MAE 5032 High Performance Computing: Methods and Applications

Lecture 6: LSF Job managing system

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

TaiYi

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

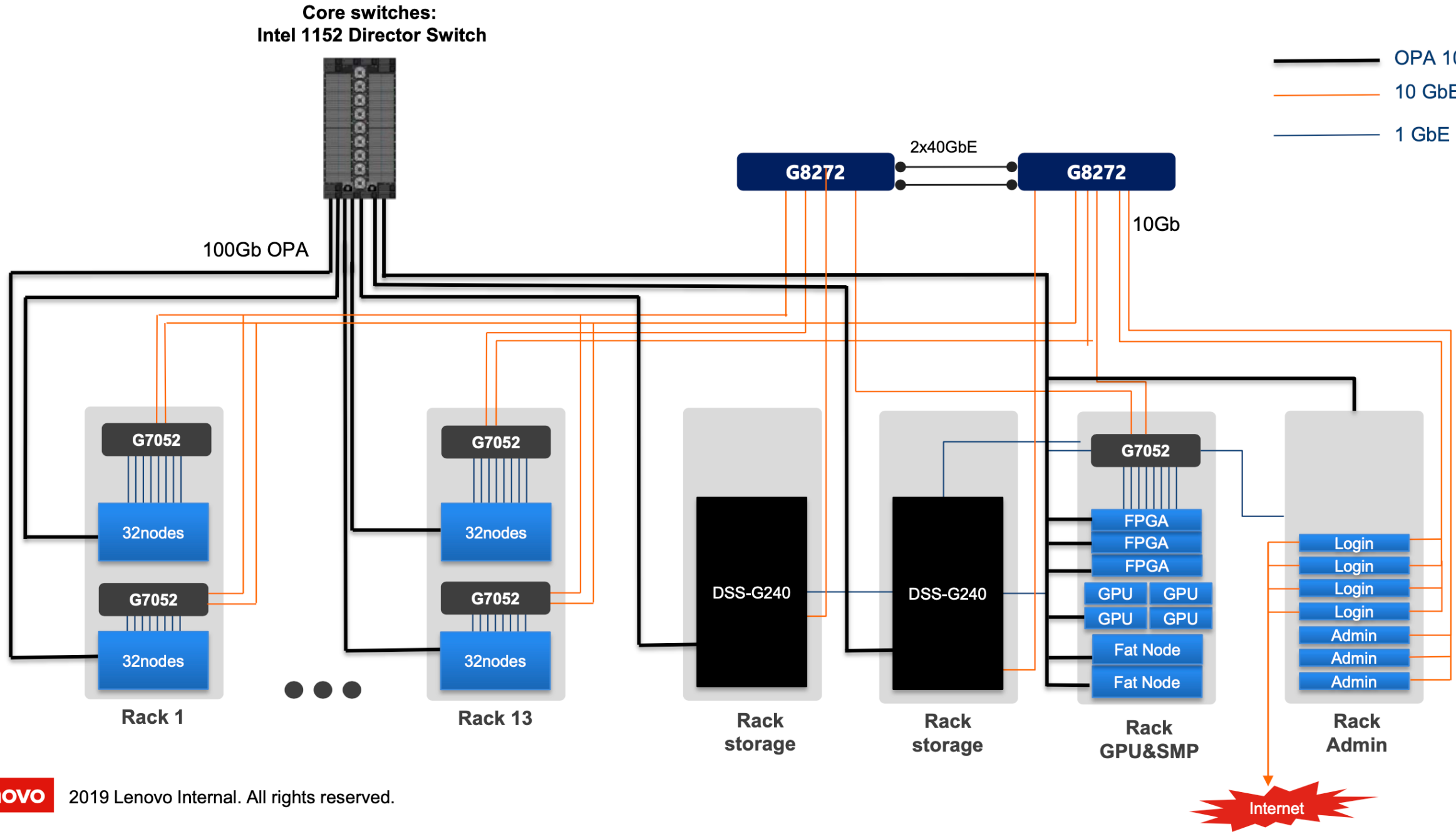
Node types on TaiYi

3 types of compute blocks:

