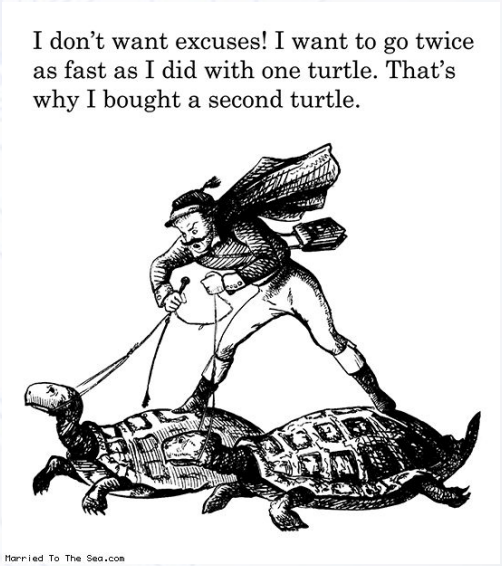
UiO : USIT

Using the HPC resources effectively

Using HPC resources effectively - Performance analysis

Ole W. Saastad, UiO/USIT/ITF/FI
March 2022

- Running programs «one of the major uses of computers is to run computer programs»
- Resources
- Environment
- Scaling



- Running programs
  - Interactive run
  - Reserve resource for interactive use
  - Set up for batch execution
- Resources
- Environment
- Scaling

Interactive usage on front end:

- short tiny jobs running only a few minutes for testing
- short compiling and testing
- copying files small about of data

Anything longer, log in to an interactive node

Batch queue system SLURM

SLURM, https://slurm.schedmd.com/

All batch job submitted using scripts, bash, python, Julia, R etc, any scripting language that uses # as comment character.

https://www.uio.no/english/services/it/research/platforms/edu-research/help/hpc/docs/fox/jobs/index.md

```
#!/bin/bash

#SBATCH --job-name=bash
#SBATCH --account=ec01
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:02:0
#SBATCH --mem-per-cpu=200M

echo "Hello world"
```

```
#!/usr/bin/env python

#SBATCH --job-name=bash
#SBATCH --account=ec01
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:02:0
#SBATCH --mem-per-cpu=200M

print("Hello world")
```

```
#!/usr/bin/env julia

#SBATCH --job-name=bash
#SBATCH --account=ec01
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:02:0
#SBATCH --mem-per-cpu=200M

println("Hello world")
```

R is fine, Lua is not. Need # as comment.

Launching your MPI executable

Serial : ./a.out

OpenMP : ./a.out

MPI: mpirun ./a.out

srun ./a.out

mpirun vs srun: it can affect startup time, after launch the performance is normally the same. Generally mpirun uses SLURM to launch and not its internal ssh launcher.

- Running programs
- Resources
  - Request resources
  - SLURM usage
  - Storage scratch and persistent
- Environment
- Scaling

Number of nodes and cores

#SBATCH --nodes=1 : Control number of nodes

#SBATCH --ntasks-per-node=1 : Control number of MPI ranks per node

#SBATCH --cpus-per-task=1 : Control how many threads per task/rank

Number of nodes and cores

#SBATCH --nodes=2 --ntasks-per-node=2 --cpus-per-task=2

Slurm sets variable for us:

$SLURM_NODELIST=c3-25,c5-12

$SLURM_NNODES=2

$SLURM_NTASKS=4

$SLURM_NPROCS=4

$SLURM_NTASKS_PER_NODE=2

$SLURM_CPUS_ON_NODE=4

$SLURM_CPUS_PER_TASK=2

$OMP_NUM_THREADS=2

Number of nodes and cores

#SBATCH --nodes=2 --ntasks-per-node=2 --cpus-per-task=2

Slurm also sets MPI variable for MPI applications:

$PMI_SIZE=4

$PMI_RANK=1 (0,1,2,3)

$MPI_LOCALRANKS=2

$MPI_LOCALRANKID=0 (0,1)

In addition a lot of variables letting MPI set up communicator space (comm_world) etc.

