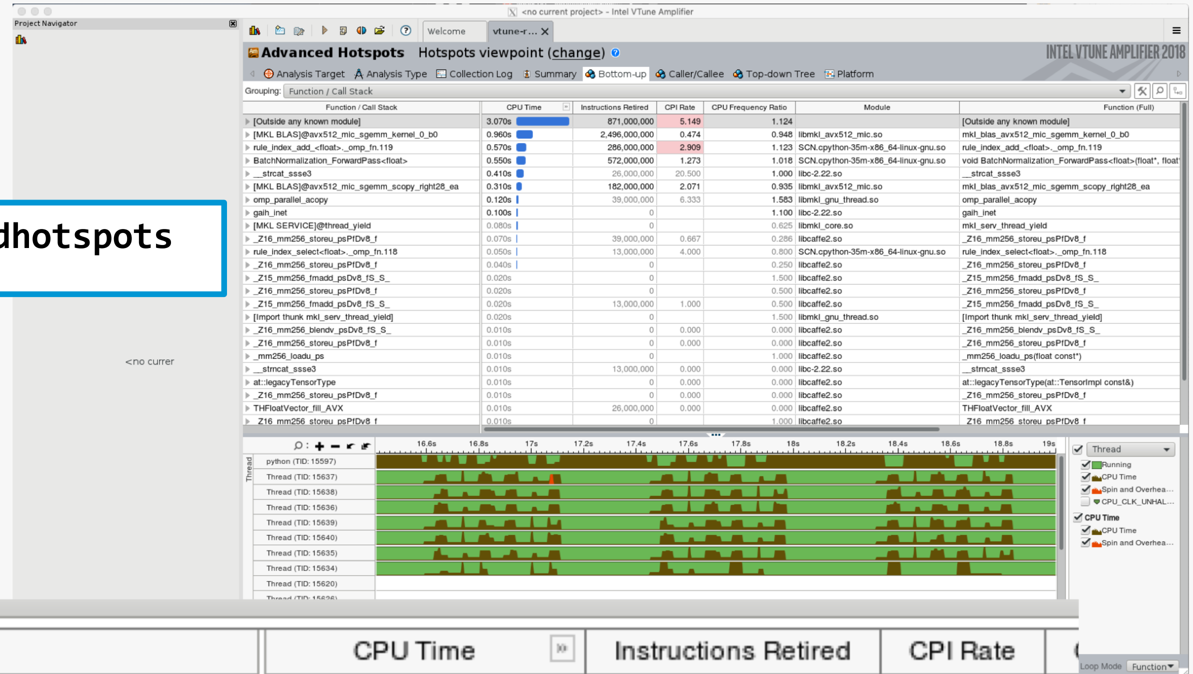
Run the finalization step after the run completes from the login nodes

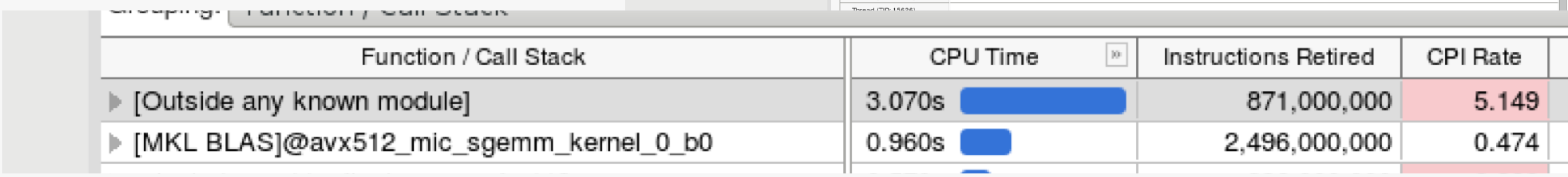
```
amplxe-cl -finalize -search-dir / -r vtune-result-dir_advancedhotspots
```

Intel Vtune – Advanced Hotspots

Run the GUI to view your results:

```
amplxe-gui  vtune-result-dir_advancedhotspots
```