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

14nm lithography, 20 cores, 40 threads, 2.4GHz, 27.5 MB L3 Cache
support SSE, AVX, AVX2, AVX-512 instruction sets
$ 3072

https://ark.intel.com/content/www/us/en/ark/products/120489/intel-xeon-gold-6148-processor-27-5m-cache-2-40-ghz.html



System access

- Each SUSTech member may have an account, which is listed under a research allocation or a teaching allocation.
- Currently, 7 students are listed under our course allocation, the rest are listed under their research group’s allocation.
- One can connect to TaiYi through its login nodes.
- ssh to compute nodes is denied, unless your job gets the permission to run.

System access

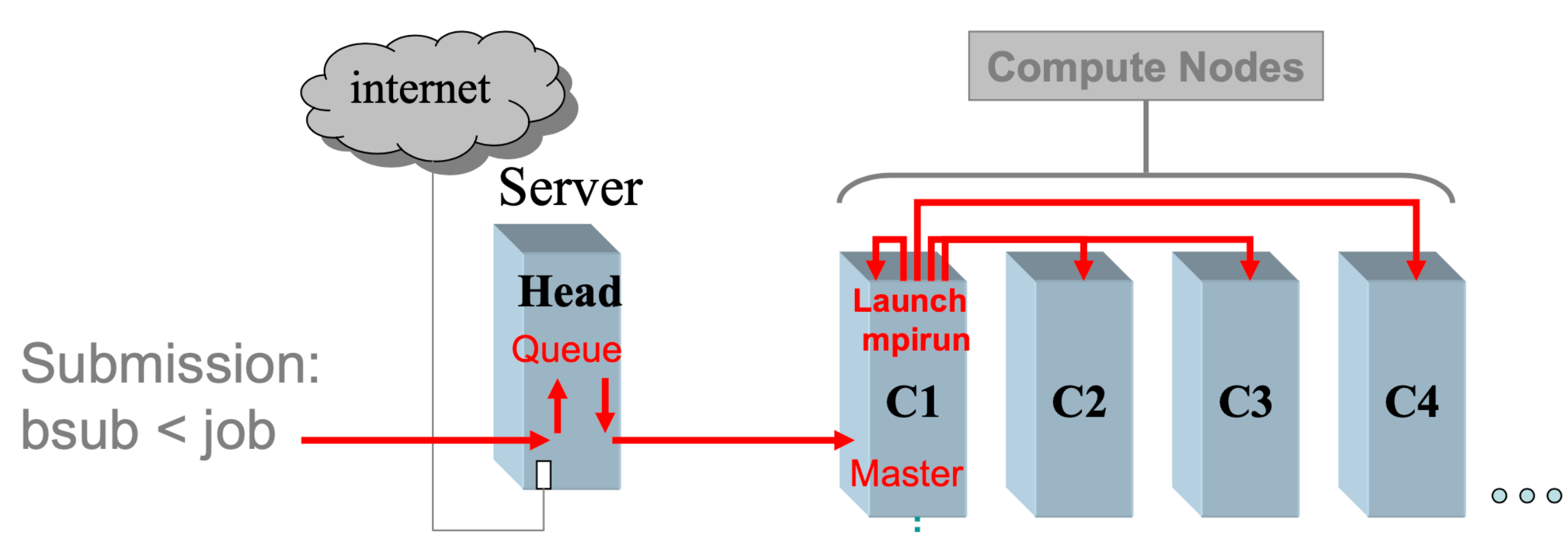
- SSH only
- UNIX users

  ssh username@login-node-address
- Windows users: Get a client
  - PuTTY
    - http://www.chiark.greenend.org.uk/~sgtatham/putty/
  - Cygwin and follow the UNIX instructions
    - htts://www.cygwin.com

Hostname	IP Addr
Login01	172.18.6.175
Login02	172.18.6.176
Login03	172.18.6.177
Login04	172.18.6.178

System access

- File transfer by scp
- If you have massive data, bring your hard drive directly to the computing center
- Each login node have 10 Gb uplink to campus network
- Jobs run in a managed environment
  - login to the login node
  - submit jobs to the scheduler
  - wait
  - ccollect results
- Running programs on the login node is highly discouraged!
  - avoid resource intensive tasks
  - exceptions include compilers, “standard” UNIX commans (ls, mkdir, cp, mv, etc.)



