Intro to SUSTech TaiYi Supercomputer

Lenovo China
January 16, 2019

Agenda

1	TaiYi System overview
2	Access System
3	Filesystem
4	Running Jobs
5	Softwere Environment
6	MPI Libraries
7	Performance analysis
8	Intel MKL

Overview

TaiYi Overview

1.68PFlop/s HPL Performance Rank #127 on top500 Nov 2018

Basic Information

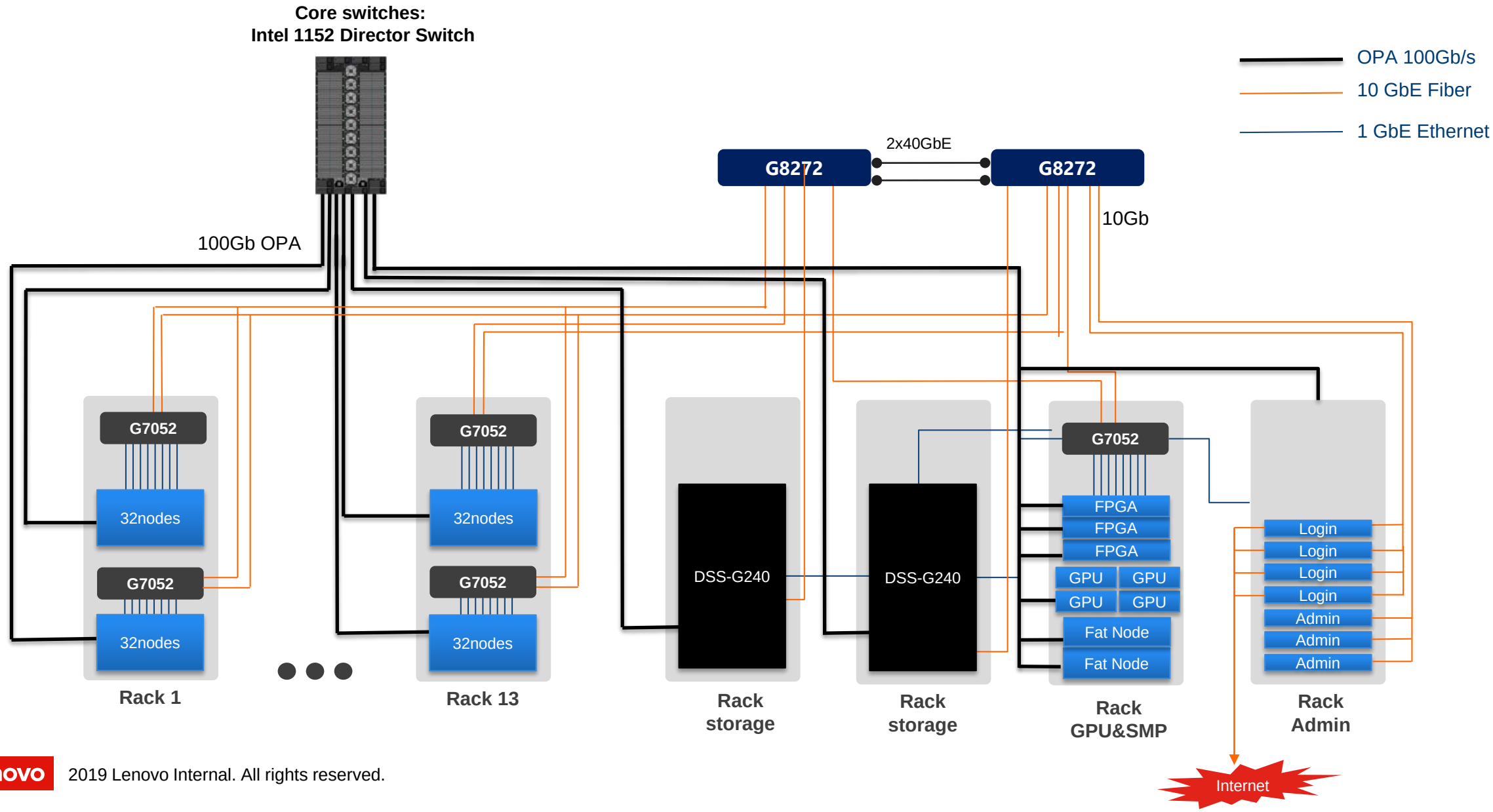
- “TaiYi” is a supercomputer based on Intel Xeon Gold processors from the Skylake generation.
- It is a Lenovo system composed of SD530 Compute Racks, an Intel Omni-Path high performance network inter-connect and running RedHat Linux Enterprise Server as operating system.
- Its current Linpack Performance is 1.67 Petafops.

Mission Statement

To empower our professors and researchers with state-of-the-art supercomputing facilities and enable them to carry out cutting-edge research in their respective domains

127	Southern University of Science and Technology China	**TaiYi** - ThinkSystem SD530, Xeon Gold 6148 20C 2.4GHz, Intel Omni-Path Lenovo	32,400	1,686.5	2,488.3

Architecture Overview



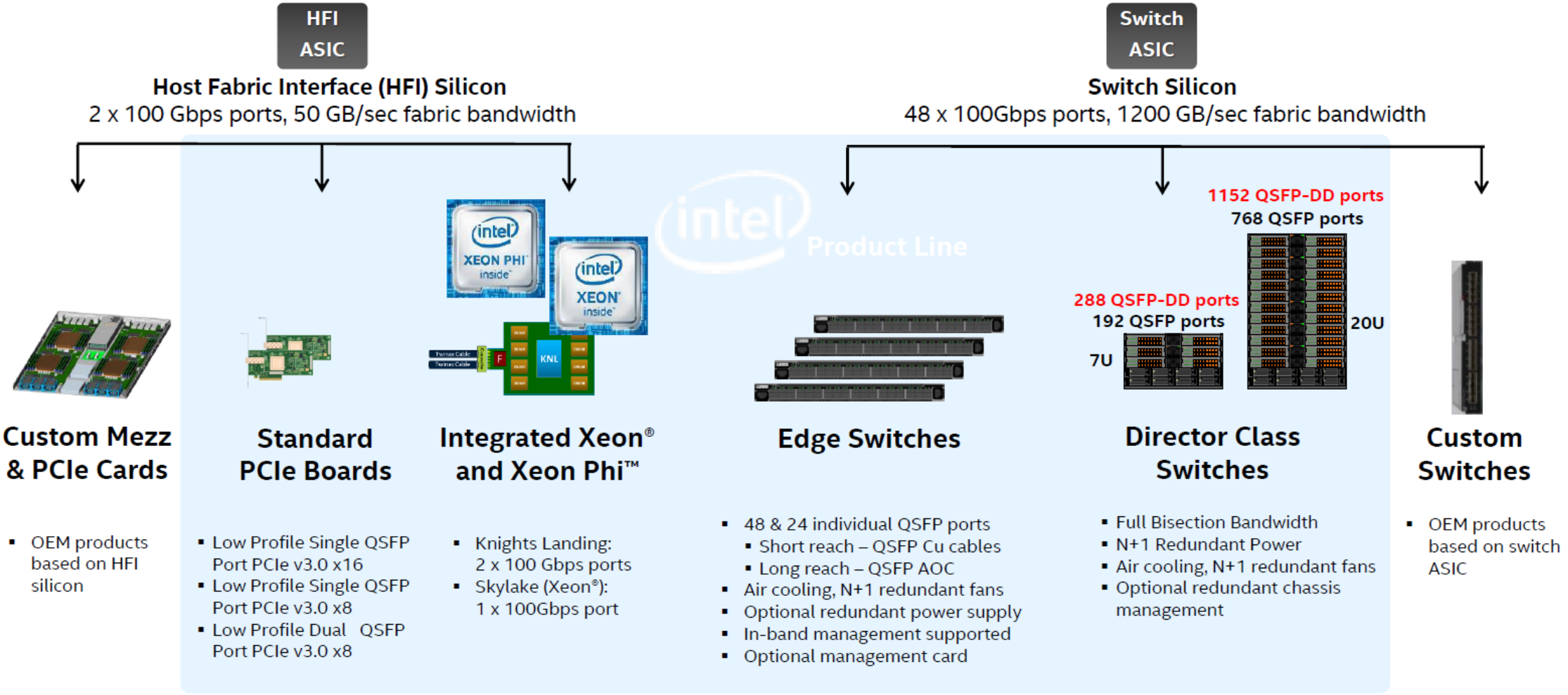
Summary of node types

- TaiYi consists five type of compute block:

Node Type	Qty	Model	Description
General Purpose	815	ThinkSystem SD530	2*Intel Gold 6148, 192GB, 240GB SSD, OPA
GPU	4	ThinkSystem SD530	2*Intel Gold 6148, 192GB, 240GB SSD, OPA, 2*NVIDIA V100 16GB
Big Memory	2	ThinkSystem SR950	8*Intel Platinum 8160, 6TB, 240GB SSD, 2*3.84TB SSD, OPA
FPGA	2		2*Intel Gold 6148, 192GB, 240GB SSD, OPA, 8*flyslice FA510Q
KNM	4		1*Intel KNM 7295, 192GB, 3.84TB SSD, OPA

Interconnect: Intel OPA

- Intel® Omni-Path Architecture (Intel® OPA), an element of Intel® Scalable System Framework (Intel® SSF), delivers the performance for tomorrow's high performance computing (HPC) workloads and the ability to scale to tens of thousands of nodes—and eventually more

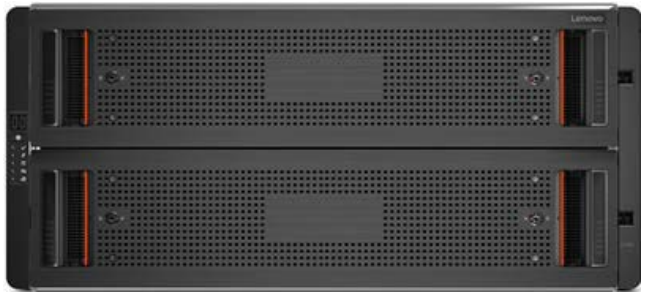


Lenovo Distributed Storage Solution for IBM Spectrum Scale (DSS-G)

- Scalable storage solution for HPC, BigData, and Cloud
  - Building-block approach to easily deploy and scale capacity and performance
- Leverages the latest Lenovo technology, along with IBM Spectrum Scale and RedHat Enterprise Linux
  - Cost-effective approach to meet the storage performance, capacity, and uptime demands in scale-out environments
- Validated, engineered solution delivered through Lenovo Scalable Infrastructure (LeSI)
  - Ensures interoperability, support and smooth deployment
  - Onsite installation and configuration provided through Lenovo or approved business partner
  - Available in 1410 rack, or can be installed in customer rack

IBM Spectrum Scale

redhat.