Pure MPI jobs is fairly simple :

#SBATCH --nodes=8 : Control number of nodes

#SBATCH --ntasks-per-node=128 : Control number of MPI ranks per node

Number of ranks equal: nodes x ntasks-per-node

Best Practice is to use n-tasks-per-node equal to the total number of cores in each node (Saga 40, Fram 32, Betzy 128) for jobs larger than a single node.

Single node shared memory OpenMP / threaded jobs

#SBATCH --nodes=1 : Control number of nodes, ONLY one for OpenMP

#SBATCH --cpus-per-task=8 : Control number of threads (check scaling!)

SLURM_CPUS_PER_TASK=8

SLURM_JOB_CPUS_PER_NODE=8

SLURM_CPUS_ON_NODE=8

OMP_NUM_THREADS=8

Launching hello.x MPI executable/OpenMP hybrid executable

```
mpirun ./a.out
```

or

```
srun ./a.out
```

Nothing else is needed for a simple run, -np or -ppn is taken care of by SLURM.

This is a hybrid job, MPI + OpenMP

nodes=2

ntasks-per-node=2

cpus-per-task=2

4 MPI ranks, 2 pr node

(nodes x ntasks-per-node)

2 threads per rank

```
Hello world from rank 0 out of 4 ranks on host c1-6
  Hello world from thread 0 out of 2 threads on rank 0
  Hello world from thread 1 out of 2 threads on rank 0
Hello world from rank 1 out of 4 ranks on host c1-6
  Hello world from thread 0 out of 2 threads on rank 1
  Hello world from thread 1 out of 2 threads on rank 1
Hello world from rank 2 out of 4 ranks on host c1-7
  Hello world from thread 0 out of 2 threads on rank 2
  Hello world from thread 1 out of 2 threads on rank 2
Hello world from rank 3 out of 4 ranks on host c1-7
  Hello world from thread 0 out of 2 threads on rank 3
  Hello world from thread 1 out of 2 threads on rank 3
```

Hybrid jobs are complicated

Scaling of MPI - now many ranks are optimal ?

Scaling of OpenMP - now many threads are optimal ?

What is the optimal MPI rank number and how well does the OpenMP part scale?

Remember each rank is a separate stand-alone executable just like any other threaded program.

Requesting memory,
https://documentation.sigma2.no/jobs/choosing_memory_settings.html

The simplest way, check your slurm output file :

```
Memory usage stats:
     JobID     MaxRSS MaxRSSTask   AveRSS MaxPages   MaxPagesTask   AvePages
------------ ---------- ---------- ---------- -------- -------------- ----------
7643
7643.batch     248M      0         248M      0              0         0
7643.extern      0       0         0         0              0         0
```

Maximum resident (RSS) memory was 248 MiB (actual memory needed).

You need to allocate at least this amount, some more to be safe (read the docs).

Storage during execution

Are all your input data available ?

How do you store the output data ?

What about scratch data during a run ?

Have you considered the access of data during a run ?

Sequential or random access ? At what size ?

Sharing among several nodes ?

Storage during execution

Are all your input data available ?

Make sure all data is present when the large parallel job start. Don’t let 1023 cores run idle just because a single core is copying og downloading input data.

Storage during execution

How do you store the output data ?

Writing log data or output data to the slurm log file is not always necessary.

Redirecting such unused data to /dev/null is a neat idea.

Storage during execution

What about scratch data during a run ?

Using localscratch on some systems (Saga) can have huge benefits. Size is limited to about 800 GB.

Storage during execution

Have you considered the access of data during a run ?

Sequential or random access ? At what size ?

Everything is easy with sequential access, even tape can handle that. If you have random access you really need to take care. If small blocks/chunks/records (less than ½ MB) even more so.

Storage during execution

Sharing among several nodes ?

Do you need to share files on a common directory during a run, if so $SCRATCH is the only option.