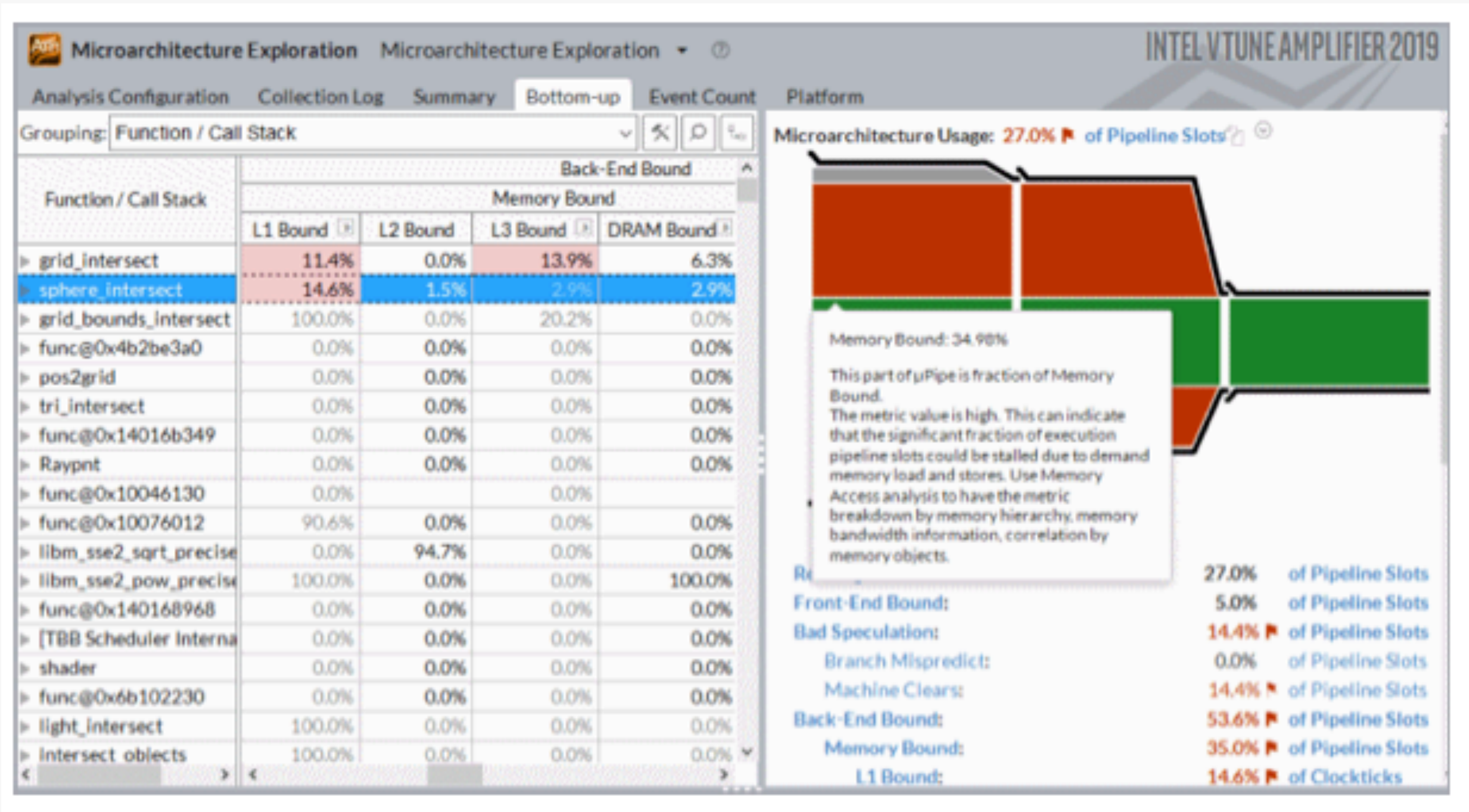
Pros

- Visualize each thread activity and the functions that cause it.
- Give a bottom up and top down view, very useful for seeing which functions are hotspots

Cons

- **Doesn't keep information at python level.**
- If your workflow uses JIT, you can lose almost all useful information.
- Understanding the information present takes some practice.

Intel Vtune – Microarchitectural Exploration



https://software.intel.com/en-us/vtune-amplifier-help-microarchitecture-exploration-analysis

Intel Vtune – Microarchitectural Exploration

```
amplxe-cl -collect uarch-exploration –r vtune-uarch -- python /full/path/to/script.py
```

Intel Vtune – Microarchitectural Exploration

```
amplxe-cl -collect uarch-exploration –r vtune-uarch -- python /full/path/to/script.py
```

knobs

collect-memory-bandwidth, pmu-collection-mode, dram-bandwidth-limits,sampling-interval, collect-frontend-bound, collect-bad-speculation, collect-memory-bound, collect-core-bound, collect-retiring.

```
$ amplxe-cl -collect uarch-exploration -knob collect-memory-bandwidth=true -–r vtune-uarch-mem -- python /full/path/to/script.py
```

Architecture-specific Tuning Guides, visit https://software.intel.com/en-us/articles/processor-specific-performance-analysis-papers.

