The SOS 24 Virtual Conference

24 March 2021

Recent Advances in Using mOS for ML Workloads

Rolf Riesen, Jai Dayal, John Attinella

Introduction

The multi-operating system for HPC (mOS) is an active, open-source project at Intel Corp. and will be available on the Aurora exascale system at Argonne.

Originally designed for traditional supercomputing workloads, we have started running Machine Learning (ML) workloads on mOS. In this talk we

1. show how mOS can be used for application performance tuning,
2. learn how ML applications and system software interact, and
3. look at a couple more ML applications and how they perform on mOS.

Introduction to mOS

mOS is designed for exascale and beyond.

- mOS is targeting Aurora
  - Alternate OS to HPE's* CLE/CNL production system
- Team of eight people working on mOS and Aurora bring up
  - Well connected to other teams at Intel®:
  - MPICH, Intel® MPI, PVC driver, Intel® oneAPI, Aurora design & bring-up,
  - Runtime systems, Linux® kernel, application performance, etc.
- Beginning to look for other customers
- Open source: `https://github.com/intel/mOS/wiki`

*Other names and brands may be claimed as the property of others.

mOS Features

The mOS lightweight kernel (LWK) is much more than a non-preemptive scheduler and physically contiguous pages.

- About 20,000 lines of code embedded in the Linux® kernel
- Five functional areas: (details in the Appendix)
  - Hard resource partitioning (Slide 37)
  - Memory management (Slide 38)
  - CPU management (Slide 39)
  - Process scheduling (Slide 40)
  - Thread/task management (Slide 41)

Performance Tuning

CANDLE Pilot 3 Example

CANDLE Pilot 3

Exascale Computing Project (ECP) Cancer Distributed Learning Environment (CANDLE)[1].

- These experiments were done in 2019
- Uses OpenMP, TensorFlow* 1.0, and Intel® MKL-DNN
- Oversubscribes cores (something LWK don't like)
- OpenMP and TensorFlow* threads compete for CPUs

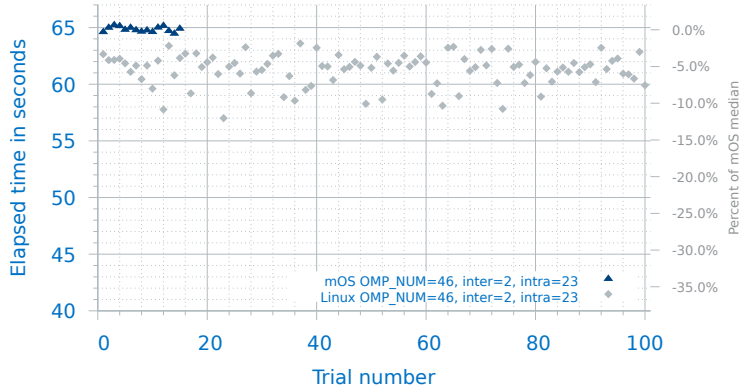
System and software configuration in Appendix, Slide 53*ff*

[1] Justin M. Wozniak et al. ' ' CANDLE/Supervisor: a workflow framework for machine learning applied to cancer research". In: *BMC Bioinformatics* 19.18 (2018), p. 491. ISSN: 1471-2105. DOI: \url{https://doi.org/10.1186/s12859-018-2508-4}. URL: \url{https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2508-4}.

*TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Initial CANDLE Pilot 3 Runs

- Used recommended parameters for
  - `inter_op`, `intra_op`, and `OMP_NUM_THREADS`.



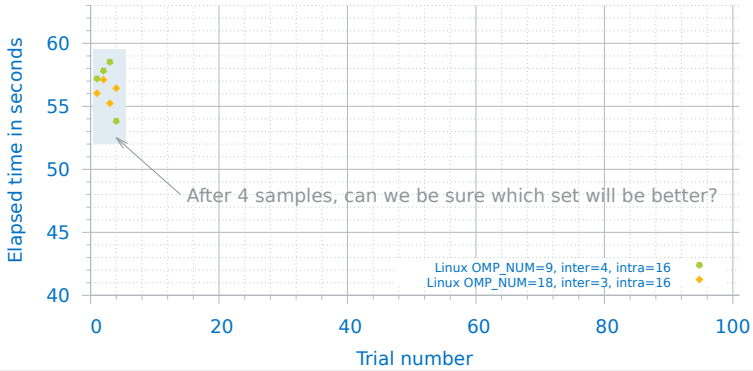
mOS

ML Workloads

Rolf Riesen

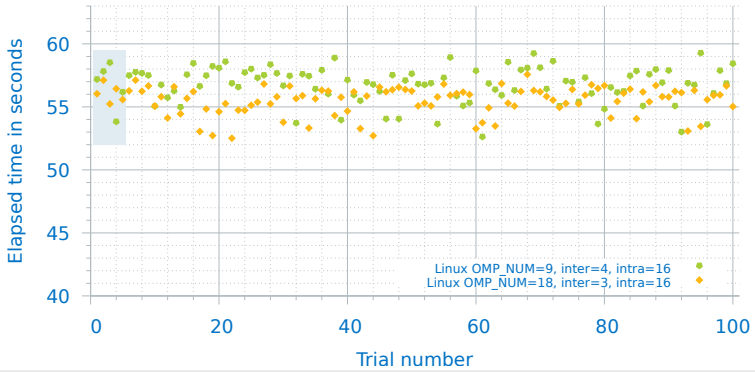
Parameter Sweep

- mOS does better with a single thread per logical CPU
- Did a parameter sweep to find best combination
  - Used single mOS runs; Linux® needs too many samples