Storage for jobs

https://documentation.sigma2.no/files_storage/clusters.html

SCRATCH : /cluster/work/jobs//cluster/work/jobs/<SLURM JOB ID>

LOCALSCRATCH : /localscratch/<SLURM JOB ID>

The local storage has different characteristics than the global file system.

(Localscratch is only available on Saga and is NVMe based and limited in size ~800G).

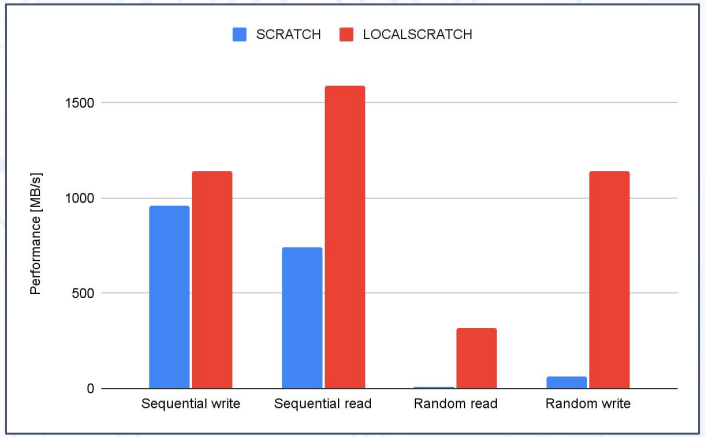
Job storage performance

	Sequential Write	Sequential Read	Random Read	Sequential Write
SCRATCH	963 MB/s	755 MB/s	6.41 MB/s	63.8 MB/s
LOCALSCRATCH	1144 MB/s	1589 MB/s	319 MB/s	1145 MB/s

Most users select $SCRATCH
from Jan 1st 2021 jobs :

$SCRATCH :	2538774	99.2%
$LOCALSCRATCH :	20145	0.8%

From Saga where localscratch is NVMe.



This iozone job:
SCRATCH 20 h
LOCALSCRATCH 47m

- Running programs
- Resources
- Environment
  - Mapping and binding
  - Controlling run time system (OpenMP/MPI)
- Scaling

Running threaded programs

```
#SBATCH --nodes=1 --tasks-per-node=1 --cpus-per-task=64
echo $OMP_NUM_THREADS
./a.out
```