IBM Spectrum Scale™

A high-performance solution for managing data at scale with the distinctive ability to perform archive and analytics in place.

Operating systems supported

- RedHat Enterprise Linux, SUSE Linux Enterprise Server, Microsoft Windows Server 2012, Microsoft Windows 7, IBM AIX®; z Systems™

Hardware supported

- x86 architecture: Intel EM64T processors or AMD Opteron, IBM POWER® architecture, z Systems (Linux only)

Maximum number of files/file system

- 2^64 (9 quintillion) files per file system

Maximum file system size

- 2^99 bytes

Minimum/maximum number of nodes

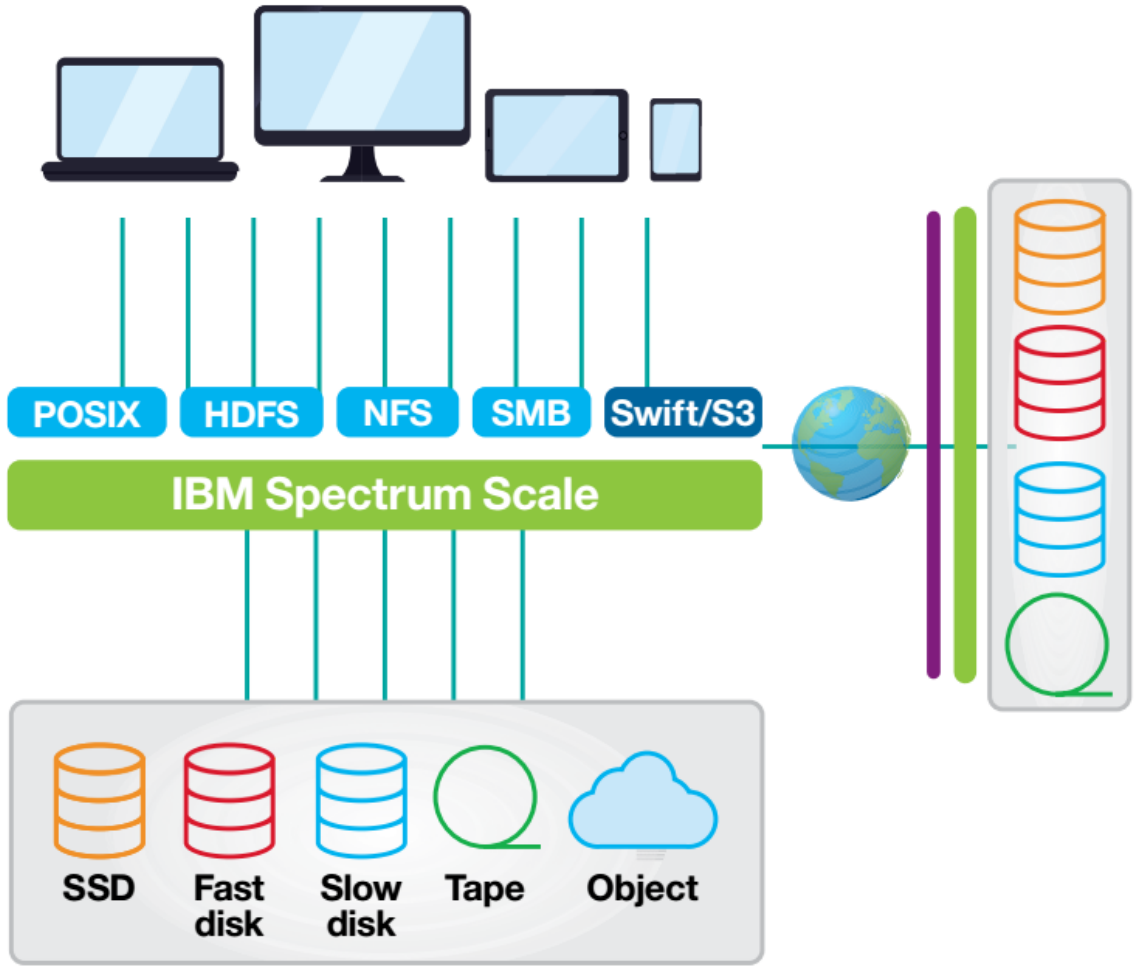
- 1 - 16,384

Protocols

- POSIX, GPFS, NFS v4.0, SMB v3.0
- Big data and analytics: Hadoop MapReduce
- Cloud: OpenStack Cinder (block), OpenStack Swift (object), S3 (object)

Cloud object storage

- IBM Cloud Storage System (Cleversafe), Amazon S3, IBM SoftLayer® Native Object, OpenStack Swift and Amazon S3 compatible providers



System Performance Summary

33144
High performance Compute Cores

1.7/2.5TF
Peak Performance

100Gb
Fat-Free Interconnect

166TB
Total Memory

45GB/s
Storage Bandwidth

Access System

Login Node

- You can connect to TaiYI using four public login nodes. Please note that only incoming connections are allowed in the whole cluster.
- The logins are:

Hostname	IP Addr
Login01	172.18.6.175
Login02	172.18.6.176
Login03	172.18.6.177
Login04	172.18.6.178

Note: ssh to compute nodes is denied. Users must submit jobs using IBM Spectrum LSF.

Password Management

- TaiYi use NIS to manage user account.
- A default password will be assigned to you, you must change the password after login.

- Command: yppasswd
- Linux command passwd will not work here

Transfer files

- There are two ways to copy files from/to the cluster
  - scp
  - sftp
- If you have massive data, please use hard drive or contact system admin
- Each login node have 10Gb uplink to campus network.
- Domain service is not enable on login nodes,

Filesystem

Filesystem

- Each user has several areas of disk space for storing files
- There are 4 different types of storage available inside a node:
  - Local SSD Disk: directory /tmp for node local scratch. The amount of space within the /tmp is about 200GB
  - GPFS filesystems
    - **/share**: Over this filesystem will reside the applications and libraries that have already been installed on the machine
    - **/work**: This is the home directories of all the users, default quota will be enforced
    - **/scratch**: Its intended use is to store temporary files of your jobs during their execution, There is no default quota on this filesystem, Data that is not part of an active job will be cleaned regularly.
    - **/data**: only on login nodes, This space is intended to store user important data. default quota will be enforced

IMPORTANT: It is your responsibility as a user of our facilities to backup all your critical data.

Quota

- The quotas are the amount of storage available for a user or a group
- A default value is applied to all users and groups and cannot be outgrown

- You can inspect your quota anytime you want using the following command from inside each filesystem

```
mmlsquota –u username --block-size auto
```

```
mmlsquota –g groupname --block-size auto
```

Running Jobs

Batch job & Queue

- IBM Spectrum LSF is the utility used for batch processing support, jobs must run through it.
- There are several queues present in the machines and different users may access different queues, all queues have different limits in amount of cores for the jobs and duration.
- The standard configuration and limits of the queues are the following:
- Default queue:
  - medium
  - Short

- bqueue
- bqueue -l

Queue name	prio	Maximum wallclock
Medium	10	72H
Short	40	24H
Smp	40	12H
Serial	30	24H
GPU	30	72H
Debug	30	1H

What is IBM Platform LSF?

IBM Platform LSF is the most powerful workload manager for demanding, distributed and mission-critical high performance computing environments.

A Simple Idea:

- Virtualize a heterogeneous infrastructure
- Optimally manage workloads according to policy

Key benefits:

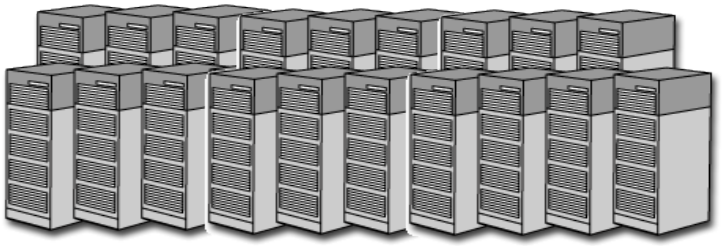
- Improves throughput - > competitive advantage
- Makes computers work harder - > financial advantage

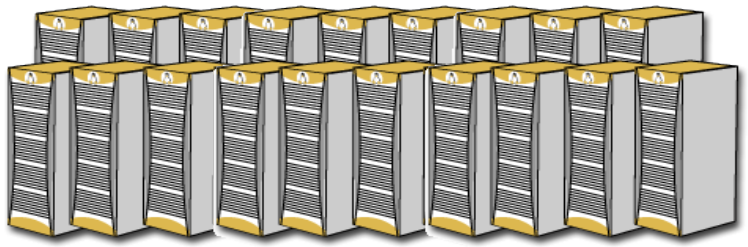
What it Does, How it Works

USER A

I need access to 100 computer systems for 48 hours to run a series of regression tests involving 600,000 test cases