Queue: Job Script waits for resources on Server

Master: Compute Node that executes the job script, launches ALL MPI processes

Launch: ssh to each compute node to start executable (e.g. a.out)

ibrun ./a.out

mpirun –np # ./a.out

System access

- If you do NOT know what you are doing, WAIT!
- If you are already running jobs on TaiYi, feel free to use your account.
- Do NOT use the class account for your research.
- Do NOT run jobs on login node!
- Do NOT do frequent I/O on the system.

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 size is about 200GB.
  - General Platform File System (GPFS):
    - /share: applications and libraries that have already been installed
    - /work: the home directories of all users. 1TB
    - /scratch: store temporary files of your jobs during their execution. There is no default quota on this filesystem. Data will be cleaned regularly
    - /data: only on login nodes. This space is intended to store user important data. 8TB.

- It is your own responsibility to backup your data.

- Use mmlsquota to check your storage usage.

Filesystem

Use mmlsquota to show quota information for a user, group, etc.

/work 1TB

/data 8TB

```
-> mmlsquota -g mae-liuj --block-size auto

Disk quotas for group mae-liuj (gid 2102):
                         Block Limits                                    |     File Limits
Filesystem type         blocks      quota      limit   in_doubt    grace |    files   quota    limit in_doubt    grace  Remarks
data       GRP          5.162T         8T     8.008T     1.493G     none |   465108       0        0      102     none

                         Block Limits                                          |     File Limits
Filesystem Fileset    type         blocks      quota      limit   in_doubt    grace |    files   quota    limit in_doubt    grace  Remark
scratch    root       GRP          5.125M       100T     100.1T          0     none |        0       0        0        0     none

                         Block Limits                                    |     File Limits
Filesystem type         blocks      quota      limit   in_doubt    grace |    files   quota    limit in_doubt    grace  Remarks
work       GRP          41.42G         1T     1.025T          0     none |   469596       0        0        0     none
```

Batch systems

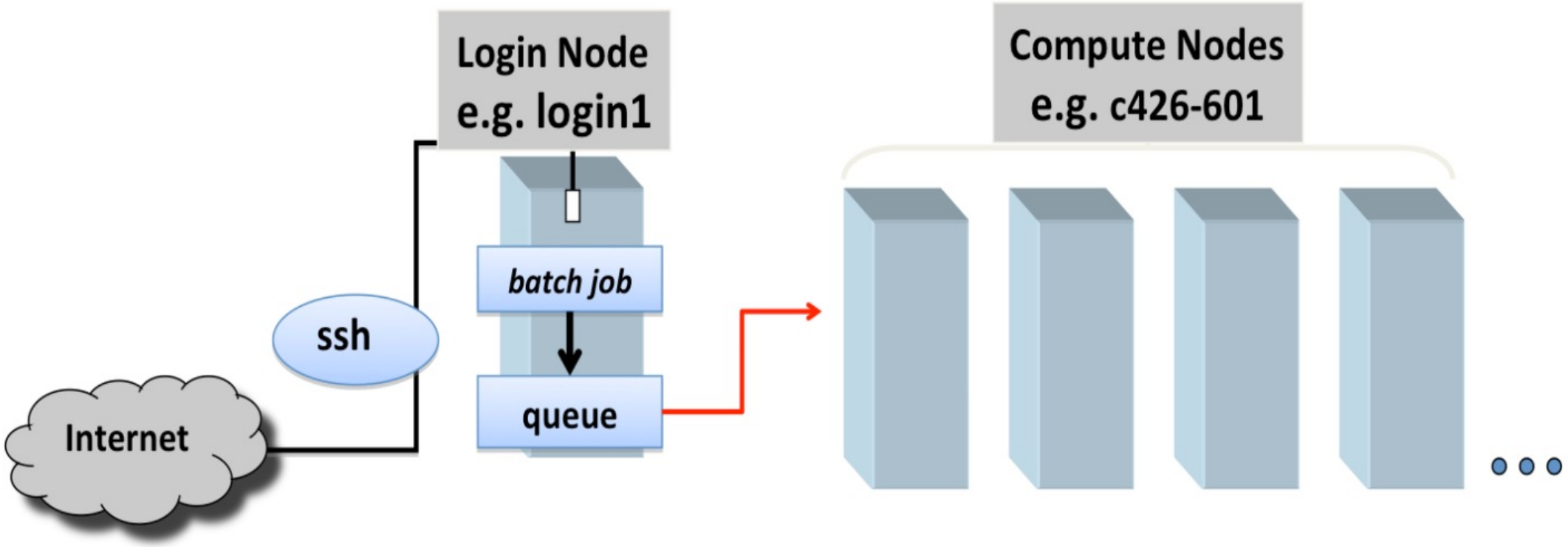
- In a number of scientific computing environments, multiple users must share a compute resource:
  - research clusters
  - supercomputing centers