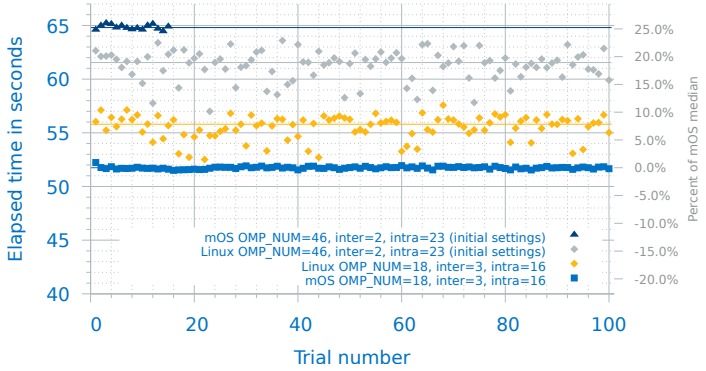
mOS: Fewer Samples Needed

- Need lots of samples to find average Linux® performance
- For the two sets below, at least 10 runs each were needed to decide which of the two parameter sets was better



Final Result

Using single runs on mOS, we found good parameters that improved Linux® ≈10% and mOS ≈25%



ML Workloads and System Software

BERT Introduction

Bidirectional Encoder Representations from Transformers (BERT) is a method of pre-training language representation[2].

Goals of these experiments:

- Configure the system and BERT to produce high performance result under Linux® and mOS
- Compare and study what improvements could be made to mOS
  - Goal is *not* to optimize BERT, rather find *enhancements for* mOS

[2] Jacob Devlin et al. ‘ ‘ BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Approach

- Parameter sweep to find good settings for intra-, and inter-operator pool sizes, and number of OpenMP threads
- Optimize using modern memory allocators: TC, JE, and TBBmalloc
- Analyze and compare best effort Linux® and mOS configurations
- Enhance mOS (work in progress)

Initial Settings

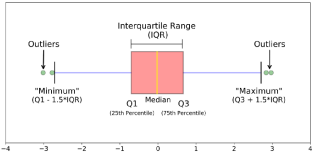
- To start with, Intel® recommends[3]:
  - Intra-op pool size = #cores
  - Inter-op pool size = 2
  - `OMP_NUM_THREADS` = #cores
  - 46 cores for skl-11 + 2 cores for OS
- Test system has $2 * 24 = 48$ cores
  - Reserve 1 core per socket for the OS
  - That leaves $2 * 23 = 46$ cores for compute
- BERT uses 18.5 GB of memory while running

[3] Nathan G. Greeneltch and Jing Xu. *Maximize TensorFlow Performance on CPU: Considerations and Recommendations for Inference Workloads*. https://software.intel.com/content/www/us/en/develop/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference.html. Jan. 2019.

Legend

- Tuples like this (OMP threads, Intra, Inter) mean:
  - Num OpenMP threads, TensorFlow* Intra and Inter thread pool sizes
  - Intra = 0 and Inter = 0 are default (i.e., let TensorFlow* decide)

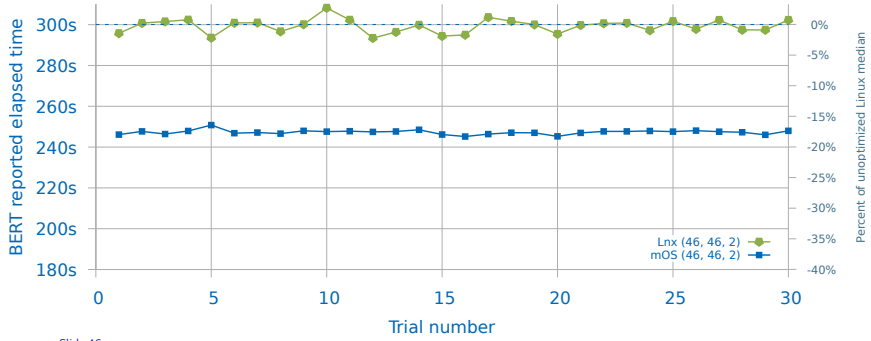
Box plots (five-number summary)



From: https://www.simplypsychology.org/boxplots.html and https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

Initial BERT Runs Plot

Last year's Summer intern helped us get started.



Data summary on Slide 46.

mOS

ML Workloads

Rolf Riesen

Notes

- Nothing optimized: Default recommended settings
  - `OMP_NUM_THREADS` = 46, Intra = 46, and Inter = 2
- mOS beats Linux® handily out of the box
  - BERT under mOS has >15% faster run time, than under Linux®
- Linux® uses *48* cores
- mOS uses *46* cores!