Environment variables

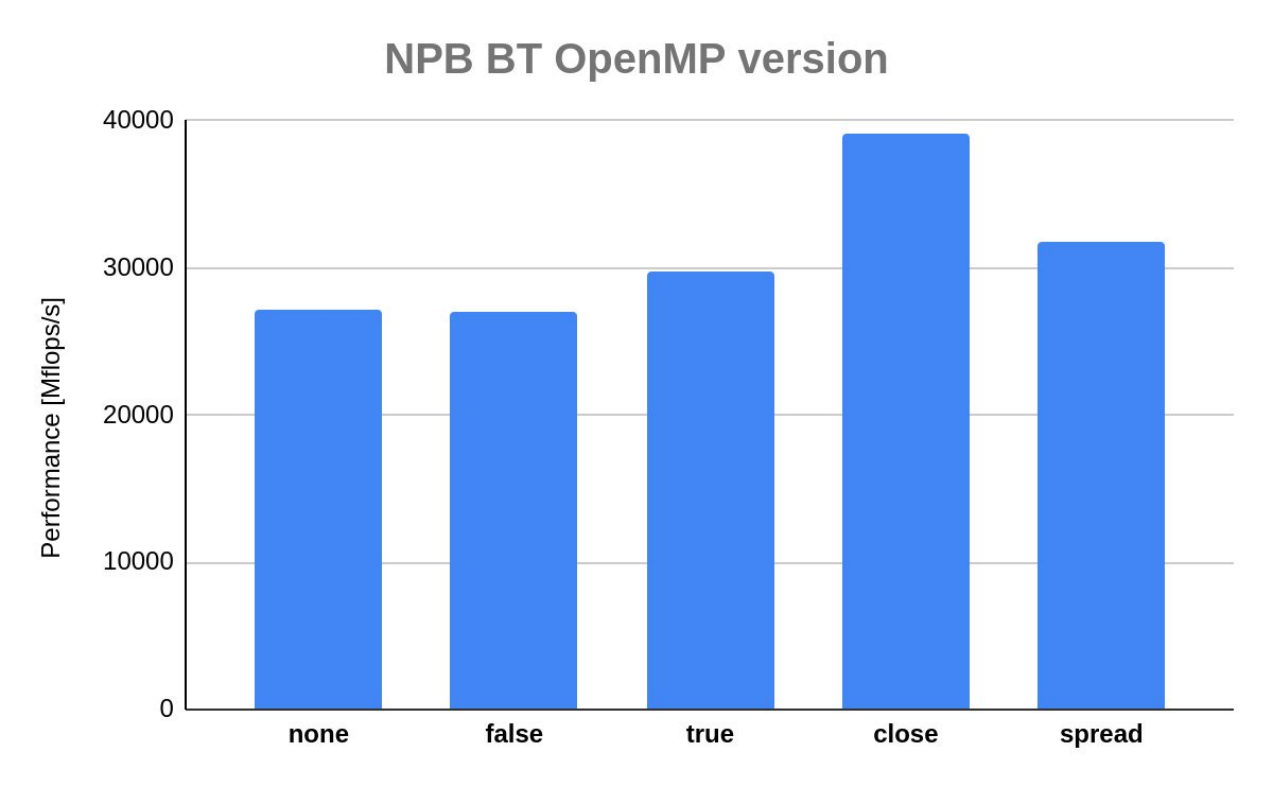
OMP_PROC_BIND

false

true

close

spread



Environment variables

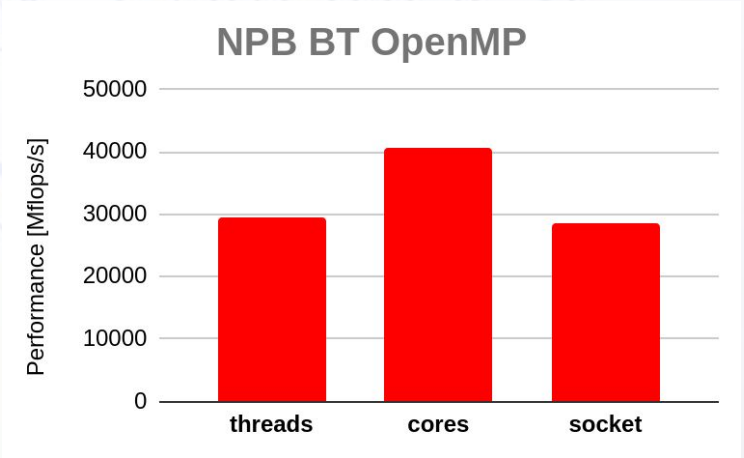
OMP_PLACES

threads «Each place corresponds to a single hardware thread»

cores «Each place corresponds to a single core (having one or more hardware threads)»

sockets «Each place corresponds to a single socket (consisting of one or more cores)»