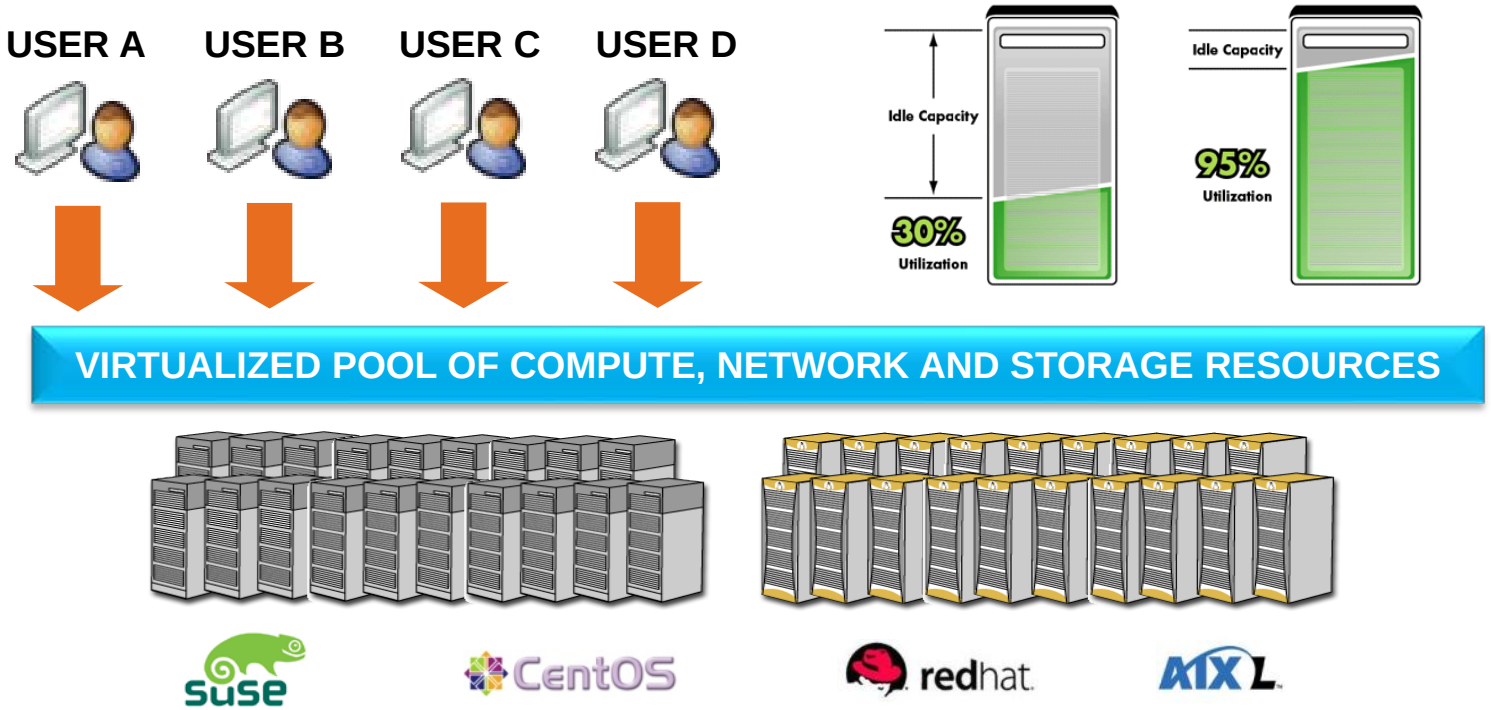
VIRTUALIZED POOL OF COMPUTE, NETWORK AND STORAGE RESOURCES





IBM Spectrum LSF

When users run applications under LSF they run more quickly, utilization is better, and applications are more reliable.



IBM Spectrum LSf can:

- Monitors systems and tracks dynamic resources (memory, swap & CPU) as well as static resources (OS, version, binary architecture, hardware).
- Selects the right system at the right time to run jobs held in configurable system wide queues.
- Ensures the reliable execution of jobs, re-starting them or migrating them as necessary in the case of hardware, network or software failures.
- Optimizes the allocation of various resources by user, project team or site, based on flexible policies that reflect business needs.

The LSF job script

- the user is required to provide a job script (also called batch file) with the specification of:
  - the resources requested, and the job constraints;
  - specific queue specification (more details later);
  - all what is needed for a working execution-environment.
- By resources we mean the number of **processors/cores/nodes**, the amount of **memory** needed (total and/or per process), and eventually some specific features (special hardware, for example), and so on.
- All these information must be written in a text file

LSF Script example

- For every line of resource specifications, #BSUB directive must be the first text of the line, and all specifications must come before any executable lines

```
#!/bin/bash
#
#BSUB –J jobname
#BSUB –n number-of-slots
#BSUB –o %J.stdout
#BSUB –e %J.stderr
#BSUB –R “span[ptile=40]”
#BSUB –q medium

module load mpi/intel/2018.4
module load vasp/5.4.4

mpirun vasp
```

- submit it by typing in a terminal the command:
  - bsub < job.lsf

Check job status

- Then you can check the status of your submission issuing the command:
- bjobs
- bjobs –l
- bjobs -W 99956
- bjobs -WL 99956

Basic Job Specifications

- These basic options are:

Basic LSF Job Specifications			
Specification	**Option**	**Example**	**Example-Purpose**
Job Name	-J [SomeText]	-J MyJob1	Set the job name to "MyJob1"
Shell	-L [Shell]	-L /bin/bash	Uses the bash shell to initialize the job's execution environment.
Wall Clock Limit	-W [hh:mm]	-W 24:15	Set wall clock limit to 24 hour 15 min
Core count	-n ##	-n 400	Assigns 400 job cores.
Cores per node	-R "span[ptile=##]"	-R "span[ptile=40]"	Request 40 cores per node.
Memory Per Core	-M [##]	-M 2GB	Sets the per process memory limit to 2GB.
Memory Per Core	-R "rusage[mem=[##]]"	-R "rusage[mem=2GB]"	Schedules job on nodes that have at least 2GBs available per core.
Combined stdout and stderr	-o [OutputName].%j	-o stdout1.%j	Collect stdout/err in stdout.[JobID]

Optional Job Specifications

- A variety of optional specifications is available to customize your job. The table below lists the specifications, which are most useful for users.

Optional LSF Job Specifications			
Specification	**Option**	**Example**	**Example-Purpose**
Set Allocation	-P #######	-P projectA	Set allocation to charge to Project projectA
Specify Queue	-q [queue]	-q short	Request only nodes in short subset.
Exclusive Node Usage	-x		Assigns a whole node exclusively for the job.
Specific node type	-R "select[gpu|phi]"	-R "select[gpu]"	Requests a node with a GPU to be used for the job.

Environment Variables

- Below is a list of most commonly used environment variables

Variable	Usage	Desc
Job ID	$LSB_JOBID	Batch job ID assigned by LSF.
Job Name	$LSB_JOBNAME	The name of the Job.
Queue	$LSB_QUEUE	The name of the queue the job is dispatched from.
Error File	$LSB_ERRORFILE	Name of the error file specified with a bsub -e.
Submit Directory	$LSB_SUBCWD	The directory the job was submitted from.
Hosts I	$LSB_HOSTS	The list of nodes that are used to run the batch job, repeated according to ptile value. *The character limit of LSB_HOSTS variable is 4096.
Hosts II	$LSB_MCPU_HOSTS	The list of nodes and the specified or default ptile value per node to run the batch job.
Host file	$LSB_DJOB_HOSTFILE	The hostfile containing the list of nodes that are used to run the batch job.

To see all relevant LSF environment variables for a job, add `env | grep LSB` to jobfile and submit job

Executable Commands

- After the resource, specification section of a job file comes the executable section.
- This executable section contains all the necessary UNIX, Linux, and program commands that will be run in the job.
- Some commands that may go in this section include, but are not limited to:
  - Changing directories
  - Loading, unloading, and listing modules
  - Launching software

Note:

- Remember that the UNIX is case sensitive and so uppercase and lowercase letters have different meanings, even in the job script specifications.
- Avoid using special characters, letters and spaces in the name of files, directories, and the job name. The script could be interpreted by the scheduler in unexpected ways.
- The default behaviour in LSF is to append the content of the file. So if you don't use unique names (like in the example), subsequent runs will add to the same file. To change the default behaviour to overwrite, use the flags -oo and -eo.
- If you do not specify the error file but only the output one, the standard output and standard error streams are merged and saved into the output file.
- The filename after the -o, -e flag can be a full path. If it is the name of an existing directory (ending with "/"), then LSF will automatically save as file with a name <jobid>.out.
- If you don't specify to save the output file somewhere, the content of the standard output and error streams will be appended to the report that is sent after completion (up to a system-predefined maximum size).

Manage LSF Jobs

- LSF Job Monitoring and Control Commands

Function	Command	Example
Submit a job	bsub < [script_file]	bsub < MyJob.LSF
Cancel/Kill a job	bkill [Job_ID]	bkill 101204
Check summary status of a single job	bjobs [job_id]	bjobs 101204
Check summary status of all jobs for a user	bjobs -u [user_name]	bjobs -u User1
Check detailed status of a single job	bjobs -l [job_id]	bjobs -l 101204
Modify job submission options	bmod [bsub_options] [job_id]	bmod -W 2:00 101204
Holding a batch job	bstop [jobid]	bstop 101204
Releasing a batch job	bresume [jobid]	bresume 101204

```
$ man [command]
```

LSF Job states

- The basic job states are:
  - **PEND** - the job is in the queue, waiting to be scheduled
  - **PSUSP** - the job was submitted, but was put in the suspended state (ineligible to run)
  - **RUN** - the job has been granted an allocation. If it's a batch job, the batch script has been run
  - **DONE** - the job has completed successfully
- A pending job can remain pending for a number of reasons
  - **Dependency** - the pending job is waiting for another job to complete
  - **Priority** - the job is not high enough in the queue
  - **Resources** - the job is high in the queue, but there are not enough resources to satisfy the job's request

MPI Job Templates

```
#!/bin/bash
#
#BSUB -J MPIJob                 ### set the job Name
#BSUB -q short                  ### specify queue
#BSUB -n 40                     ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”       ### ask for 40 cores per node
#BSUB -W 10:00                  ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out          ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err          ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute
# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
module load mpi/intel/2018.4

### This uses the LSB_DJOB_NUMPROC to assign all the cores reserved
### This is a very basic syntax. For more complex examples, see the documentation
mpirun -np $LSB_DJOB_NUMPROC ./MPI_program
```

OpenMP Jobs under LSF

```
#!/bin/bash
#
#BSUB -J OpenMPjob            ### set the job Name
#BSUB -q short                ### specify queue
#BSUB -n 40                   ### ask for number of cores (default: 1)
#BSUB -R "span[hosts=1]"      ### specify that the cores MUST BE on a single host!
#BSUB -W 16:00                ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out        ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err        ### -o and -e mean append, -oo and -eo mean overwrite

# set OMP_NUM_THREADS _and_ export!
OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
export OMP_NUM_THREADS

# Program_name_and_options
./openmp_program [options]
```

If you make use of special environment variables for your OpenMP program, remember to put them in your script (use the same syntax as the `OMP_NUM_THREADS` line in the script).