- On multi-user HPC clusters, the **batch system** is a key component for aggregating compute nodes into a single sharable computing resource.

- The batch system becomes the nerve center for coordinating the use of resources and controlling the state of the system in a way that must be fair to its users.

- As current and future expert users of large-scale compute resources, you need to be familiar with the basics of a batch system.

Batch systems



Batch systems

- The core functionality of all batch system are essentially the same, regardless of the size or specific configuration of the compute hardware:
  - Multiple job queues
    - queues provide an orderly environment for managing a large number of jobs
    - queues are defined with a variety of limits for maximum run times, preprocessor counts, they are often assigned different priority levels as well.
    - may be interactive or non-interactive.
  - Job control
    - submission of individual jobs to do some work
    - simple monitoring and manipulation of individual jobs and collection of resource usage statistics (CPU usage, elapsed wallclock time per job, etc.)
  - Job scheduling
    - policy which decides priority between individual user jobs
    - allocate resources to scheduled jobs

Batch systems

- Common batch systems you may encounter in scientific computing
  - LSF
  - PBS
  - SLURM
  - etc.

- All have similar functionality but different syntax

- Reasonably straightforward to convert your job scripts from one system to another

- Above all include specific batch system directives which can be placed in a shell script to request certain resources (processors, queues, etc.)

Batch systems

- LSF is the utility used for batch processing.

- There are several queues present in the machines and different users may access different queues, all queues have different limits in number of cores for the jobs and duration.

Queue Name	Maximum # of nodes	Maximum wall clock
medium	805	72 hours
short	805	24 hours
smp	2	12 hours
ser	10	24 hours
gpu	4	24 hours
debug	10	1 hours

Batch systems

bhosts	Displays configured compute nodes and their static and dynamic resources (including job slot limits)
lsload	Displays dynamic load information for compute nodes (avg CPU usage, memory usage, available /tmp space)
bsub	submits a batch job to LSF
bqueues	displays information about available queues
bjobs	displays information about running and queued jobs
bhist	displays historical information about jobs
bstop	suspends unfinished jobs
bresume	resumes one or more suspended jobs
bkill	Sends signal to kill, suspend, or resume unfinished jobs
bhpart	Displays global fairshare priority
lshosts	Displays hosts and their static resource configuration
lsuser	Shows user job information