Intel Vtune – Memory Access

Grouping: Bandwidth Domain / Bandwidth Utilization Type / Function / Call Stack

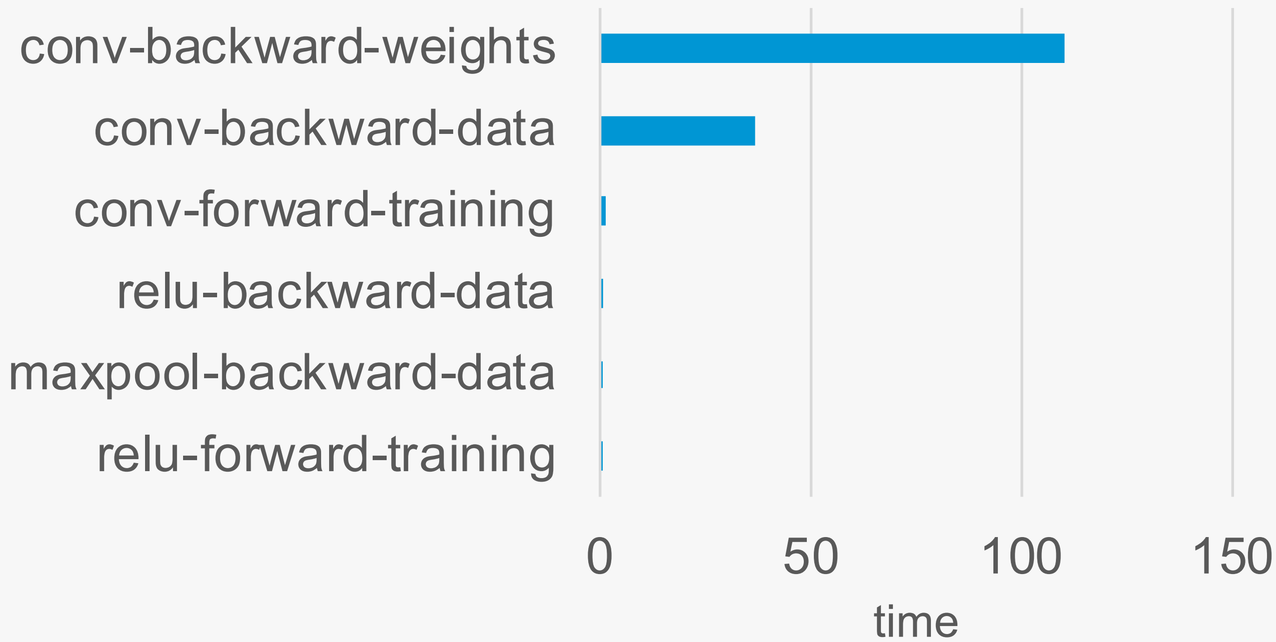
Bandwidth Domain / Bandwidth Utilization Type / Function / Call Stack	CPU Time	Memory Bound	Loads	Stores	LLC Miss Count	Average Latency (cycles)
DRAM, GB/sec	9.703s	64.3%	6,517,0...	4,141,26...	191,811,508	92
High	4.253s	56.8%	2,345,0...	2,111,23...	119,007,140	115
main	4.059s	54.6%	2,170,0...	2,046,83...	119,007,140	108
__intel_ssse3_rep_memcpy	0.177s	100.0%	175,000...	63,000,945	0	223
__do_softirq	0.012s	0.0%	0	0	0	0
run_timer_softirq	0.002s		0	0	0	0
__do_page_fault	0.001s	0.0%	0	0	0	0
numa_migrate_prep	0.001s	0.0%	0	0	0	0
task_cputime	0s	0.0%	0	1,400,021	0	0
Medium	2.880s	70.3%	2,765,0...	981,414,...	52,853,171	83