Hybrid MPI/OpenMP jobs under LSF

- MPI and OpenMP can be combined to write programs that exploit the possibility given by multi-core SMP machines, and the possibility of using the Message Passing Interface based communications between nodes on the cluster

```
#!/bin/bash
#
#BSUB –J Hybrid_MPI_OpenMP    ### set the job Name
#BSUB -q short                ### specify queue
#BSUB -n 400                  ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”     ### ask for 40 cores per node
#BSUB -W 10:00                ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out        ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err        ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute
# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
module load mpi/intel/2018.4

# -- OpenMP environment variables --
Num=$(echo $LSB_SUB_RES_REQ | sed -e 's/span\[ptile=/ /g' -e 's/\]/ /g')
OMP_NUM_THREADS=$Num
export OMP_NUM_THREADS

mpirun -npernode 1 ./mpi_bybrid_program
```

Job Dependencies under LSF

- There may be situations where there are obvious dependencies between jobs, and one cannot start before another has finished(for example because it is dependent on the results of the previous jobs).
- LSF provides a dependency feature that can be used for this purpose. One can therefore specify in the batch script of a job, that this job has a dependency on another one. This is done with the syntax
- #BSUB -w "done(<jobid>)"

- This will ensure that the current job is started only after the job with the corresponding jobid is completed successfully. Some of the most common options are
  - **done** : job state done
  - **ended** : job state exit or done
  - **exit** : job state is EXIT and eventually with a specific exit code exit(<jobid>,<exit-code>)
  - **started** : job state DONE, EXIT, USUSP, SSUSP or RUN

- To know the jobs that a job depends on, type: bjdepinfo <jobid>
- To show the jobs that depend on a job, use the -c option: bjdepinfo -c <jobid>

Recommended Settings for Large Jobs

- For jobs larger than 2000 cores (50+ nodes), the following MPI settings are recommended to reduce the MPI startup time:

```
export I_MPI_HYDRA_BOOTSTRAP=lsf
export I_MPI_HYDRA_BRANCH_COUNT=XXX
export I_MPI_LSF_USE_COLLECTIVE_LAUNCH=1
```

- The XXX number should match the number of nodes (#Nodes = #Cores / #CoresPerNode) that your job will request.

Batch Job Translation Guide

- We provide some basic informations for the purposes of moving from SLURM/PBS/Torque to LSF

User Commands	LSF	Slurm	PBS/Torque
Job submission	bsub [script_file]	sbatch [script_file]	qsub [script_file]
Job deletion	bkill [job_id]	scancel [job_id]	qdel [job_id]
Job status (by job)	bjobs [job_id]	squeue --job [job_id]	qstat [job_id]
Job status (by user)	bjobs -u [user_name]	squeue -u [user_name]	qstat -u [user_name]
Queue list	bqueues	squeue	qstat -Q

Environment Variables	LSF	Slurm	PBS/Torque
Job ID	$LSB_JOBID	$SLURM_JOBID	$PBS_JOBID
Submit Directory	$LSB_SUBCWD	$SLURM_SUBMIT_DIR	$PBS_O_WORKDIR

Job Specifications

The table below lists various directives that set characteristics and resources requirements for jobs

Job Specification	LSF	Slurm	PBS/Torque
Script directive	#BSUB	#SBATCH	#PBS
Node Count	N/A (Calculated from: CPUs/CPUs_per_node)	-N [min[-max]]	-l nodes=[count]
CPUs Per Node	-R "span[ptile=count]"	--ntasks-per-node=[count]	-l ppn=[count]
CPU Count	-n [count]	-n [count]	N/A (Calculated from: CPUs_per_node * Nodes)
Wall Clock Limit	-W [hh:mm]	-t [min] or -t[days-hh:mm:ss]	-l walltime=[hh:mm:ss]
Memory Per Core	-M [mm] AND -R "rusage[mem=mm]" * Where mm is in MB	--mem-per-cpu=mem[M/G/T]	-l mem=[mm] * Where mm is in MB
Standard Output File	-o [file_name]	-o [file_name]	-o [file_name]
Standard Error File	-e [file_name]	-e [file_name]	-e [file_name]
Combine stdout/err	(use -o without -e)	(use -o without -e)	-j oe (both to stdout) OR -j eo (both to stderr)
Event Notification	-B and/or -N * For Begin and/or End	--mail-type=[ALL/END] * See 'man sbatch' for all types	-m [a/b/e] * For Abort, Begin, and/or End
Email Address	-u [address]	--mail-user=[address]	-M [address]
Job Name	-J [name]	--job-name=[name]	-N [name]
Account to charge	-P [account]	-account=[account]	-l billto=[account]
Queue	-q [queue]	-q [queue]	-q [queue]

Software Environment

Software env

- All software and numerical libraries available at the cluster can be found at /share. If you need something that is not there please contact us to get it installed.
  - /share/apps: applications such as vasp, openmpi, cae, bio, etc.
  - /share/base: basic libraries, gcc, etc.
  - /share/intel: Intel cluster tools, Intel C/C++/Fortran, MKL, Intel MPI, Vtune, etc.

Compilers

- In the cluster you can find these C/C++ compiles:
  - icc/icpc: Intel C/C++ Compilers
  - gcc/g++: GNU Compilers for C/C++
  - ifort: Intel Fortran Compiler
  - gfortran: GNU Compiler for Fortran

- To find out what version available , you can use the following command:
  - $ module avail intel
  - $ module avail gcc

Distributed Memory Parallelism

- To compile MPI programs it is recommended to use the following handy wrappers: **mpicc**, **mpicxx** for C and C++ source code, **mpif77**/**mpif90** for Fortran source code. You need to choose the Parallel environment first. These wrappers will include all the necessary libraries to build MPI applications without having to specify all the details by hand.

- For Intel MPI library, we are able to use **mpiicc**/**mpiicpc**/**mpiifort** wrappers, which combine intel mpi and intel compilers.

Shared Memory Parallelism

- OpenMP directives are fully supported by the Intel C and C++ compilers. To use it, the flag -qopenmp must be added to the compile line.

- You can also mix **MPI + OPENMP** code using -qopenmp with the mpi wrappers

Optimization flags(Intel compiler)

Flag	Description
-O3	Performs -O2 optimizations and enables more aggressive loop transformations.
-xHost	Tells the compiler to generate vector instructions for the highest instruction set available on the host machine.
-fast	Convenience flag. In linux this is shortcut for -ipo, -O3, -no-prec-div, -static, and -xHost
-ip	Perform inter-procedural optimization within the same file
-ipo	Perform inter-procedural optimization between files
-parallel	enable automatic parallelization by the compiler (very conservative)
-opt-report=[n]	generate optimization report. n represent the level of detail (0 ..3, 3 being most detailed)
-vec-report[=n]	generate vectorization report. n represents the level of detail (0..7 , 7 being most detailed)

For highest performance, we recommend using option：-xSKYLAKE-AVX512

Use a Specific Memory Model: **-mcmodel=medium** and **-shared-intel** If you get a link time error relating to R_X86_64_PC32

Flags affecting floating point operations (Intel compiler)

- Some optimization might affect how floating point arithmetic is performed. This might result in round off errors in certain cases. The table below shows a number of flags to instruct the compiler how to deal with floating point operations:

Flag	Description
-fp-model precise	disable optimizations that are not value safe on floating point data (See man page for other options)
-fltconsistency	enables improved floating-point consistency. This might slightly reduce execution speed.
-fp-speculation=strict	tells the compiler to disable speculation on floating-point operations (See man page for other options)

Modules Environment

- The Environment Modules package (http://modules.sourceforge.net/) provides a dynamic modifcation of a user's environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically, in a clean fashion.

- Installed software packages are divided into serval categories:
  - Compilers: Compiler suites available for the system (intel, gcc, . . .)
  - Parallel: openmpi/mvapich/intelmpi, etc
  - Tools: useful tools which can be used at any time (pdsh, darshan, . . .)
  - Applications: High Performance Computers programs (VASP, . . .)
  - Libraries: Those are tipycally loaded at a compilation time, they load into the environment the correct compiler and linker flags (FFTW, LAPACK, . . .)

Module usage

- To find out what software packages are available under the Modules system use one of the following commands:

```
$ module avail [packageName]
```

- In order to use some software on our clusters, the module for the software must be loaded first. To load the module for a software package, use the following command:

```
$ module load packageName/version
```

- To find out what packages/modules you have loaded on your current session, use the following command:

```
$ module list
```

- To remove one module from your current session, use the following command:

```
$ module unload [packageName]
```

- To remove ALL modules from your current session, use the following command:

```
$ module purge
```

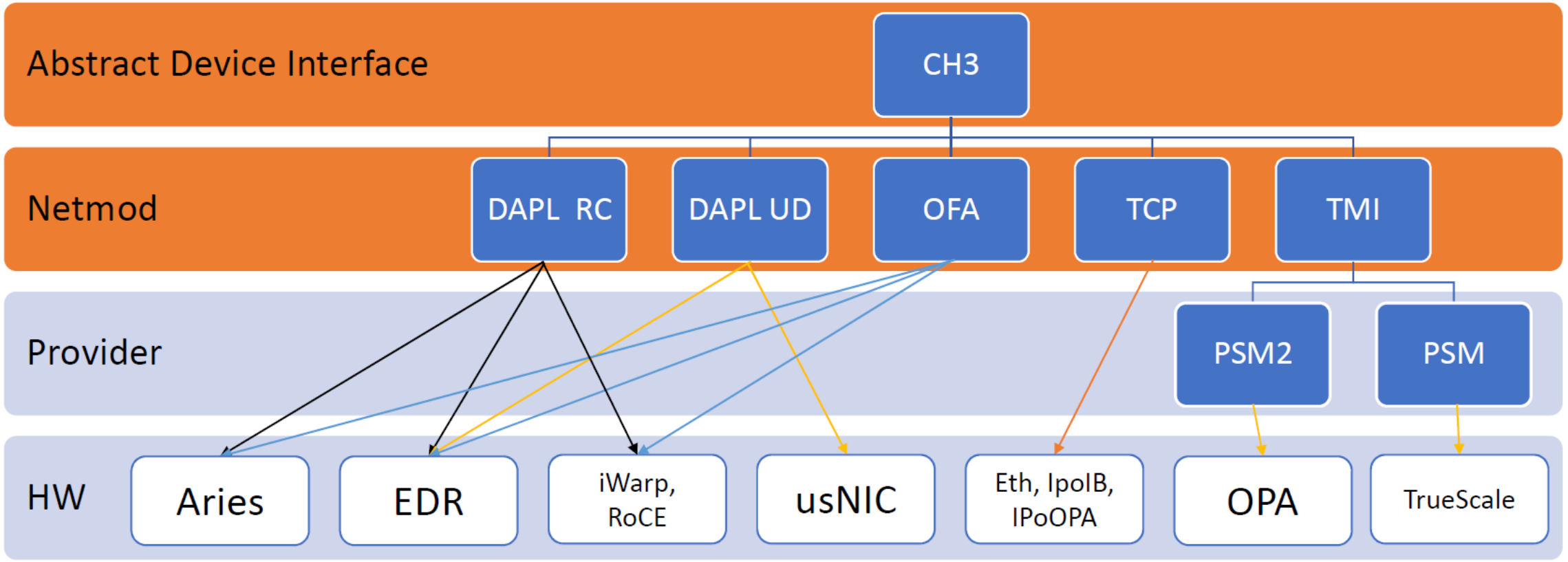
Write your own modulefile

- Define MODULEPATH Environment Variables

```
#%Module

#
# Author                : Qiwang,Chen
# Modified              : Thu Jan 10 17:37:52 CST 2019
#

module-whatis "Enable usage for mpi_trace version 0.11-b78a27b"

setenv MPI_TRACEROOT /share/base/mpitrace/0.11-b78a27b

prepend-path PATH            $env(MPI_TRACEROOT)/bin
prepend-path INCLUDE         $env(MPI_TRACEROOT)/include
prepend-path LD_LIBRARY_PATH $env(MPI_TRACEROOT)/lib
prepend-path MANPATH         $env(MPI_TRACEROOT)/share/man
```