Batch systems

```
-> bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS   PEND    RUN  SUSP
bench            90  Closed:Active     -    -    -    -     0      0      0     0
medium_tmp       90  Closed:Active     -    -    -    -     0      0      0     0
short_tmp        90  Closed:Active     -    -    -    -     0      0      0     0
test             90  Open:Active       -    -    -    -   160    160      0     0
test-fj          90  Open:Active       -    -    -    -     0      0      0     0
large_tmp        90  Closed:Active     -    -    -    -     0      0      0     0
admin            50  Closed:Active     -    -    -    -     0      0      0     0
short            40  Open:Active       -    -    -    - 13360     40  13320     0
spec             40  Open:Active       -    -    -    -     0      0      0     0
debug            40  Open:Active       -  160    -    -   144      0    144     0
medium           30  Open:Active       -    -    -    - 11240    200  11040     0
ser              30  Open:Active       -    -    -    -  1089     80   1009     0
ser_tmp          30  Closed:Active     -    -    -    -     0      0      0     0
gpu              30  Open:Active       -    -    -    -    27      0     27     0
smp              30  Open:Active       -    -    -    -   418    228    190     0
smp_tmp          30  Closed:Active     -    -    -    -     0      0      0     0
large            30  Open:Active       -    -    -    -  7600   3560   4040     0
```

Batch systems

队列	队列时长限制	应用场景	单作业核心数限制
large	36小时	大规模	600核以上（含600）
medium	72小时	中等规模	200核以上（含200）
short	144小时	小规模	40核以上（含40）
ser	168小时	串行	无限制
smp	168小时	大内存	无限制
debug	20分钟	调试	80核以上（含80）
gpu	168小时	GPU	无限制

use `bqueues -l short` to see the queue details

Batch systems

`lsload` : shows the node status

```
HOST_NAME       status   r15s    r1m   r15m    ut     pg  ls     it    tmp    swp    mem
login05             ok    0.0    0.0    0.8    0%    0.0  16      0   424G   3.8G   363G
r07n48              ok    0.0    0.0    1.0    0%    0.0   0  81664   212G   3.1G   161G
r09n18              ok    0.0    0.0    0.0    0%    0.0   0  81664   212G   2.5G   162G
r01n39              ok    0.0    0.0    0.0    0%    0.1   0  81664   212G     2G   162G
r05n15              ok    0.0    0.0    2.5    0%    0.0   0  81664   212G   2.8G   162G
r12n64              ok    0.0    0.0    0.0    0%    0.0   0   3716   212G   3.9G   169G
```

status:

ok: can accept new jobs

```
-> lsload r01n01
HOST_NAME       status   r15s    r1m   r15m    ut     pg  ls     it    tmp    swp    mem
r01n01              ok   17.8   23.3   25.7   39%    0.0   0  17424   202G   3.5G   155G
```

r15s r1m r15m : the load over the past 15 seconds, 1 minute, and 15 minutes, respectively.

tmp : the size of the tmp folder

swp/mem : the swap and main memory size

Batch systems

use `bhosts hg_ser` to see the node belonging to ser queue.

```
-> bhosts hg_ser
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
r01n01             ok              -     40     25     25      0      0      0
r01n02             ok              -     40     31     31      0      0      0
r01n03             ok              -     40      5      5      0      0      0
r01n04             closed          -     40     40     40      0      0      0
r01n05             closed          -     40     40     40      0      0      0
r01n06             ok              -     40     36     36      0      0      0
r01n07             ok              -     40     30     30      0      0      0
r01n08             closed          -     40     40     40      0      0      0
r01n09             ok              -     40     38     38      0      0      0
```

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.

Job scripts

- The user is required to provide a job script with the specification of
  - the resources requested and the job constraints
  - specific queue specification
  - all what is needed for a working execution-environment

- By resources we mean the number of processors/cores/nodes, the amount of memory needed and some specific features, and so on.

- All these information must be written in a text file.

Job script

- 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<jobscript

Job script

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]

Job script

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.

https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=bsub-options

Job script

-n Cores

the number of cores needed for the job; the number of MPI tasks

-W Wall Clock time

Set the the run time limit of the job.

-w “done(job-id)”

set the job dependency

-q Queue

Specify the queue for the job. If not specified, a default queue will be selected.