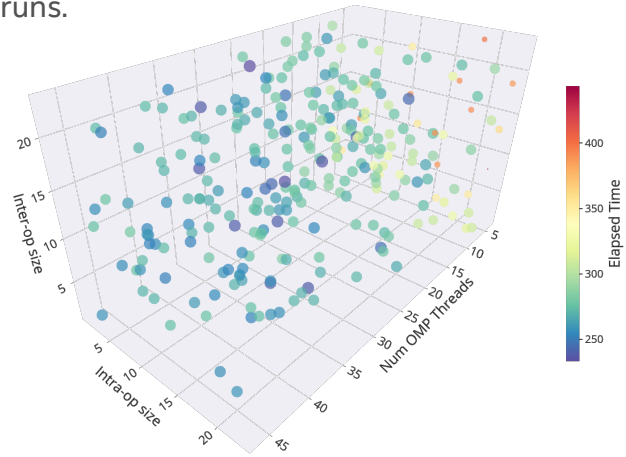
Parameter Search

Are other settings for `OMP_NUM_THREADS`, Intra, and Inter better?

- With > 6-minute run times, search space is too large for an exhaustive search
- Use random sampling
  - Num OMP threads = random[1…47]
  - Intra op size = random[1…24]
  - Inter op size = random[1…24]
  - Limit to 8 minutes per run
    - We already know best configuration leads to 6-minute runs or better

Random Parameter Sweep under Linux

278 valid runs.

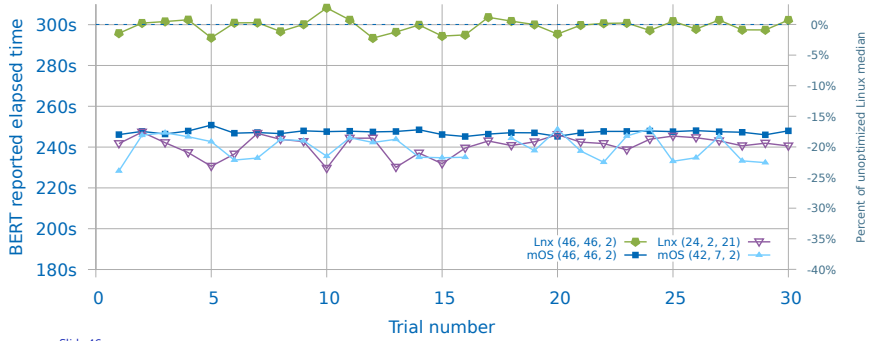


Findings

- More OpenMP threads is better for mOS
- A medium number of OpenMP threads is good for Linux®
- No strong contender for best Intra- and inter-op size
  - This is *different* from what we saw with CANDLE and TensorFlow* 1.0
  - Best triple for mOS: 42 OMP threads, Intra = 7, Inter = 2: (42, 7, 2)
  - Best triple for Linux®: 24 OMP threads, Intra = 2, Inter = 21: (24, 2, 21)

Improved TensorFlow*Settings

Average performance for both OSs improved, but run-to-run variability got worse.



Data summary on Slide 46.

mOS

ML Workloads

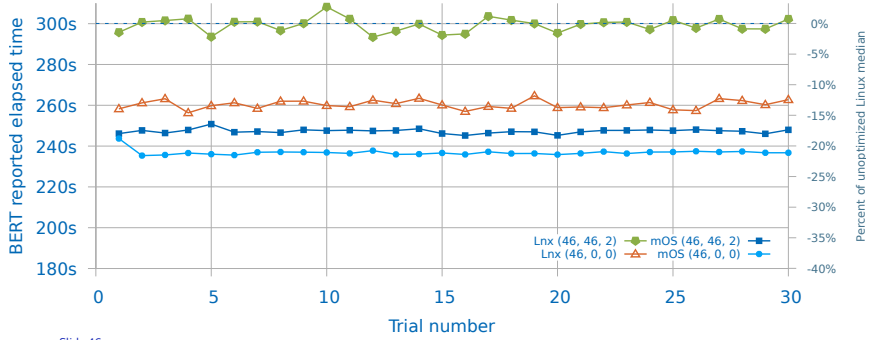
Rolf Riesen

Let TensorFlow* Determine Pool Sizes

mOS

Set Intra and Inter to 0 and let TensorFlow* 2.1 determine pool size.

- Linux® improves by ≈13%; mOS by ≈4%



Data summary on Slide 46.

Modern Malloc

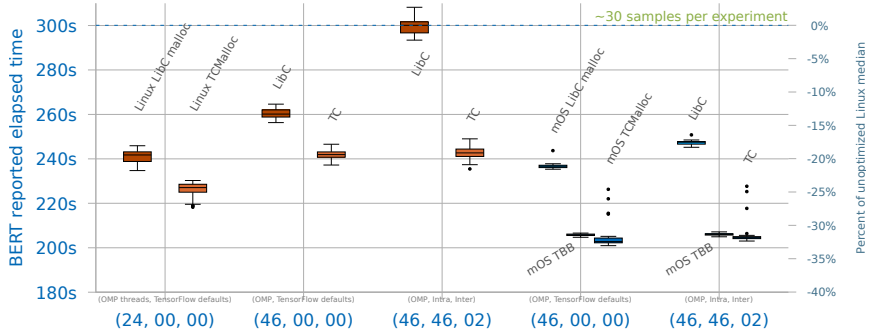
Multi-threaded applications in multiprocessor systems can benefit from modern memory allocators.

- E.g., JEmalloc, TBBmalloc, and TCMalloc
- They replace the standard lib C routines; e.g., `malloc()` and `free()`
- Link against the new libraries, or use `LD_PRELOAD`
- Reduces lock contention among threads, uses large pages, and allocates local caches

More detail about malloc is in the Appendix, Slide 10*ff*

Compare Improvements

Comparing TCMalloc improvements under Linux® and mOS.



Data summary on Slide 47.