MPI Library

Intel® MPI 2018 SW stack/ecosystem



Each transport layer required independent optimization

Intel mpi usage

- $ source /share/intel/2018u4/impi/2018.4.274/intel64/bin/mpivars.sh
- $ module av mpi
- $ module load mpi/intel/2018.4
- In IFS 10.8, the OFI/PSM2 provider turns on PSM2 multiple endpoint support by default and as the result PSM2 shared context error(can not open hfi …), to revert this behavior please explicitly turn of multi-EP support by setting:
  - PSM2_MULTI_EP=0

Opensource mpi libraries

- OpenMPI, MVAPICH
- To build you own openmpi, please check config output: PSM2/LSF

```
Transports
----------------------
Cisco usNIC: yes
Cray uGNI (Gemini/Aries): no
Intel Omnipath (PSM2): yes
Intel SCIF: no
Intel TrueScale (PSM): no
Mellanox MXM: no
Open UCX: no
OpenFabrics Libfabric: yes
OpenFabrics Verbs: yes
Portals4: no
Shared memory/copy in+copy out: yes
Shared memory/Linux CMA: yes
Shared memory/Linux KNEM: yes
Shared memory/XPMEM: no
TCP: yes
```

```
Resource Managers
----------------------
Cray Alps: no
Grid Engine: no
LSF: yes
Moab: no
Slurm: yes
ssh/rsh: yes
Torque: no
```

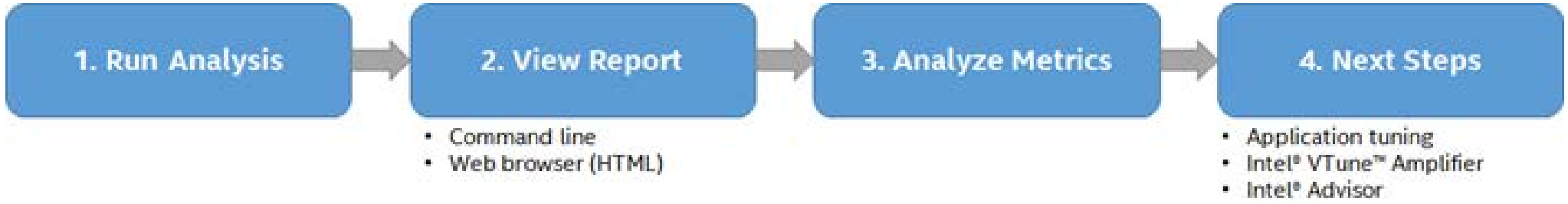
```
# OSU MPI Bandwidth Test (Version 2.0)
# Size          Bandwidth (MB/s)
1               0.764273
2               2.583719
4               7.653185
8               15.529966
16              34.431356
32              69.514725
64              139.040702
128             218.157069
256             435.291548
512             866.630642
1024            1708.908343
2048            3203.145219
4096            5059.645795
8192            6572.488659
16384           5341.648766
32768           7655.100320
65536           9154.967758
131072          12038.723085
262144          12212.343447
524288          12290.517632
1048576         12343.011731
2097152         12378.688587
4194304         12375.198676
```

Performance analysis

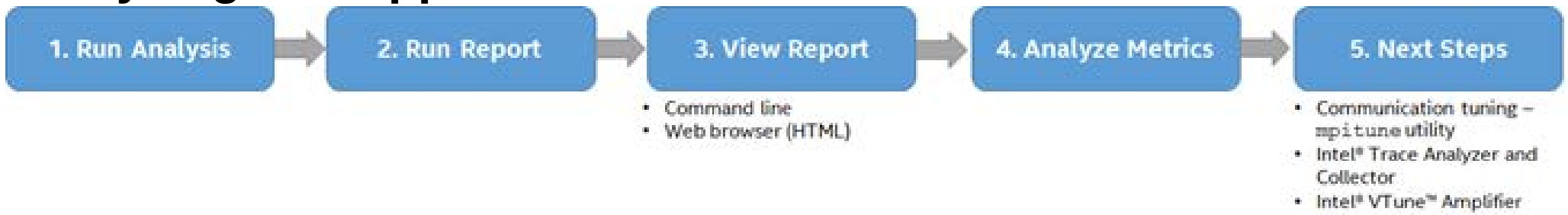
Intel APS

- Use Application Performance Snapshot for a quick view into a shared memory or MPI application's use of available hardware (CPU, FPU, and Memory).

Analyzing Shared Memory Applications



Analyzing MPI Applications



Running APS

- Before running the tool, you need to set up your environment :
  - $ source /share/intel/aps2018u3/apsvars.sh
- Run the following command:
  - $ aps <my app> <app parameters>
- After the analysis completes, a report appears in the command window: aps_result_01012017_1234.html
- You can run aps under lsf, put commands into lsf script and submit jobs

Run APS with MPI

• Before running the tool, you need to set up your environment :

```
$ source /share/intel/aps2018u3/apsvars.sh
```

• Run the following command:

```
$ mpirun –np # aps vasp
```

• Run the following command to complete the analysis:

```
$ aps --report=aps_result_20181211
```

You download HTML report and open in a supported browser.

APS sample output

Intel® VTune™ Amplifier

Application Performance Snapshot

Application: *main*
Report creation date: *2018-12-11 02:15:37*
Number of ranks: *1024*
Ranks per node: *32*
HW Platform: *Intel(R) Xeon(R) Processor code named Skylake*
Logical Core Count per node: *40*
Collector type: *Driverless Perf per-process counting*

245.23s
Elapsed Time

1569.87
SP GFLOPS

1.08
CPI
(MAX 1.19, MIN 0.97)

Your application is memory bound.

Use memory access analysis tools like Intel® VTune™ Amplifier for a detailed metric breakdown by memory hierarchy, memory bandwidth, and correlation by memory objects.

	Current run	Target	Delta
MPI Time	28.09%	<10%	
Memory Stalls	45.01%	<20%	
FPU Utilization	0.90%	>50%	
I/O Bound	0.00%	<10%	

APS Sample output

MPI Time

68.87s
28.09% of Elapsed Time

MPI Imbalance
12.81s
5.22% of Elapsed Time

TOP 5 MPI Functions	%
Waitall	17.76
Barrier	4.65
Init	2.36
Reduce	1.26
Isend	1.06

Memory Footprint

Resident	PEAK	AVG
Per node:	3049.15 MB	3038.57 MB
Per rank:	97.65 MB	94.96 MB

Virtual	PEAK	AVG
Per node:	39694.50 MB	39391.57 MB
Per rank:	1294.47 MB	1230.99 MB

Memory Stalls

45.01% of pipeline slots

Cache Stalls
14.23% of cycles

DRAM Stalls
17.65% of cycles

Average DRAM Bandwidth
Not Available

NUMA
10.93% of remote accesses

FPU Utilization

0.90%

SP FLOPs per Cycle
0.58 Out of 64.00

Vector Capacity Usage
15.96%

FP Instruction Mix
% of Packed FP Instr.: 10.84%
% of 128-bit: 1.80%
% of 256-bit: 9.05%
% of 512-bit: 0.00%
% of Scalar FP Instr.: 89.16%

FP Arith/Mem Rd Instr. Ratio
0.96

FP Arith/Mem Wr Instr. Ratio
1.88

I/O Bound

0.00%
(AVG 0.00, PEAK 0.08)

Read
AVG 2.1 KB, MAX 2.1 KB

Write
AVG 4.8 MB, MAX 5.9 MB

Mpi_trace

- **MPITRACE** is a tool developed and supplied by the IBM teams. With this tool, the behaviour of an MPI/OpenMP application can be analysed and its critical parts easily identified.

- The utilisation of MPITRACE takes place in three steps:
  - Instrumentation of the application
  - Execution of the instrumented application
  - Analysis of the results

Mpi_trace usage

- To instrument your MPI application, you need to position (before linking) the module

```
$ module load mpitrace/0.11-b78a27b-impi
```

- Set Environment Variables
- By default, the process 0 profile is collected. After the execution, the following files are generated:
  - mpi_profile.pid.0

```
export PROFILE_IMBALANCE=no
export PROFILE_COMMUNICATORS=no
export PROFILE_VPROF=no
export PROFILE_JIO=yes
export JIO_LEVEL=DETAILED
export PROFILE_HPM=no
export PROFILE_ENERGY=yes
export PROFILE_XML=no
export SAVE_ALL_TASKS=no

export LD_PRELOAD=/share/base/mpitrace 0.11-b78a27b/lib/impi5/libmpitrace.so
```

Mpitrace sample output

```
Data for MPI rank 0 of 16384:
Times and statistics from MPI_Init() to MPI_Finalize().
---------------------------------------------------------------
MPI Routine                        #calls     avg. bytes      time(sec)
---------------------------------------------------------------
MPI_Comm_size                           2            0.0         0.000
MPI_Comm_rank                           2            0.0         0.000
MPI_Isend                        15949418          196.3         6.471
MPI_Irecv                        15949418          196.3         4.434
MPI_Waitall                      15949418            0.0       614.012
MPI_Bcast                               2        65536.0         0.164
MPI_Reduce                          71510            8.0       128.798
MPI_Allreduce                         274          142.0         0.304
---------------------------------------------------------------
total communication time = 754.183 seconds.
number of MPI calls      = 47920044.
total real communication time = 754.183 seconds.

total MPI-IO time        = 0.000 seconds.
number of MPI-IO calls   = 0.
total IO time            = 0.000 seconds.
number of IO calls       = 0.
total mem time           = 0.000 seconds.
number of mem calls      = 0.
total elapsed time       = 452.280 seconds.
user cpu time            = 389.501 seconds.
system time              = 4.859 seconds.
max resident set size    = 130.953 MBytes.
max virtual memory size  = 1592.941 MBytes.
…
Communication summary for all tasks:

  minimum communication time = 1.280 sec for task 0
  median  communication time = 1.283 sec for task 215
  maximum communication time = 1.284 sec for task 9
```