Example

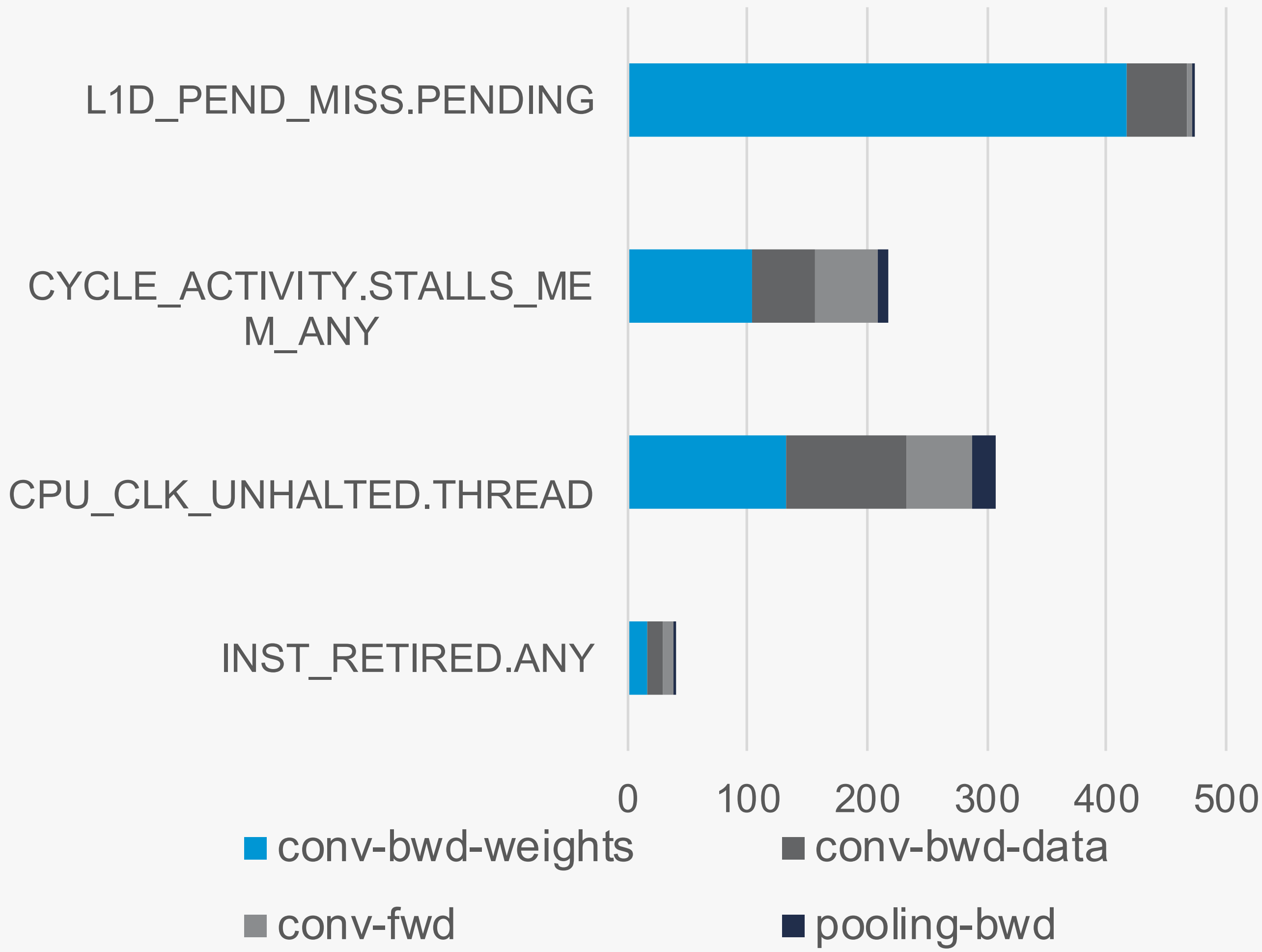
Simple CNN in Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu’))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax’))

model.compile(…....)

model.fit(.....)