More Experiments

We did a lot more experiments to learn about these new types of workloads.

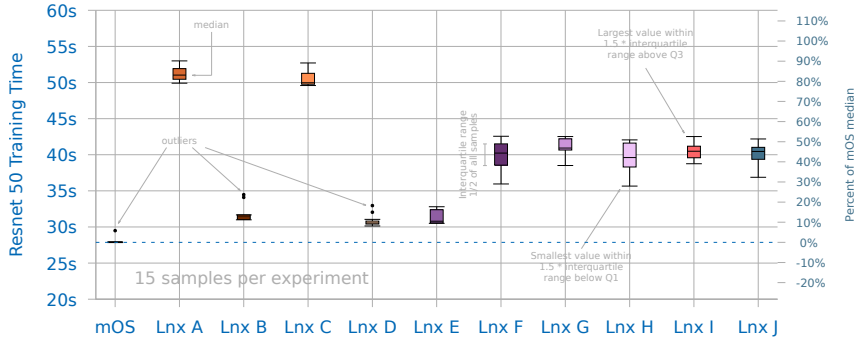
- Thread affinity *(100s of threads, low overhead scheduling is important)*
- Memory interleaving *(Needed)*
- SNC-4 mode *(helps Linux® 3% and mOS 8% on Ice Lake)*
- Page size impact *(TCMalloc likes 2 MiB and 1 GiB pages)*
- Tuning Linux® memory *(THP + TCMalloc is better than HugeTLBfs)*
- CPU sets, etc. for Linux® *(Fewer compute CPUs = lower performance)*

Invite me for a talk at your institution, if you want the gory details! ;-)

More ML Examples

ResNet-50 Experiment Set 1

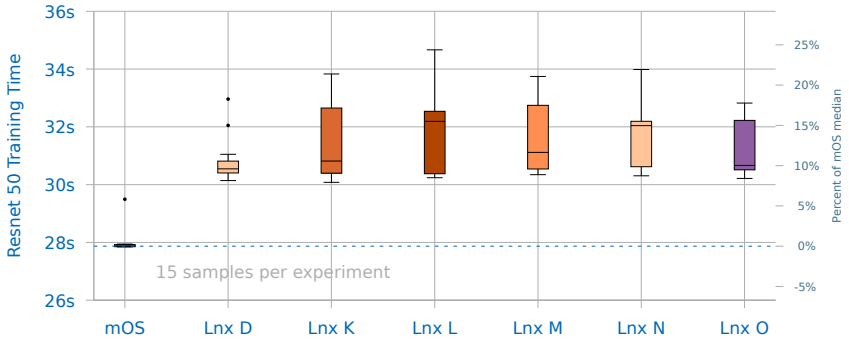
After parameter sweep, settling on (2, 40, 40) for mOS and (6, 32, 24) for Linux®. Now, find best possible Linux® configuration.



Data summary on Slide 48.

ResNet-50 Experiment Set 2

Try to improve upon Lnx D.



Data summary on Slide 48.

Configurations for Set 1

Setting	Lnx	A	B	C	D	E	F	G	H	I	J	mOS
`selinux=0`		●	●	●	●	●	●	●	●	●	●	●
`nmi_watchdog=0`		●	●	●	●	●	●	●	●	●	●	●
`nohz_full=1-23,25-47,49-71,73-95`		●	●				●	●	●	●	●	●
`kernelcore=16G`		●	●									●
`intel_pstate=disable`		●	●	●	●							●
`intel_pstate=active`							●			●	●	
`intel_pstate=per_cpu_perf_limits`						●		●	●			
`/sys/.../scaling_governor` = performance						●			●			
`/sys/.../scaling_governor` = powersave								●				
LibC malloc		●		●								
TCMalloc			●		●	●	●	●	●	●	●	●
`KMP_AFFINITY=none`					●	●	●	●	●	●	●	●
`granularity=fine,compact,1,0`		●	●	●								
`numactl -interleave=all`		●	●	●	●	●	●	●	●			
`-physcpubind=1-23,25-47,49-71,73-95`							●	●	●	●	●	
`-localalloc`											●	
Median training time (15 samples each)		51.0	31.4	50.0	30.6	30.8	40.2	40.9	39.6	40.5	40.5	27.9

Configurations for Set 2

Setting	Lnx	D	K[1]	L	M	N	O

`selinux=0`
`nmi_watchdog=0`
`nohz_full=1-23,25-47,49-71,73-95`
`kernelcore=16G`
		●
		●
			●
			●
				●
				●
					●
					●
						●
						●
							●
							●

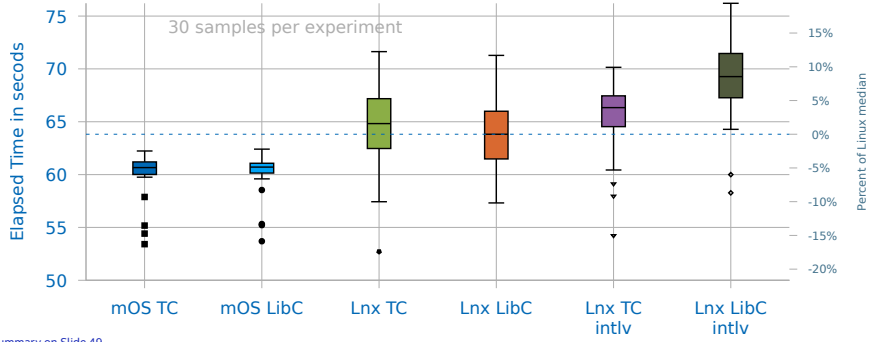
`intel_pstate=disable`
`intel_pstate=per_cpu_perf_limits`
`/sys/.../cpufreq/scaling_governor` = performance
`/sys/.../cpufreq/scaling_governor` = powersave
		●	●	
				●
				●
					●	●	●