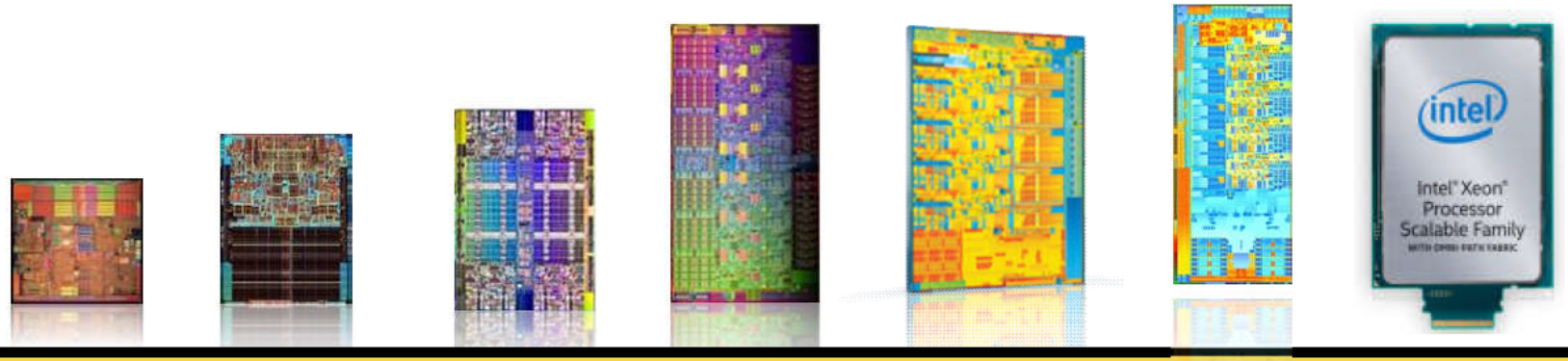
```
---------------------------------------------------------------
Message size distributions:

MPI_Isend                  #calls    avg. bytes      time(sec)
                          7774686          64.0          3.140
                          7774686         128.0          2.986
                           400040        4096.0          0.344
                                6        4128.0          0.000

MPI_Irecv                  #calls    avg. bytes      time(sec)
                          7774686          64.0          2.113
                          7774686         128.0          1.658
                           400040        4096.0          0.663
                                6        4128.0          0.000

MPI_Bcast                  #calls    avg. bytes      time(sec)
                                2       65536.0          0.164

MPI_Reduce                 #calls    avg. bytes      time(sec)
                            71510           8.0        128.798

MPI_Allreduce              #calls    avg. bytes      time(sec)
                               256          8.0          0.188
                                18       2048.0          0.116

---------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 660.709 sec for task 5400
  median  communication time = 729.643 sec for task 4265
  maximum communication time = 813.646 sec for task 6

  minimum real communication time = 660.709 sec for task 5400
  median  real communication time = 729.643 sec for task 4265
  maximum real communication time = 813.646 sec for task 6
```

Intel MKL

Intel® Math Kernel Library

- Speeds computations for scientific, engineering,financial and machine learning applications
- Provides key functionality for dense and sparse
- linear algebra (BLAS, LAPACK, PARDISO), FFTs,vector math, summary statistics, deep learning primitives and more
- Provides scientific programmers and domain scientists
  - Interfaces to de-facto standard APIs from C++,
  - Fortran, C#, Python and more
  - Support for Linux*, Windows* and OS X* operating systems
  - Extract great performance with minimal effort
- Optimized for single core vectorization and cache utilization
- Automatic parallelism for multi-core and many-core
- Scales to clusters

Automatic Dispatching to Tuned ISA-specific Code Paths



	Intel® Xeon® Processor 64-bit	Intel® Xeon® Processor 5100 series	Intel® Xeon® Processor 5500 series	Intel® Xeon® Processor 5600 series	Intel® Xeon® Processor E5-2600 v2 series	Intel® Xeon® Processor E5-2600 v3 series v4 series	Intel® Xeon® Scalable Processor[1]
Up to Core(s)	1	2	4	6	12	18-22	28
Up to Threads	2	2	8	12	24	36-44	56
SIMD Width	128	128	128	128	256	256	512
Vector ISA	Intel® SSE3	Intel® SSE3	Intel® SSE4- 4.1	Intel® SSE 4.2	Intel® AVX	Intel® AVX2	Intel® AVX-512

MKL Usage

- MKL is part of the Intel compiler releases. Once you load a compiler module (for example, module load intel/2019.4) the environment variable MKLROOT is automatically set and the path to the MKL library is automatically included in your default path.

```
echo $MKLROOT
/share/intel/2019u4/compilers_and_libraries_2019.5.274/linux/mkl
```

- Intel employs a layered model for MKL. There are three layers:
  - the interface layer: The LP64 interface is for using 32-bit integer type and the ILP64 interface is for using 64-bit integer type.
  - the threading layer:
    - Sequential
    - Threaded
  - and the computational layer.

Compiling and Linking with Intel MKL

- Use the online Intel MKL Link Line Advisor to determine the libraries and options to specify on your link or compilation line
- For example, to do a dynamic linking and use parallel Intel MKL supporting LP64 interface for an OpenMP code, the MKL Link Line Advisor suggests:

```
-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm -ldl
```

- The -mkl Switch: The Intel compiler provides an -mkl switch to link to certain parts of the MKL library. In many cases, using this switch (instead of having to explicitly specify the MKL libraries) is all you need

```
-mkl[=]
        parallel   - link using the threaded Intel(R) MKL libraries. (Default)
        sequential - link using the non-threaded Intel(R) MKL libraries
        cluster    - link using the Intel(R) MKL Cluster libraries plus
                     the sequential Intel(R) MKL libraries
```

```
--start-group -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 --end-group

--start-group -lmkl_intel_lp64 -lmkl_sequential -lmkl_core --end-group

--start-group -lmkl_intel_lp64 -lmkl_cdft_core -lmkl_scalapack_lp64 \
 -lmkl_blacs_lp64 -lmkl_sequential -lmkl_core -liomp5 --end-group
```

Intel MKL Link Line Advisor

Intel® Math Kernel Library (Intel® MKL) Link Line Advisor v4.7 Reset

Select Intel® product:	Intel(R) MKL 2018.0
Select OS:	Linux*
Select usage model of Intel® Xeon Phi™ Coprocessor:	None
Select compiler:	Intel(R) Fortran
Select architecture:	Intel(R) 64
Select dynamic or static linking:	Dynamic
Select interface layer:	32-bit integer
Select threading layer:	OpenMP threading
Select OpenMP library:	Intel(R) (libiomp5)
Select cluster library:	☐ Cluster PARDISO (BLACS required) ☐ CDFT (BLACS required) ☐ ScaLAPACK (BLACS required) ☐ BLACS
Select MPI library:	<Select MPI>
Select the Fortran 95 interfaces:	☐ BLAS95 ☐ LAPACK95
Link with Intel® MKL libraries explicitly:	☐

Use this link line:

```
 -L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
-liomp5 -lpthread -lm -ldl
```

Compiler options:

```
 -I${MKLROOT}/include
```

Notes:

o Set the INCLUDE, MKLROOT, LD_LIBRARY_PATH, LIBRARY_PATH, CPATH and NLSPATH environment variables in the command shell using one of mklvars script files in the 'bin' subdirectory of the Intel(R) MKL installation directory. Please see also the Intel(R) MKL User Guide.

o Please be sure that you have used the recommended compiler options for the selected interface layer. Caution: linking Intel(R) MKL libraries with your objects compiled for different interface layer may lead to run-time errors.

MKL FFTW Interface

- FFTW is a free collection of C routines for computing the discrete Fourier Transform (DFT) in one or more dimensions, and provides portability across platforms.
- The Intel Math Kernel Library (MKL) offers FFTW2 (for version 2.x) and FFTW3 (for version 3.x) interfaces to the Intel MKL Fast Fourier Transform and Trigonometric Transform functionality. These interfaces enable applications using FFTW to gain performance with Intel MKL without changing the application source code.

- The MKL interfaces are available in the form of source files (under /share/intel/ [Version]/mkl/interfaces), which can be built into libraries and then linked when you compile your application.
  - fftw2xc
  - fftw2x_cdft
  - fftw2xf
  - fftw3xc
  - fftw3x_cdft
  - fftw3xf

```
module load intel/2018.4

ifort -O3                                                          \
      -o fftw_example.exe fftw_example.f                           \
      -I/share/intel/2018u4/mkl/include/fftw                        \
       /share/intel/2018u4/mkl/interfaces/fftw3xf/libfftw3xf_intel.a  \
      -mkl
```

Intel MKL 2019 New Features and Optimizations

Deep Learning for Knights Mill

New 8-bit and 16-bit Integer Matrix Multiply

Optimized SGEMM

DNN Convolution & Inner Product Optimizations

BLAS

BLAS Group & Batch: Efficiency & Performance

2x Speedup over TRSM

New - Batched Triangular Solve Matrix

~1.6x Speedup over GEMM

Faster GEMM_BATCH

Compact BLAS and LAPACK functions

LAPACK

Aasen's Algorithm

Factor & Solve Routines

Improved ScaLAPACK performance

Without Pivoting

Fast LU factorization and Inverse without pivoting

New Vector Math Functions

NEW!