-m Hostname

Select a particular host or host group to run on. This may be useful for maintaining consistency of the jobs in, e.g., scaling tests.

```
#!/bin/bash
#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -w "done(12345)"
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[ptile=1]"
#BSUB -m 'r13n45'
```

Job control commands

- There may be situations where there are obvious dependencies between jobs and one cannot start before another has finished.

- LSF provides a dependency feature that can be used for this purpose. One can therefore specify in the batch script of a job that it depends on another one:

  #BSUB –w “done(<jobid>)”

- This will ensure that the current job is started only after the job with the jobid is completed successfully. Some options are:
  - done : job state done
  - ended : job state exit or done
  - started : job state DONE, EXIT, USUSP, SSUSP, or RUN

Job script

-R span[ptile=X]

Specify the number of cores to place on each node

-o output file

Name the file containing the standard output from the run.

-e Error file

Name the file containing the standard error message fro mthe run.

-J jobname

Specify a name to the job.

```
#!/bin/bash
#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -w "done(12345)"
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[ptile=1]"
#BSUB -m 'r13n45'
```

Job control commands

Use `bjobs` to view the status of submitted jobs

```
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
3395677    mae-liu PEND  ser        login03                 *n-20c-ser Mar 15 23:02
```

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
- **RUN** the job has been granted an allocation.
- **DONE** the job has completed successfully

Job 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

Batch job translation

The syntax can be translated to other system:

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

Batch job translation

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]

Batch job translation

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 6
#SBATCH --sockets-per-node 2
#SBATCH --cores-per-socket 3
#SBATCH -p paratera

yhrun ../fsi_tet4_3d -nz_estimate 80
```

Software environment

Software environment

- Libraries installed on the cluster can be found at /share
  - /share/apps: applications such as vasp, openmpi, etc.
  - /share/base: basic libraries, gcc, etc.
  - /share/intel: Intel tools

- Use module avail to see the available libraries
  - $module avail intel
  - $module avail gcc

Software environment

- You may find the following compilers
  - icc/icpc: Intel C/C++ compilers
  - gcc/g++: GNU Compilers
  - ifort: Intel Fortran compiler
  - gfortran: GNU Fortran compiler

- To compile MPI progams, use **mpicc** and **mpicxx** for C and C++ source code, **mpif77** or **mpif90** for Fortran source code. These are wrappers that include all necessary libraries to build MPI applications.

- For Intel MPI, use **mpiicc/mpiicpc,mpiifort** wrappers, which include intel mpi and intel compilers.

Optimization flags

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)

Use –xSKYLAKE-AVX512 for AVX512 instruction set

use –mcmodel=medium –shared-intel if you get a link time error relating to R_X86_64_PC32

Optimization flags

Some optimization may affect how floating point arithmetic is performed. The following 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

- The environment module package provides a way for dynamic modification of user's environment via module files. Each module file contains the information to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically in a clean fashion.
  - **module avail** [packagename] shows the packages available under the module system
  - **module load** packagename/version use some software on the cluster
  - **module list** show the packages loaded on the current session
  - **module unload** packagename/version remove one module from your current session
  - **module purge** remove all modules from your current session

Tutorial

- We have a simple code for vector manipulations.

```
#define ARRAY_SIZE 1024
#define NUMBER_OF_TRIALS 1000000
```

```
int main(int argc, char *argv[])
{
  int i,t;

  double m = 1.0001;

  /* Populate A and B arrays */
  for (i=0; i < ARRAY_SIZE; i++)
  {
    b[i] = i;
    a[i] = i+1;
  }

  /* Perform an operation a number of times */
  for (t=0; t < NUMBER_OF_TRIALS; t++)
  {
    for (i=0; i < ARRAY_SIZE; i++)
    {
      c += m*a[i] + b[i];
    }
  }

  return 0;
}
```

Tutorial

```
-> module load intel/2018.4
mae-liuj::login02 { ~/mae-5032/06-opt-flag }
-> icc simple.c -O3 -qopt-report=2 -qopt-report-phase=vec -o simple2
icc: remark #10397: optimization reports are generated in *.optrpt files
mae-liuj::login02 { ~/mae-5032/06-opt-flag }
-> cat simple.optrpt
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.