www.openmp.org



Running MPI programs

#SBATCH --nodes=64 --ntasks-per-node=128 *A total of 8192 MPI ranks*

export I_MPI_<option>=<settings>

mpirun <options> ./a.out *Where option is is not -np, in most cases none*

Tuning MPI program at runtime

I_MPI_PIN=1

I_MPI_BIND_NUMA=localalloc

I_MPI_ADJUST_ALLTOALL

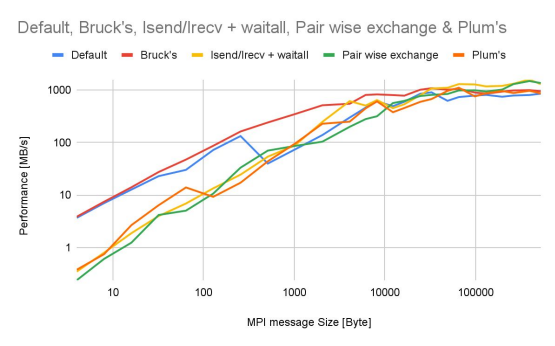
#SBATCH --nodes=1024

Load Intel MPI

export I_MPI_<option>=<settings>

mpirun <options> ./a.out

NPB Integer Sort
Default : 2966 Mops
ALL2ALLV=1: 3271 Mops



https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference.html

Intel MPI lightweight statistics

export I_MPI_STATS=3 ; export I_MPI_STATS_SCOPE=coll

mpirun ./is.D.64 ; Generated file : stats.txt

```
Operation      Context  Algo  Comm size Message size   Calls  Cost(%)
-----------------------------------------------------------------------
Allreduce           0    2         64           4116      11    0.63
Alltoall            0    1         64              4      11    0.00
Alltoallv           0    1         64      134217728      11   10.16
Bcast               0    7         64              4       1    0.00
Reduce              0    0         64              4       1    0.00
=======================================================================
```

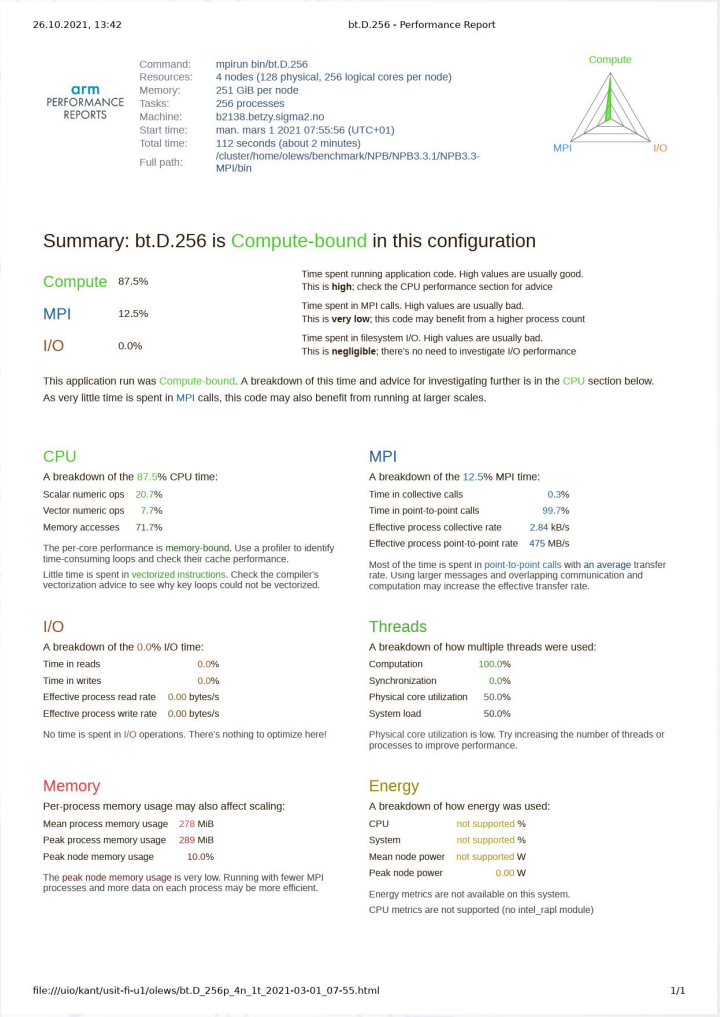
UiO : USIT

ARM performance report provide a quick overview.

```
Summary: bt.D.256 is Compute-bound in this configuration
Compute:                                 87.5% |========|
MPI:                                     12.5% ||
I/O:                                      0.0% |

CPU:

A breakdown of the 87.5% CPU time:
Scalar numeric ops:                      20.7% |=|
Vector numeric ops:                       7.7% ||
Memory accesses:                         71.7% |======|
```