24 Added

v?Fmod, v?Remainder, v?Powr, v?Exp2; v?Exp10; v?Log2; v?Logb; v?Cospi; v?Sinpi; v?Tanpi; v?Acospi; v?Asinpi; v?Atanpi; v?Atan2pi; v?Cosd; v?Sind; v?Tand; v?CopySign; v?NextAfter; v?Fdim; v?Fmax; v?Fmin; v?MaxMag; v?MinMag

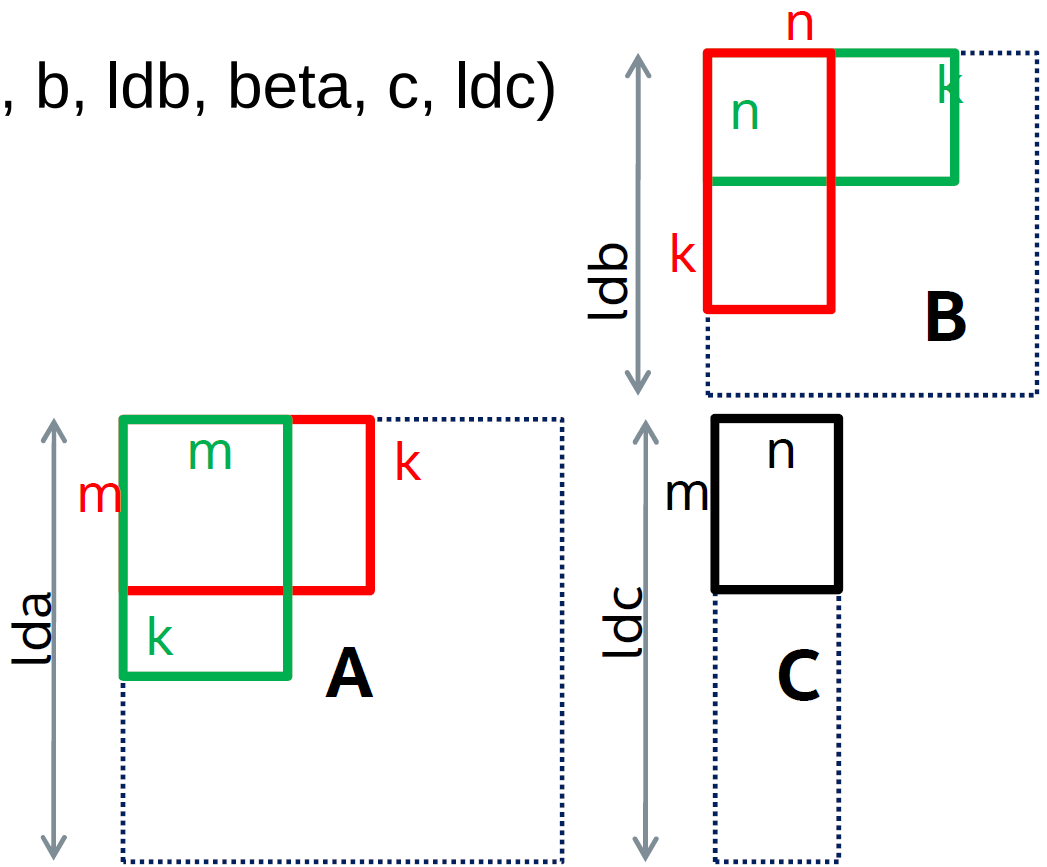
Matrix-matrix multiply

- Part of the Intel MKL BLAS (Basic Linear Algebra Subprograms)
- Important for scientific, engineering, and machine learning
- Applications
- *GEMM(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

C = alpha op(A) * op(B) + beta C

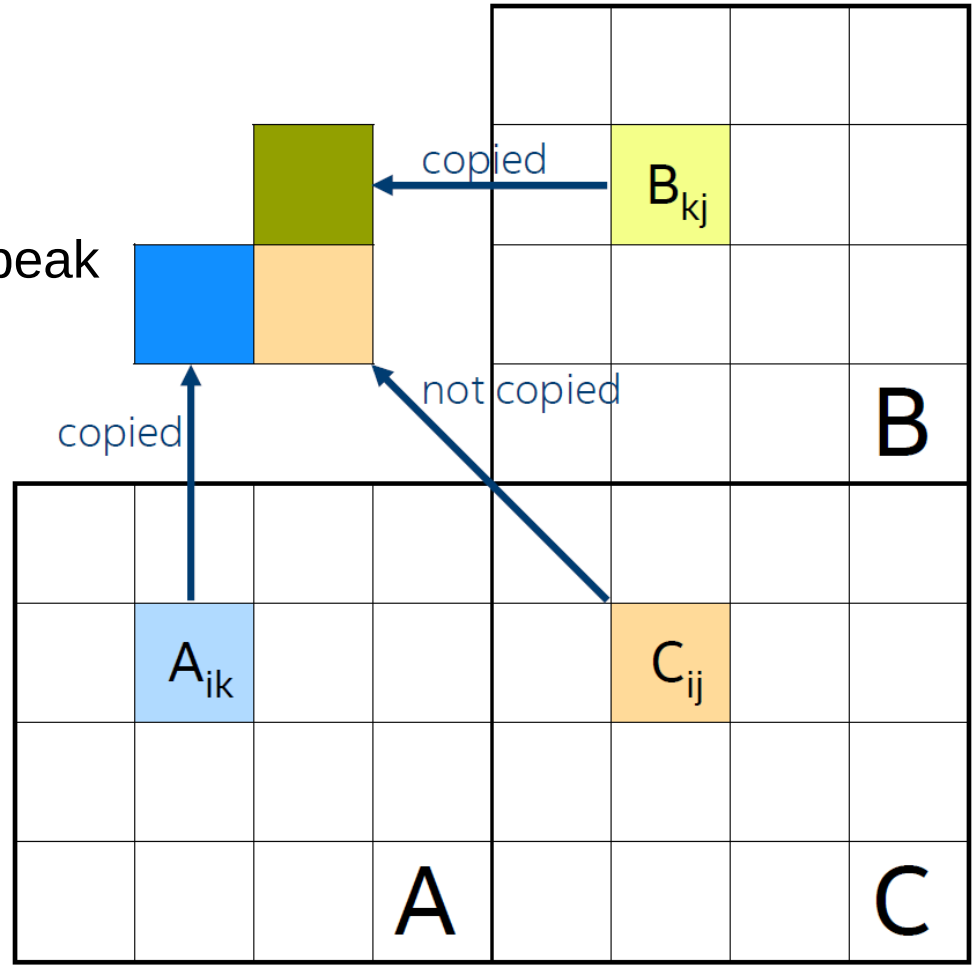
op(X) = X or XT

```
C = beta*C
DO i=1,M
    DO j=1,N
        DO kk=1,K
            C(i,j) +=
alpha*A(i,kk)*B(kk,j)
        END DO
    END DO
END DO
```



Matrix-Matrix Multiply Optimizations

- Highly tuned for Intel Architectures
  - SIMD instructions for high throughput
  - Parallel algorithms to employ multi-/many-cores
  - Tiling to maximize cache reuse
- Hand-tuned assembly kernels that run close to machine peak performance
- Input matrices, A and B, are copied into buffers
  - Contiguous data access pattern inside the kernels
  - Minimize cache and TLB misses
- Copy overheads are negligible for large matrices
  - Computation: O(N3), Copy: O(N2)
- Intel MKL skips copying if matrices are small
  - Requires good leading dimensions (avoid multiples of 256)
  - Non-transposed A and B are more suitable



Classification of MATRIX Sizes And Performance Challenges

- Small Sizes
  - M, N, K < 20
  - Challenges: High function call overheads, low vectorization, low parallelization
  - Solutions: Batch API, Compact API and MKL_DIRECT_CALL
- Medium Sizes
  - 20 < M, N, K < 500
  - Challenges: Low parallelization, high copy overheads
  - Solutions: Batch API and Pack API
- Skewed Sizes
  - M < 500 and large N
  - N < 500 and large M
  - Challenge: High copy overheads
  - Solution: Pack API
- Large Sizes
  - M, N, K > 5000
  - Performance close to machine's theoretical peak

Intel® MKL Solutions for SMALL, MEDIUM, and SKEWED Sizes

- MKL_DIRECT_CALL
  - Improves performance for small sizes (M, N, K < 20)
  - Skips error checking and function call overheads
  - Enabled for several LAPACK and BLAS functions: potrf, getrf, gemm, gemm3m, axpy, dot,…
- Compact APIs
  - Improves performance for small sizes (M, N, K < 20)
  - Enables vectorization over very small matrix dimensions by reformatting the data in a compact layout
  - Enabled for several LAPACK and BLAS functions: potrf, geqrf, gemm, trsm, …
- Batch APIs
  - Improves performance for small-medium sizes (M, N, K < 500)
  - Groups several independent function calls together
  - Enabled for gemm, gemm3m and trsm BLAS functions
- Packed APIs
  - Improves performance for small-medium M or N sizes (M or N < 500)
  - Allows amortizing copy overheads over several GEMM calls with same input matrix
  - Enabled for sgemm and dgemm BLAS function

