Profiling and Understanding DL Workloads on Supercomputing Systems

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

Argonne Leadership Computing Facility



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/cuda9.1.85/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



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 of
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

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

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 of this path

9.shared_load_throughput, shared_load_transactions 10.shared_store_throughput, shared_store_transactions 11.l1_cache_local_hit_rate

12.11 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



Metrics and Events

Metrics relevant to identify compute, memory, IO characteristics

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



Metrics and Events

Metrics relevant to identify compute, memory, IO characteristics

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

1 of 32 threads = 3%

32 of 32 threads = 100%

Warps efficiency/active cycles

19 Argonne Leadership Computing Facility



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

	; 🛛 🔪 🗔 Details 📮 Console 🗔 Settings				
	🛺 Export PDF Report				
1. CUDA A	pplication Analysis				
The guided stages to h application you can exp When optic compute an should loo performance	analysis system walks you through the various analysis elp you understand the optimization opportunities in you . Once you become familiar with the optimization process olore the individual analysis stages in an unguided mode mizing your application it is important to fully utilize the nd data movement capabilities of the GPU. To do this you k at your application's overall GPU usage as well as the ce of individual kernels.				
Examine GPU Usage					
Determine your application's overall GPU usage. This analysis requires an application timelin so your application will be run once to collect it if it is not already available.					
	🛺 Examine Individual Kernels				
Determine whi opportunity for application wil	Examine Individual Kernels ich kernels are the most performance critical and that have the most improvement. This analysis requires utilization data from every kernel, so y be run once to collect that data if it is not already available.				
Determine wh opportunity for application wil	Examine Individual Kernels ich kernels are the most performance critical and that have the most improvement. This analysis requires utilization data from every kernel, so y be run once to collect that data if it is not already available. Delete Existing Analysis Information				
Determine wh opportunity for application will If the application may be stale a	Examine Individual Kernels Examine Individual Kernels ch kernels are the most performance critical and that have the most improvement. This analysis requires utilization data from every kernel, so y be run once to collect that data if it is not already available. Delete Existing Analysis Information on has changed since the last analysis then the existing analysis information defended before continuing.				