TCMalloc		●	●	●	●	●	●
`KMP_AFFINITY=` `KMP_AFFINITY=granularity=fine,compact,1,0`		●	●	●	 ●	 ●	 ●
`numactl -interleave=all`		●	●	●	●	●	●
Linux® HPC tuning (see Table on Slide 68) `KMP_BLOCKTIME=0`						●	● ●
Median training time (15 samples each)		30.6	30.8	32.2	31.1	32.0	30.7

[1] Lnx K is to confirm that Lnx D is reproducible.

CANDLE Uno

- Thanks to Jai Dayal and Murali Krishna Emani for app and data
- Very preliminary data!
  - TCMalloc doesn't help and memory interleaving is no good!



Data summary on Slide 49.

Conclusion

- Using CANDLE Pilot 3, we saw how mOS can help with performance tuning
- Using BERT and malloc libraries we saw that system software plays an important role
- With ResNet-50 we realized that Linux® configuration for HPC performance is not easy
- Looking at CANDLE Uno, we learn that each ML application is unique

We learned that mOS is suitable to run ML applications and produces good results without much tuning. We will continue to study these applications and improve mOS.

Thanks for inviting me and listening to my presentation!

People involved with mOS

Architecture, design, implementation, and testing:

- John Attinella
- Sharath Bhat
- Jai Dayal
- David van Dresser
- Tom Musta
- Rolf Riesen
- Andrew Tauferner

Vision, management, and guidance:

- Michael Blocksome
- Leonardo Borges
- Todd Inglett
- Pardo Keppel
- Lance Shuler
- Philippe Thierry
- Robert W. Wisniewski

Collaboration with RIKEN, Japan

- Balazs Gerofi
- Yutaka Ishikawa
- Masamichi Takigi

https://github.com/intel/mOS/wiki

Appendix

mOS

About mOS

mOS is an open-source, Linux-based OS under development at Intel. The source code can be obtained from `https://github.com/intel/mOS/wiki`.
Note that some of the results presented here have been obtained using mOS versions that have not been pushed to the public repository yet. It is our intent to publish the latest version as soon as possible.

Hard Resource Partitioning

- Three-stage resource management
  - At boot time or between jobs, *designate* resources for LWK
  - At process launch, *reserve* a subset from the designated pool
  - Running process *allocates* resources from reserved set
- Processes remain within the logical CPUs reserved for them
  - Each process "owns" a set of CPUs (and memory)
  - This reduces cache and TLB flushes and the associated interrupts

While not suitable for all situation, strict partitioning reduces noise and increases performance and determinism.

Memory management

- Physical memory is mapped at time of `mmap()`, `brk()`, or process launch (bss, thread stack, heap)
  - LWKMEM v2 has some additional options; e.g., fault on first touch
- Per-process physical memory allocations
  - No lock contention among processes
- Physically contiguous whenever possible
- NUMA aware allocation
  - Very launch-time configurable (think numactl on steroids)
  - Automatic interleaving and page size selection: 4k, 2m, or 1g
- Policy and page size can be independently requested for data bss, heap, anonymous mmap, and thread stack.

CPU management

- Keep non-LWK tasks off logical CPUs managed by LWK
- Includes almost all kernel tasks
  - Exceptions are: `ksoftirqd`, `cpuhp`, threads in the Linux "stop" class
  - They don't run during the bulk of the application execution
- All interrupts that can be moved are moved away
- Disable polling for recoverable machine checks while job is active
  - Poll is done when job ends
  - Exploit hardware buffering of log events as much as possible
- LWK specific idle loop
  - Provides CSTATE behaviors more suited to HPC workloads
  - Flexible launch-time options to control idle for each process

Process scheduling

- No timer interrupts when a single thread is runnable
- Simple round-robin scheduler otherwise
  - Similar to Linux `SCHED_RR`; configurable per process at launch time
  - Priority system available
- Deterministic task placement
- Wake-up balancing to coexist with runtimes that explicitly place threads
- Utility thread API

Thread/task management

- Aims for one thread per logical CPU
- Compute threads never move unless triggered (by user or runtime)
- Can push utility threads into Linux partition
- Thread balancing (push and pull) capability

JEmalloc

Came into use as the FreeBSD libc allocator in 2005… Using multiple arenas for scalability in multi-processor and multi-threaded systems.

- Better with memory fragmentation than lib C
- `https://github.com/jemalloc/jemalloc`
- Use `LD_PRELOAD=..../libjemalloc.so.2` to activate

TCMalloc

TCMalloc is Google's customized implementation of C's `malloc()` and C++'s operator `new`...TCMalloc is designed to be more efficient at scale than other implementations...Performance scales with highly parallel applications.

- `https://github.com/google/tcmalloc`
- Requires `bazel`, `abseil`, and *Internet access(!)* just to build
- Does not build dynamic, shared libraries
- Installing `google-perftools4` is easiest way to get access
- Use `LD_PRELOAD=/usr/lib/libtcmalloc.so.4` to activate
- Using TCMalloc version 4.4.5 for the experiments in this report

TCMalloc, cont.