Begin optimization report for: main(int, char **)

    Report from: Vector optimizations [vec]
```

Tutorial

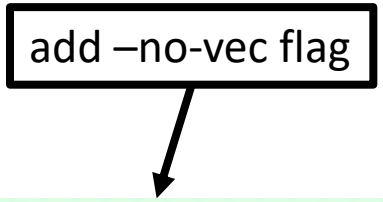
```
LOOP BEGIN at simple.c(20,3)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at simple.c(27,3)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at simple.c(29,5)
      remark #15300: LOOP WAS VECTORIZED
   LOOP END
LOOP END
```

We can see that two loops were vectorized.

Tutorial




```
-> icc simple.c -O3 -no-vec -qopt-report=2 -qopt-report-phase=vec -o simple2_no_vec
icc: remark #10397: optimization reports are generated in *.optrpt files in the output locatio
mae-liuj::login02 { ~/mae-5032/06-opt-flag }
-> cat simple.optrpt
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.
```

Open simple.optrpt and we can see that the loop is not vectorized.

Tutorial

As mentioned earlier, the Intel compiler use SSE (128-bit) instruction by default. That means the compiled codes will run on general architectures, but can be further optimized.

We add –xSKYLAKE-AVX512 now.

```
-> icc simple.c -O3 -xSKYLAKE-AVX512 -qopt-report=2 -qopt-report-phase=vec -o simple2_avx512
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
```

If you open the vectorization report, you will see that it include the remark “compiler has chose to target XMM/YMM vector” along with a suggestion to try the option –qopt-zmm-usage=high.

Tutorial

Let us try to get rid of the odd remark. It comes because Intel compiler will not necessarily use the 512-bit ZMM register by default when code is auto-vectorized for Skylake.

The reason is that the Skylake is a general purpose processor that may encounter a wide variety of workloads.

We need –qopt-zmm-usage=high to tell the compiler to use 512-bit vector instructions as much as possible.

```
-> icc simple.c -O3 -xSKYLAKE-AVX512 -qopt-zmm-usage=high -qopt-report=2 -qopt-report-phase=vec -o s
imple2_zmm_hi
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
```

If you open the vectorization report, the strange remarks are gone.

Tutorial

Generate a job script and run on compute node.

```
#!/bin/bash

#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -e log
#BSUB -o log
#BSUB -R "span[ptile=1]"
#BSUB -m 'r13n45'

# Non-vectorized code
/usr/bin/time -f "simple2_no_vec: %e" ./simple2_no_vec

# Standard SSE vectorized code
/usr/bin/time -f "simple2: %e" ./simple2

# AVX-512 vectorized code
/usr/bin/time -f "simple2_avx512: %e" ./simple2_avx512

# AVX-512 vectorized code, with tweaks
/usr/bin/time -f "simple2_zmm_hi: %e" ./simple2_zmm_hi
```

Tutorial

From no-vec to –O3, the gain is 2.7X.
SIMD contributes to most of the speedup.
Do not forget the FMA and pipelinging that may also affect the efficiency.

```
The output (if any) follows:

simple2_no_vec: 0.67
simple2: 0.25
simple2_avx512: 0.12
simple2_zmm_hi: 0.08
```

From –O3 to –avx512, the gain is about 2X.

When –qopt-zmm-usage=high is turned on, the gain is more than 3X

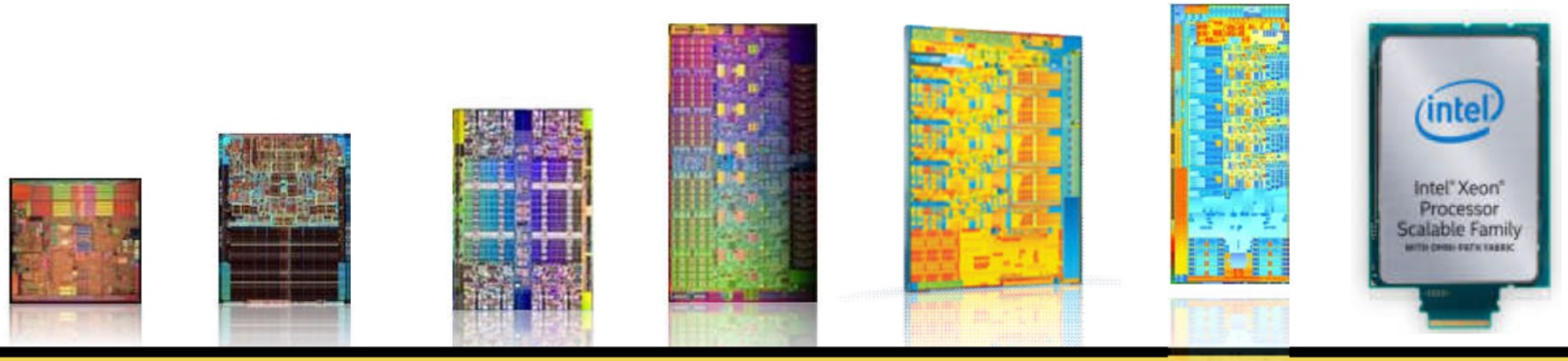
Intep provide profiler that helps to get better understanding of the speedup.

Intel MKL

Intel MKL

- Speeds computation for scientific and engineering applications.
- Provides key functionality for dense and sparse algebraic operations.
- Linear algebra (BLAS, LAPACK,PARDISO), FFT, vector math, statistics, deep learning primitives, and more.
- Provides scientific programmers and domain scientists
  - interface to de-facto standard API from C++, C, Fortran, ect.
  - Support UNIX, Windows, and OS X systems
  - Extract great performance with minimal effort
- Optimized for single core vectorization and cache utilization
- Automatic parallelism for multicore and manycore
- 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

Intel MKL usage

- MKL is part of the intel compiler. Once you load a compiler module, the environment variable MKLROOT is automatically set and the path to the MKL library is automatically included in your default path.

  echo $MKLROOT

- Go to the online Intel MKL Link Line Advisor to determine the libraries and options to specify on your link or compilation line
  - https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-link-line-advisor.html#gs.syut5b

Intel MKL usage

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.

Intel MKL usage: matrix-matrix multiplication

- Part of the Intel MKL BLAS
- *GEMM(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
- C = alpha op(A) * op(C) + beta C
- op(A) = A or A transpose

```
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
```

Intel MKL usage: matrix-matrix multiplication

- Part of the Intel MKL BLAS
- *GEMM(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
- C = alpha op(A) * op(C) + beta C
- op(A) = A or A transpose

```
gettimeofday(&stime, NULL);
double time_st  = dsecnd();
for (i=0; i<LOOP_COUNT; ++i)
{
  dgemm(&transa, &transb, &m, &n, &k, &alpha, A, &m, B, &k, &beta, C, &m);
}
double time_end = dsecnd() ;
```

Intel MKL usage: matrix-matrix multiplication

- Highly tuned for intel architectures
  - SIMD instructions for high throughput
  - Tiling to maximize cache reuse
- Input matrices are moved with cache miss minimized
- For large matrices copy overheads are negligible
  - computation O(N*N*N) copy O(N*N)
- For large sizes (M,N,K>5000) performance close to machine’s theoretical peak

Tutorial

- Compile the code by icc

```
icc main.c -lmkl_intel_lp64 -lmkl_intel_thread
-lmkl_core -liomp5 -lpthread -lm -ldl
```

Tutorial

- Prepare a job script and submit it to the system

```
#!/bin/bash
#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[ptile=1]"
#BSUB -m 'r13n45'

module purge
module load intel/2018.4
module load mpi/intel/2018.4

mpirun /work/mae-liuj/mae-5032/06-lsf/dgemm > $LSB_JOBID.log 2>&1
```

References

- Running jobs with IBM Spectrum LSF
- TaiYi supercomputer user manual