26.10.2021, 13:42 bt.D.256 - Performance Report

arm PERFORMANCE REPORTS

Command:	mpirun bin/bt.D.256
Resources:	4 nodes (128 physical, 256 logical cores per node)
Memory:	251 GiB per node
Tasks:	256 processes
Machine:	b2138.betzy.sigma2.no
Start time:	man. mars 1 2021 07:55:56 (UTC+01)
Total time:	112 seconds (about 2 minutes)
Full path:	/cluster/home/olews/benchmark/NPB/NPB3.3.1/NPB3.3-MPI/bin

Compute MPI I/O

Summary: bt.D.256 is Compute-bound in this configuration

Compute	87.5%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	12.5%	Time spent in MPI calls. High values are usually bad. This is **very low**; this code may benefit from a higher process count
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below. As very little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 87.5% CPU time:

Scalar numeric ops	20.7%
Vector numeric ops	7.7%
Memory accesses	71.7%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the 12.5% MPI time:

Time in collective calls	0.3%
Time in point-to-point calls	99.7%
Effective process collective rate	2.84 kB/s
Effective process point-to-point rate	475 MB/s

Most of the time is spent in point-to-point calls with an average transfer rate. Using larger messages and overlapping communication and computation may increase the effective transfer rate.

I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

Threads

A breakdown of how multiple threads were used:

Computation	100.0%
Synchronization	0.0%
Physical core utilization	50.0%
System load	50.0%

Physical core utilization is low. Try increasing the number of threads or processes to improve performance.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	278 MiB
Peak process memory usage	289 MiB
Peak node memory usage	10.0%

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

Energy

A breakdown of how energy was used:

CPU	not supported %
System	not supported %
Mean node power	not supported W
Peak node power	0.00 W

Energy metrics are not available on this system.

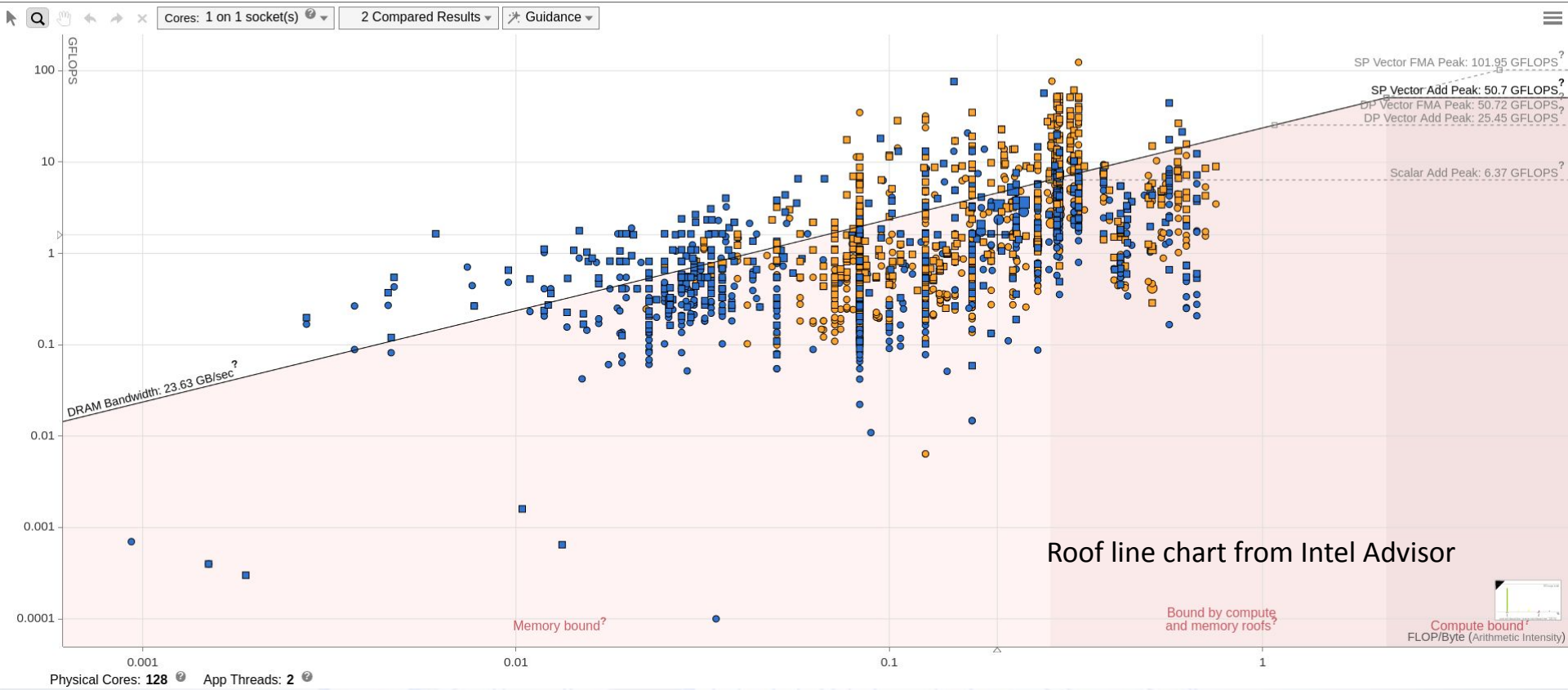
CPU metrics are not supported (no intel_rapl module)