- Does per-CPU caching (No longer **T**hread **C**aching malloc)
  - Maintains memory caches local to individual logical cores
- TCMalloc requests memory from the OS in large chunks
  - typically 1 GiB regions (per documentation)
- TCMalloc assumes address space is reserved, but not backed by physical memory until it is used.

TBBMalloc

Threading Building Blocks (TBB) scalable memory allocator bundled with Intel® oneAPI software stack.

- Designed to support TBB, but can be used stand-alone
- Uses segregated (for similar-sized objects), thread-private heaps[4]
- Use `LD_PRELOAD=..../libtbbmalloc_proxy.so.2` to activate

[4] Alexey Kukanov and Michael J. Voss. ‘ ‘ The Foundations for Scalable Multi-core Software in Intel Threading Building Blocks". In: *Intel Technology Journal* 11 (4 Nov. 2007), pp. 309322. URL: https://www.intel.com/content/dam/www/public/us/en/documents/research/2007-vol11-iss-4-intel-technology-journal.pdf.

Data Summary for Initial BERT Runs

For plots see Slides 16, 21, and 22.

OS	Settings	n runs	min	$\overline{x}$	max	$\widetilde{x}$	s	$\sigma_{\overline{x}}$
Linux	(46, 46, 2)	30	293.4	299.4	308.2	300.1	3.38	0.63
mOS	(46, 46, 2)	30	245.2	247.3	250.8	247.5	1.06	0.20
Linux	(24, 2, 21)	30	229.6	240.7	247.4	242.0	4.86	0.90
mOS	(42, 7, 2)	28	228.3	239.7	248.9	240.3	5.94	1.14
Linux	(46, 0, 0)	30	256.3	260.3	264.6	260.2	2.17	0.39
mOS	(46, 0, 0)	30	235.4	236.8	243.7	236.7	1.43	0.27

Sample $\overline{x}$ mean, $\widetilde{x}$ median, s standard deviation, $\sigma_{\overline{x}}$ standard error of the mean

Data Summary for TCMalloc Experiments

For plot see Slide 24.

OS	Settings	malloc	n runs	min	$\overline{x}$	max	$\widetilde{x}$	s	$\sigma_{\overline{x}}$
Linux	(24, 0, 0)	LibC	30	234.7	241.2	245.9	241.8	2.99	0.56
Linux	(24, 0, 0)	TC	30	218.3	226.0	230.3	227.1	3.49	0.65
Linux	(46, 0, 0)	LibC	30	256.3	260.3	264.6	260.2	2.12	0.39
Linux	(46, 0, 0)	TC	30	237.2	242.0	246.6	241.9	2.00	0.37
Linux	(46, 46, 2)	LibC	30	293.4	299.4	308.2	300.1	3.38	0.63
Linux	(46, 46, 2)	TC	30	235.5	242.6	249.0	242.7	3.00	0.56
mOS	(46, 0, 0)	LibC	30	235.4	236.8	243.7	236.7	1.43	0.27
mOS	(46, 0, 0)	TBB	30	204.7	205.8	206.6	205.8	0.47	0.09
mOS	(46, 0, 0)	TC	30	201.0	205.1	226.3	202.8	6.18	1.15
mOS	(46, 46, 2)	LibC	30	245.2	247.3	250.8	247.5	1.06	0.20
mOS	(46, 46, 2)	TBB	30	204.9	206.0	207.2	206.1	0.47	0.09
mOS	(46, 46, 2)	TC	28	203.0	206.5	227.7	204.7	6.21	1.20

Sample $\overline{x}$ mean, $\widetilde{x}$ median, s standard deviation, $\sigma_{\overline{x}}$ standard error of the mean

Data Summary for ResNet-50

For plots see Slides 27 and 28.

OS	Trials (n)	min	$\overline{x}$	max	$\widetilde{x}$	s	$\sigma_{\overline{x}}$
mOS	15	27.845	27.989	29.494	27.870	0.418	0.112
Lnx A	15	49.907	51.123	52.994	51.027	0.844	0.226
B	15	30.994	31.694	34.473	31.393	1.080	0.289
C	15	49.581	50.383	52.705	49.965	0.909	0.243
D	15	30.142	30.824	32.961	30.548	0.740	0.198
E	15	30.505	31.375	32.801	30.803	0.954	0.255
F	15	35.960	39.925	42.558	40.232	1.903	0.509
G	15	38.513	41.057	42.521	40.927	1.221	0.326
H	15	35.665	39.610	42.063	39.611	2.015	0.539
I	15	38.753	40.492	42.522	40.489	0.955	0.255
J	6	36.876	40.063	42.187	40.462	1.812	0.810
K	15	30.080	31.428	33.831	30.817	1.212	0.324
L	15	30.240	31.666	34.667	32.190	1.314	0.351
M	15	30.346	31.611	33.745	31.116	1.171	0.313
N	15	30.306	31.640	33.987	32.043	0.987	0.264
O	15	30.216	31.181	32.823	30.666	0.967	0.258

Sample $\overline{x}$ mean, $\widetilde{x}$ median, s standard deviation, $\sigma_{\overline{x}}$ standard error of the mean

Data Summary for CANDLE Uno

For plot see Slide 31.