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()
```



29 - Augorino Loudororing Comparing Laonity

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











achieved_occupancy



Argonne

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





Impact of batch size



IPC



dram utilization



achieved_occupancy





global memory efficiency



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

 $0.00\% \ 20.00\% \ 40.00\% \ 60.00\% \ 80.00\% 100.00\% \ 20.00\%$

■256 ■128 ■64



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	<pre>vecAdd(float*, float*, float*)</pre>		
1.04%	6.3680us	1	6.3680us	6.3680us	6.3680us	<pre>nonLin(float*, float*, float*, float*)</pre>		

==28493== API calls:

Time(응)	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



Combine many small data transfers to few large data transfers



Optimization





2.5x performance gain

42 Argonne Leadership Computing Facility



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
- 47 Argonne Leadership Computing Facility

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



• 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 vtuneresult-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 vtuneresult-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
		-

	Grouping: Function / Call Stack						- × ×
	Function / Call Stack	CPU Time 🖻 In	structions Retired	CPI Rate	CPU Frequency Ratio	Module	Function (Full)
	[Outside any known module]	3.070s	871,000,000	5.149	1.124		[Outside any known module]
	[MKL BLAS]@avx512_mic_sgemm_kernel_0_b0	0.960s	2,496,000,000	0.474	0.948	libmki_avx512_mic.so	mkl_blas_avx512_mic_sgemm_kernel_0_b0
	Fule_index_abd_ <iioalsomp_in.119< p=""></iioalsomp_in.119<>	0.5705	206,000,000	2.909	1.123	SCN.cpython-35m-x86_64-linux-gnu.so	rule_index_add_ <lioatsomp_in.119< td=""></lioatsomp_in.119<>
	Batchivormalization_ForwardPass <noat></noat>	0.5505	572,000,000	1.2/3	1.016	SCN.cpython-35m-x86_64-linux-gnu.so	Void Batcrinormalization_ForwardPass <noat>(noat*, no</noat>
	streat_ssses	0.4105	26,000,000	20.500	1.000	IDC-2.22.50	streat_ssses
	[MKL BLAS]@avx512_mic_sgemm_scopy_ngnt26_ea	0.3105	162,000,000	2.071	0.935	IDMKI_avx512_mic.so	mki_bias_avx512_mic_sgemm_scopy_ngnt26_ea
	▶ omp_paraliel_acopy	0.1205	39,000,000	6.333	1.563	libmki_gnu_thread.so	omp_parallel_acopy
	gain_inet	0.0005	0		1.100	IIDC-2.22.50	gan_inet
tspots	 Z16 mm056 sharey soD(D)8 4 	0.0005	20,000,000	0.007	0.025	Ibraffe2 as	716 mm0E6 starsu seDiDu8 f
	2.16_mm256_stored_psPiDV6_1	0.0705	13,000,000	4.000	0.200	CON another 25m u86 64 linux mu an	_216_mm256_stored_ps=10v6_1
	Fore_index_select <noatsomp_in.res< p=""></noatsomp_in.res<>	0.0505	13,000,000	4.000	0.000	SCN.cpython-35m-x86_64-imux-gnu.so	716 mm0E6 sterey seD(Dy8 f
	216_mm256_stored_psPiDv8_1	0.0405	0		0.250	libcalle2.so	_216_mm256_stored_psP1DV8_1
	J_215_mm256_madu_pstv8_15_5_	0.0205	0		1.500	libcalle2.so	_215_mm256_madu_pstv6_15_5_
	Z16_mm256_stored_psPiDv8_1	0.0205	13,000,000	1.000	0.500	libcalle2.so	_216_mm256_stored_psP1DV6_1
	ZIS_mm256_imadd_pstw6_iS_S_	0.0205	13,000,000	1.000	0.500	Ibcalle2.so	_z15_mm256_madd_pstvv6_15_5_
	Fimport trunk mkt_serv_triead_yteroj	0.0205	0	0.000	1.500	Ibmki_gru_trread.so	[Import Indrik miki_serv_Inread_yield]
	Z16_mm256_btenuv_psDv6_15_5_	0.0105	0	0.000	0.000	libcalle2.so	_216_mm256_blendv_psDv6_15_5_
	Z16_mm256_storeu_psP1Dv8_1	0.0105	0	0.000	0.000	Ibcalle2.so	_216_mm256_stored_psP1DV6_1
<no currer<="" td=""><td>_mm256_loadu_ps</td><td>0.0105</td><td>17,000,000</td><td>0.000</td><td>1.000</td><td>libcalle2.so</td><td>_mm256_loadu_ps(loat constr)</td></no>	_mm256_loadu_ps	0.0105	17,000,000	0.000	1.000	libcalle2.so	_mm256_loadu_ps(loat constr)
	stmcat_ssses	0.0105	13,000,000	0.000	0.000	IIDC-2.22.80	sincal_ssec
	at::iegacyTensorType	0.0105	0	0.000	0.000	Ibcane2.so	at::iegacyTensorType(at::TensorImpi consta)