model.evaluate(.....)
```

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py



primitive	propagation-time	auxilliary info	time
convolution	backward_weights	alg:convolution_direct	110.393
convolution	backward_data	alg:convolution_direct	36.48
convolution	forward_training	alg:convolution_direct	1.41211
eltwise	backward_data	alg:eltwise_relu	0.726074
pooling	backward_data	alg:pooling_max	0.51001
eltwise	forward_training	alg:eltwise_relu	0.0969238


L1D_PEND_MISS.PENDING
CYCLE_ACTIVITY.STALLS_MEM_ANY
CPU_CLK_UNHALTED.THREAD
INST_RETIRED.ANY
0
100
200
300
400
500
conv-bwd-weights
conv-bwd-data
conv-fwd
pooling-bwd

Operations on backward weights, data have stalls → high memory requirements

– Convolution layer is sensitive to compute units, memory and cachelines
– Dense layer is sensitive to communication -> bandwidth

Profiling Example – Tensorflow FFTs

An application that had very slow performance with Tensorflow on Theta, though with all optimized settings. Using vtune hotspots and advanced hotspots, it is reported that

- 31% of the application time was spent doing FFTs with tensorflow
- 10% was spent creating tensorflow traces
- 8% was computing loss functions.
- 25% was spent creating and optimizing the tensorflow graph (measured for a short run, this is a smaller fraction for production runs)

Most important hotspot **(FFT) was underperforming on Theta by up to 50x compared with the optimized FFT in Numpy**.

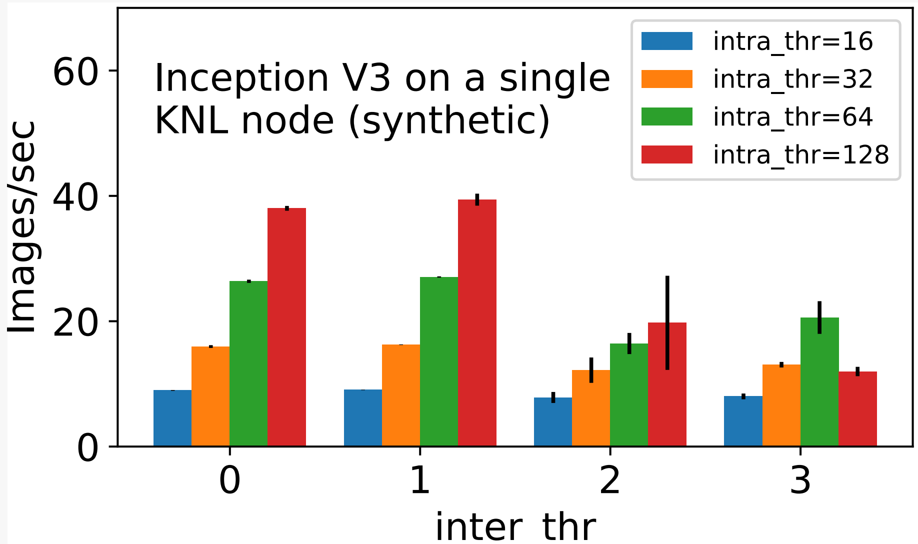
For this workflow, replacing tensorflow with numpy FFT + autograd for gradient calculations made a huge impact in their performance.

Optimization

Different configurations have different performance impact

intra_op_parallelism_threads: Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.

inter_op_parallelism_threads: All ready nodes are scheduled in this pool.




```
config = tf.ConfigProto()
config.intra_op_parallelism_threads = num_intra_threads
config.inter_op_parallelism_threads = num_inter_threads
tf.Session(config=config)
```

https://www.tensorflow.org/guide/performance/overview

Performance Setting Guidelines

Performance with Tensorflow on KNLs requires management of many parameters at both build and run time.

Intel Performance Guidelines: https://software.intel.com/en-us/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference

ALCF Performance Guidelines: https://www.alcf.anl.gov/user-guides/machine-learning-tools

Key Takeaways:

- Set **OMP_NUM_THREADS**=[number of physical cores = 64 on Theta]
- Set **KMP_BLOCKTIME**=0 (sometimes =1 can be better for non-CNN)
- (tensorflow only) Set intra_op_parallelism_threads == OMP_NUM_THREADS == number of physical cores == 64
- (tensorflow only) Set inter_op_parallelism_threads for your application. 0 will default to the number of cores, the optimal value can be different for different applications.

Useful Commands

```
amplxe-cl -c hotspots -- python3 myapp.py
amplxe-cl -R hotspots -report-output report-hotspots.csv -format csv

amplxe-cl -c uarch-exploration -k sampling-interval=100 -- python3 myapp.py
amplxe-cl -R uarch-exploration -report-output report-uarch-exploration.csv -format csv

amplxe-cl -c memory-access -k sampling-interval=100 -- python3 myapp.py
amplxe-cl -R memory-access -report-output report-memory-access.csv -format csv

amplxe-cl -c memory-consumption -k sampling-interval=100 -- python3 myapp.py
amplxe-cl -R memory-consumption -report-output report-memory-consumption.csv -format csv

change sampling interval
-k sampling-interval=<number>
```

Useful Commands

```
amplxe-cl -report hw-events/summary -r r000ue/ -report-output ./report-uarch.csv -format
csv

amplxe-cl -collect hotspots -strategy ldconfig:notrace:notrace -- python myapp.py

## get MKL-DNN verbose
export MKLDNN_VERBOSE=2
amplxe-cl -collect hotspots -strategy ldconfig:notrace:notrace -- python myapp.py
```

Thank you!

GEMM – 2*m*n*k operations

m, k – hidden layer size

n = minibatch size

2 * 512 * 512 * 64 = 0.03 GFLOP

Peak upper limit = 6000 GFLOP/s

Runtime ~ 5.6 usec

Time(%)	Time	Calls	Avg	Min	Max	Name
93.93%	575.72us	8	**71.964us**	70.241us	78.945us	maxwell_sgemm_128x64_tn