OS	Malloc	CPUs	Trials (n)	min	$\overline{x}$	max	$\widetilde{x}$	s	$\sigma_{\overline{x}}$
Linux	LibC	96	30	57.318	63.920	71.278	63.819	3.333	0.6189
Linux	TC	96	30	52.707	64.586	71.637	64.823	4.277	0.7942
Linux intlv	LibC	96	30	58.270	69.233	76.207	69.269	4.129	0.7668
Linux intlv	TC	96	30	54.244	65.392	70.150	66.338	3.635	0.6751
mOS	LibC	92	30	53.696	60.126	62.410	60.707	1.996	0.3707
mOS	TC	92	30	53.416	60.078	62.236	60.661	2.125	0.3946

Sample $\overline{x}$ mean, $\widetilde{x}$ median, s standard deviation, $\sigma_{\overline{x}}$ standard error of the mean

Test System

Test System Used for Experiments

mOS

Single node CPU results only.

- CPU type
  - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
  - Model 85, Stepping 4
  - Cores per socket 24 (two sockets)
- Microcode patch level
  - 0x200004d
- Memory size/speed
  - 192 GB, 2666 MT/s
  - 2 memory controllers per CPU; 3 memory channels each
- Caches
  - L1d: 32K, L1i: 32K, L2: 1024K, L3: 33792K
- Network
  - Intel Corporation Omni-Path HFI Silicon 100 Series [discrete] (rev 11)

CANDLE Pilot 3 Setup and Software Used

Software

- CANDLE using Pilot 3, P3B1
- `p3b1_baseline_keras2.py`
- From the CORAL2 deep learning suite v6
  - `https://asc.llnl.gov/coral-2-benchmarks/`
- Intel® Parallel Studio XE 2018.3.051 with MKL-DNN
- With Intel® Python 3.6.8 and Optimizations for TensorFlow*
- CentOS Linux® 7 (Core)

System config on Slide 51

OS Kernel

Operating system: CentOS 7 using a 4.14.134 kernel and mOS v0.7 using the same kernel version.

```
CONFIG_MOS_FOR_HPC=y
CONFIG_MOS_MOVE_SYSCALLS=y
CONFIG_MOS_SCHEDULER=y
CONFIG_MOS_LWKMEM=y
CONFIG_NO_HZ_FULL=y
CONFIG_NO_HZ_FULL_ALL=y
CONFIG_RCU_NOCB_CPU=y
CONFIG_RCU_NOCB_CPU_ALL=y
```

Boot command line arguments:

```
kernelcore=16G intel_pstate=disable nmi_watchdog=0
```

Linux Setup

Command line:

```
python ./p3b1_baseline_keras2.py --inter-op-threads 2 --intra-op-
    threads 23
```

Environment:

```
OMP_NUM_THREADS=46
KMP_AFFINITY=granularity=fine,compact,1,0
KMP_BLOCKTIME=0
I_MPI_PIN_DOMAIN=96
KMP_HW_SUBSET=1t
```

mOS Setup

Command line:

```
yod python ./p3b1_baseline_keras2.py --inter-op-threads 2 --intra-op-
    threads 23
```

Environment:

```
KMP_AFFINITY=granularity=fine,compact,1,0
I_MPI_PIN=off
OMP_NUM_THREADS=46
KMP_BLOCKTIME=0
```

BERT Setup and Software Used

Operating System

- Kernel: mOS v0.8 based on Linux version 5.4.18
  - Linux experiments: `lwkctl -d` (No LWK partition)
  - mOS experiments: `lwkctl -c "lwkcpus=auto lwkmem=auto"`
    - 1-23, 25-47, 49-71, 73-95 = 92 (logical) CPUs out of 96 available
    - 94,371,840 kB + 94,371,840 kB = 180 GB of LWK memory
- Operating system distribution:
  - CentOS Linux 7 (Core)
- Boot command line:
  - ```
    BOOT_IMAGE=/vmlinuz-5.4.18_0.8.mos.old
    root=UUID=bf3d1d16-78d9-405b-ba67-3922095718e9 ro
    console=tty0 console=ttyS0,115200n8 selinux=0
    intel_pstate=disable nmi_watchdog=0 kernelcore=4G
    lwkmem=180G lwkcpus=0.1-23,49-71:24.25-47,73-95
    ```

System config on Slide 51
    ```

Software

- Intel® oneAPI beta 0.8
  - Python 3.7.7
  - TensorFlow*-2.1.0
- BERT March 11, 2020 (Smaller BERT Models)[5]
  - Vocab and config files from `uncased_L-12_H-768_A-12`
- TCMalloc version 4.4.5 (from gperftools-libs-2.6.1)
- JEmalloc version 5.2.1 (latest as of Sep 25, 2020)

[5] Iulia Turc et al. *Well-Read Students Learn Better: On the Importance of Pre-training Compact Models.* https://arxiv.org/abs/1908.08962. 2019.

Command