	Z16_mm256_storeu_psP1DV8_1	0.0105	0	0.000	0.000	Ibcane2.so	_216_mm256_stored_psP1DV8_1
	THHOatvector_III_AVX	0.0105	26,000,000	0.000	0.000	libcane2.so	THPIOatVector_III_AVX
	Z16 mm256 storeu psPtDv8 1	0.0105	0		1.000	IIDcarrez.so	216 mm256 storeu psPtDvd t
	0: + 16.6a	16.8s 17s 17.	2a 17.4a	17.6s	17.8s 18	a 18.2a 18.4a 18.	6a 18.8a 19a
		terl entered terrestered					Thread •
	python (TID: 15597)		-		· • •		Running
	E Thread (TID: 15637)						CPU Time
	Thread (TID: 15638)		_				Spin and Overhea
	Thread (TID: 15636)						□ ♥ CPU_CLK_UNHAL
	Thread (TID: 15639)		_	4		- 1 -	CPU Time
	Thread (TD: 15940)						CPU Time
	Inread (IID: 15640)						Spin and Overhea
	Thread (TID: 15635)		_				
	Thread (TID: 15634)						

sinceping. (Failed off) can ocacit				
Function / Call Stack	CPU Time 🛛	Instructions Retired	CPI Rate	Loop Mode Function*
[Outside any known module]	3.070s	871,000,000	5.149	
[MKL BLAS]@avx512_mic_sgemm_kernel_0_b0	0.960s 🛑	2,496,000,000	0.474	
	_			

🛎 🏡 🖆 🎲 🕨 🕱 🖚 😂 🕐 Welcome 🛛 vtune-r..

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



- 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

Microarchitecture	Exploration	Microarch	itecture Explo	oration 👻 🔿		INTEL V TUNE AMPLIFIER 20
Analysis Configuration	Collection L	og Summa	ry Bottom-	up Event Count	Platform	
Grouping: Function / Call	Stack			~ × 0 to	Microarchitecture Usage: 27.0% 🍋	of Pipeline Slots 👌 Θ
SSSCOULSSSSSS			Back	-End Bound ^		
Function / Call Stack			Memory Bou	nd		
	L1 Bound 📧	L2 Bound	L3 Bound D	DRAM Bound		
grid_intersect	11.4%	0.0%	13.9%	6.3%		
sphere_intersect	14.6%	1.5%	2.9%	2.9%		<u> </u>
grid_bounds_intersect	100.0%	0.0%	20.2%	0.0%		
func@0x4b2be3a0	0.0%	0.0%	0.0%	0.0%	Memory Bound: 34,98%	
pos2grid	0.0%	0.0%	0.0%	0.0%	This part of µPipe is fraction of Mem	ory
tri_intersect	0.0%	0.0%	0.0%	0.0%	Bound. The metric value is high. This can ind	icate
func@0x14016b349	0.0%	0.0%	0.0%	0.0%	that the significant fraction of execut	tion
Raypot	0.0%	0.0%	0.0%	0.0%	pipeline slots could be stalled due to memory load and stores. Use Memory	demand
func@0x10046130	0.0%		0.0%		Access analysis to have the metric	.,
func@0x10076012	90.6%	0.0%	0.0%	0.0%	breakdown by memory hierarchy, m	emory .
libm_sse2_sqrt_precise	0.0%	94.7%	0.0%	0.0%	memory objects.	
libm_sse2_pow_precise	100.0%	0.0%	0.0%	100.0%	Rí	27.0% of Pipeline Sto
func@0x140168968	0.0%	0.0%	0.0%	0.0%	Front-End Bound:	5.0% of Pipeline Sto
TBB Scheduler Interna	0.0%	0.0%	0.0%	0.0%	Bad Speculation:	14.4% 🏲 of Pipeline Sto
shader	0.0%	0.0%	0.0%	0.0%	Branch Mispredict:	0.0% of Pipeline Sto
func@0x6b102230	0.0%	0.0%	0.0%	0.0%	Machine Clears:	14.4% Nof Pipeline Slo
light_intersect	100.0%	0.0%	0.0%	0.0%	Back-End Bound:	53.6% P of Pipeline Sto
intersect objects	100.0% <	0.0%	0.0%	0.0% *	Memory Bound: L1 Bound:	35.0% P of Pipeline Slo 14.6% P of Clockticks

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

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

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

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



Intel Vtune – Memory Access

Grouping: Bandwidth Domain / Bandwidth Utilization Type / Function / Call Stack						¢ t.
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	Os	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





conv-backward-weights conv-backward-data conv-forward-training relu-backward-data maxpool-backward-data relu-forward-training

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







Operations on backward weights, data have stalls \rightarrow 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: <u>https://software.intel.com/en-us/articles/maximize-</u> tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference

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

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