file:///uio/kant/usit-fi-u1/olews/bt.D_256p_4n_1t_2021-03-01_07-55.html 1/1

/cluster/home/olews/benchmark/Bifrost_bench_v1.1.1/RUNS/adv2/rank.0 | 4.2.2021

Roofline Chart from the Intel Advisor

Performance Metrics Summary



Roof line chart from Intel Advisor

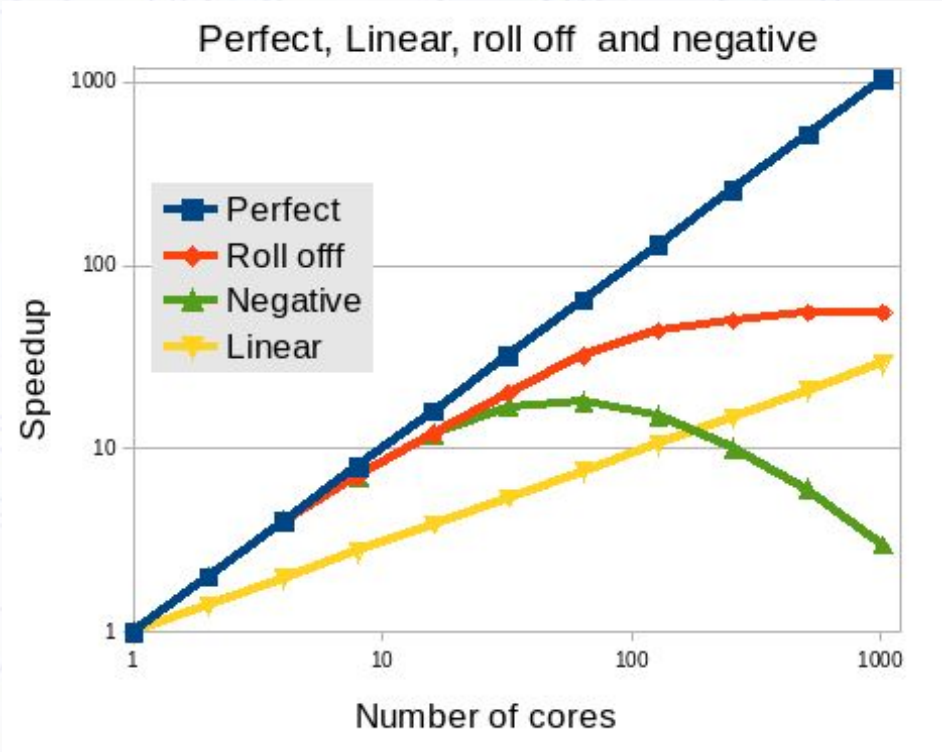
Scaling

SLURM:

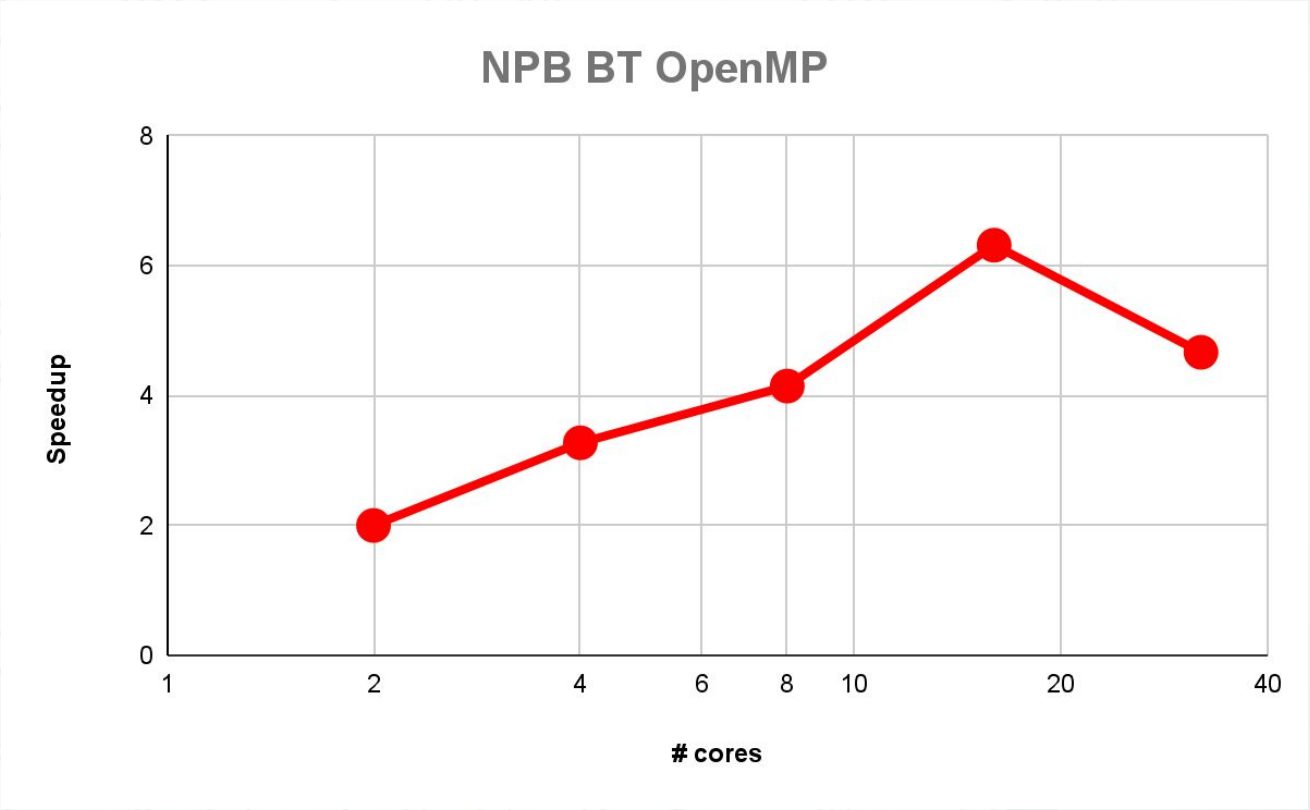
--nodes=N

--tasks-per-node=M

--cpus-per-task=K



#cores	Perf	Speedup
2	12744,8	2
4	20856,1	3,3
8	26422,6	4,1
16	40250,3	6,3
32	29738,0	4,7



Check scaling !

Do a scaling check of your application with relevant input.

Do not use more cores than needed, it’s counterproductive.

/projects/ec34/inf9380/Exercises-23.Mar/