```
python run_classifier.py --task_name=MRPC --do_train=true \
  --do_eval=true --data_dir=${DATA_PATH}/datasets/glue/MRPC \
  --vocab_file=${DATA_PATH}/data/uncased_L-12_H-768_A-12/vocab.txt \
  --bert_config_file=${DATA_PATH}/data/uncased_L-12_H-768_A-12/
      bert_config.json \
  --save_checkpoint_steps=10000000 --max_seq_length=128 \
  --train_batch_size=64 --learning_rate=2e-05 \
  --num_train_epochs=1.0 --output_dir=${OUTPUT_PATH}
```

Random Sweep Settings

- `KMP_AFFINITY=granularity=thread,scatter`
- Timeout: 8 minutes

R2R Experiments

- `KMP_AFFINITY=granularity=thread,scatter`
- Timeout: 7 minutes

ResNet-50 Setup and Software Used

System Software

- Kernel 5.6.0-mos-20210115 mOS kernel
- TCMalloc version 4.4.5 (from gperftools-libs-2.6.1)
- Intel® oneAPI from `https://software.intel.com` using
  - BaseKit 2021.1.0.2659, HPCKit 2021.1.0.2684, and AIKit 2021.1.0.935.

System config on Slide 51

Kernel Build

mOS kernel based on `kernel.org` 5.6.0 with these kernel build parameters:

```
CONFIG_NO_HZ_FULL = y
CONFIG_RCU_NOCB_CPU = y
CONFIG_NODES_SHIFT = 4
CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE = y
CONFIG_CPU_FREQ_GOV_PERFORMANCE = y
CONFIG_CPU_FREQ_GOV_ONDEMAND = y
CONFIG_X86_INTEL_PSTATE = y
```

Source Code and Image Data

Code obtained from `https://github.com/pytorch/examples` and using `imagenet/main.py`. Experiments done with commit 49e1a8847c8c4d8d3c576479cb2fe2fd2ac583de. Modified `main.py` to measure execution time of `time()` function and to allow setting `torch.set_num_interop_threads` and `intraop` thread pool sizes from the command line.
The full set of image data[6] contains 1,281,167 images in 1,000 sub-directories. The training set we use is ILSVRC2012 and we take the first image in each of the 1,000 sub-directories of the training set to create a more managable subset for benchmarking.

[6] Jia Deng et al. *Imagenet Full (Fall 2011 release).* 2011. URL: `http://www.image-net.org/about-overview`.

Parameter Search

We did a parameter search for Linux and mOS to find good values and came up with these. The search was not exhaustive, so they may not be perfectly optimal. Also, they are quite likely different depending on system and software environment characteristics.

Parameter	Linux	LWK
`OMP_NUM_THREADS`	6	2
InterOp pool size	24	40
IntraOp pool size	32	40

Increasing the batch size from 32 to 96 helps both Linux and LWK. Batch sizes beyond that don't seem to help.

Configuring Linux for HPC

All Linux runs used (6, 32, 24) for thread pool sizes, while we used (2, 40, 40) for LWK runs. Linux used Transparent Huge Pages (THP), Each sample is 1 epoch using a batch size of 96.

```
/sys/kernel/mm/transparent_hugepage/defrag = defer+madvise
/sys/kernel/mm/transparent_hugepage/khugepaged/max_ptes_none = 0
/sys/kernel/mm/transparent_hugepage/enabled = always
/sys/devices/system/cpu/cpufreq/boost = 1
sudo swapoff -a
/proc/sys/vm/drop_caches = 3
```

CANDLE Uno Setup and Software Used

Software

- ECP CANDLE Uno benchmark
  - `https://github.com/ECP-CANDLE/Benchmarks/tree/loocv`
  - loocv branch
- Python 3.8.5
- Conda 4.9.2, TensorFlow* 2.2
- TCMalloc version 4.4.5

System config on Slide 51

Kernel Build

mOS kernel based on SLES 15 Service Pack 2 version 5.3.18-22 with these kernel build parameters:

```
CONFIG_NO_HZ_FULL = y
CONFIG_RCU_NOCB_CPU = y
CONFIG_NODES_SHIFT = 4
CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE = y
CONFIG_CPU_FREQ_GOV_PERFORMANCE = y
CONFIG_CPU_FREQ_GOV_ONDEMAND = y
CONFIG_X86_INTEL_PSTATE = y
```

Linux Setup

Command line:

```
LD_PRELOAD=/usr/lib64/libtcmalloc_minimal.so. \
python Pilot1/Uno/uno_tf2.py --train_sources CCLE \
--cache /tmp/cache/all --use_landmark_genes True \
--preprocess_rnaseq source_scale \
--no_feature_source True --no_response_source True
```

mOS Setup

Command line:

```
LD_PRELOAD=/usr/lib64/libtcmalloc_minimal.so.4 \
yod python Pilot1/Uno/uno_tf2.py --train_sources CCLE \
--cache /tmp/cache/all --use_landmark_genes True \
--preprocess_rnaseq source_scale --no_feature_source True \
--no_response_source True
```

Legal Notices and Disclaimers

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

*Other names and brands may be claimed as the property of others.

Intel technologies features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at `intel.com`, or from the OEM or retailer.

Optimization notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice Revision #20110804

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit `http://www.intel.com/performance`. *Other names and brands may be claimed as the property of others.

Notices, Disclosures, and Disclaimers

FTC Disclaimer For Performance Claims

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit `www.intel.com/benchmarks`.

Security Update Disclosure

Performance varies by use, configuration and other factors. Learn more at `www.Intel.com/Performance`.
Performance results are based on testing between September 11, 2019 and Marh 3, 2021 and may not reflect all publicly available updates. See Appendix for configuration details. No product or component can be absolutely secure.
Your costs and results may vary.
Intel technologies may require enabled hardware, software or service activation.

General Technology Disclaimer

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at `intel.com`.


intel