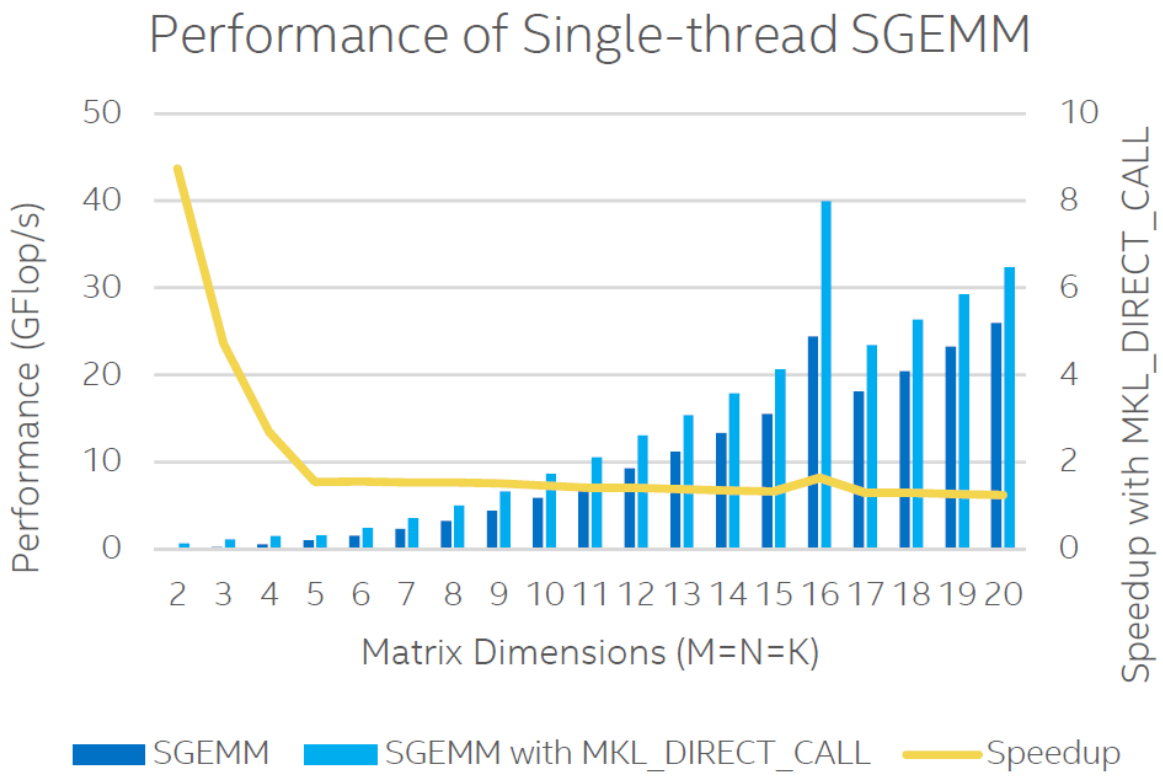
MKL_DIRECT_CALL Compilation FLAG

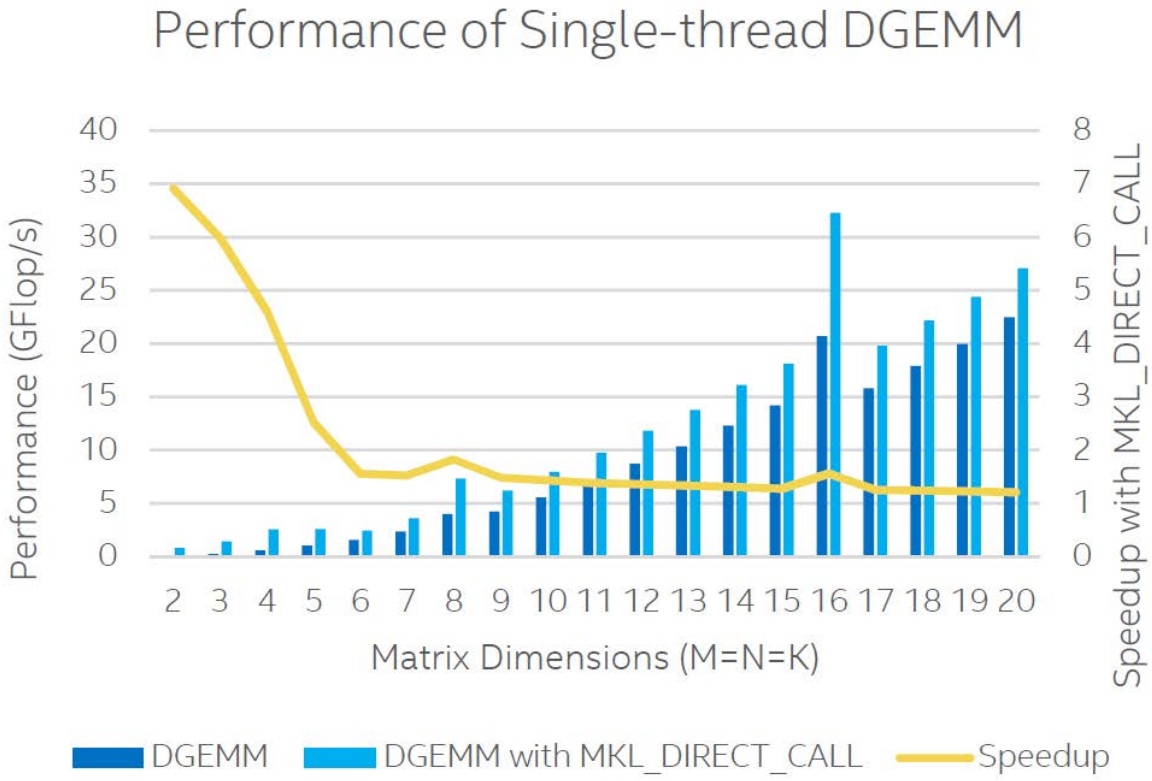
- Define the preprocessor macro MKL_DIRECT_CALL
  - If threading is not required, use MKL_DIRECT_CALL_SEQ
- Improves the performance of small sizes (M, N, K < 20)
  - Instead of calling a library function, a C implementation may be used
  - Starting from Intel MKL 2019.1, compiler intrinsics kernels for DGEMM Intel AVX2+
- Intel MKL can avoid some of the overheads
  - No error checking
  - No MKL_VERBOSE support
  - No CNR (Conditional Numerical Reproducibility) support
- Minimal modification is required, just add the preprocessor macro and the header file

```
// icc –DMKL_DIRECT_CALL –
std=c99 …
#include <mkl.h>
void main(void) {
        dgemm(…);
}
```

```
! ifort –DMKL_DIRECT_CALL –fpp …
#  include “mkl_direct_call.fi”
    program DGEMM_MAIN
    DGEMM(…)
```

MKL_DIRECT_CALL Performance on Intel® Xeon® Platinum Processor





Batch API

- Execute independent general matrix multiplication (GEMM) operations simultaneously with one function call
- Ensure no data dependency between the operations
- Take advantage of all cores even for small-medium sizes (M, N, K < 500)
- Minimize library overheads
- Some code modification is required to group same size matrices together

C1 = *alpha* . op(A1) . op(B1) + *beta* . C1
C2 = *alpha* . op(A2) . op(B2) + *beta* . C2
C3 = *alpha* . op(A3) . op(B3) + *beta* . C3

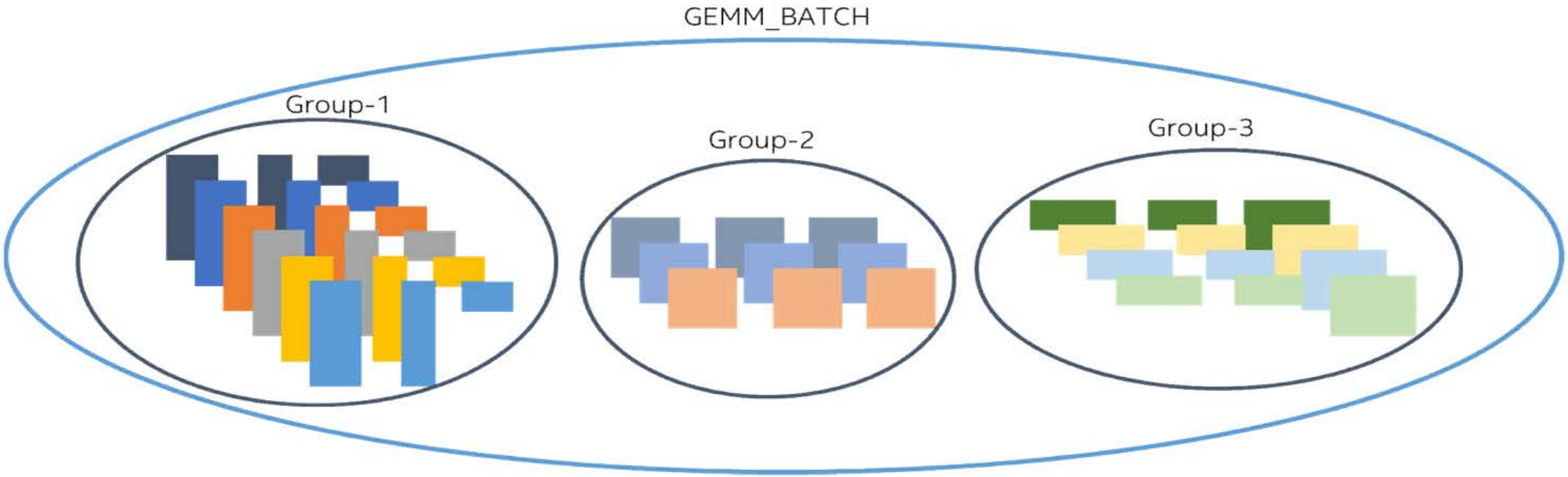
} Execute in parallel assuming no pointer aliasing

C2 = *alpha* . op(A4) . op(B4) + *beta* . C2

} Wait for a previous write to C_2

Group Concept in Batch API

- Group: a set of GEMM operations with same input parameters (matrix pointers can be different)
- Transpose, size, leading dimensions, alpha and beta
  - One GEMM_BATCH call can handle one or more groups



Batch API USAGE Example

- Group the GEMM calls with same parameters together
- Two groups of GEMM calls in the example below

```
#include <mkl.h>
int group_count = 2;

// Create arrays size of group_count to store GEMM arguments
CBLAS_TRANSPOSE transA[] = {CblasNoTrans, CblasNoTrans};
CBLAS_TRANSPOSE transB[] = {CblasTrans, CblasNoTrans};
MKL_INT m[] = {4, 3};
MKL_INT k[] = {4, 6};
MKL_INT n[] = {8, 3};
MKL_INT lda[] = {4, 6};
MKL_INT ldb[] = {4, 6};
MKL_INT ldc[] = {8, 3};
double alpha[] = {1.0, 1.0};
double beta[] = {0.0, 2.0};
MKL_INT size_per_grp[] = {20, 30};

// Call cblas_dgemm_batch to perform 50 GEMM operations
cblas_dgemm_batch(CblasRowMajor, transA, transB, m, n, k,
alpha, a_array, lda, b_array, ldb,
beta, c_array, ldc, group_count, size_per_group);
```

Packed API

- Amortize copy (pack) operation over multiple GEMM calls with the same
- input matrix
- Copy (pack) the data once and reuse it in many GEMM calls
- Improves the performance for medium or skewed sizes (M or N < 500) with input matrix reuse

```
C1 = alpha . op(A1) . op(B1) + beta . C1
C2 = alpha . op(A1) . op(B2) + beta . C2
C3 = alpha . op(A1) . op(B3) + beta . C3
```

Input matrix A_1 is shared between three GEMM calls

Important Performance Tips for Intel MKL

- Set KMP_AFFINITY to avoid thread migration
  - Intel Hyper-Threading Technology Enabled:
    - Linux*/macOS*: export KMP_AFFINITY=compact,1,0,granularity=fine
    - Windows*: set KMP_AFFINITY=compact,1,0,granularity=fine
  - Intel Hyper-Threading Technology Disabled:
    - Linux*/macOS*: export KMP_AFFINITY=compact
    - Windows*: set KMP_AFFINITY=compact
- Avoid leading dimensions that are multiples of 256
  - If ldim % 256 = 0, then add 16 to ldim
- Align memory along page boundaries
  - Use mkl_malloc and mkl_free for allocating and freeing aligned memory
- Maximize access to the local memory
- Let Intel MKL use high-bandwidth memory for internal buffers
  - Install memkind library
  - Numactl
- For more details see “Developer Guide for Intel MKL: Managing performance and memory”

Intel MKL Resources

- Intel MKL Developer Reference: https://software.intel.com/en-us/articles/mkl-reference-manual
- Intel MKL Developer Guide: https://software.intel.com/en-us/mkl-linux-developer-guide
- Intel® Math Kernel Library Link Line Advisor: https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
- Intel MKL Forum: https://software.intel.com/en-us/forums/intel-math-kernel-library/
- No cost option for Intel MKL: https://software.intel.com/en-us/articles/free-ipsxe-tools-and-libraries
- Intel MKL-DNN: https://github.com/01org/mkl-dnn

thanks.

Different is better