ExPBB: A framework to explore the performance of Burst Buffer

Cray User Group 2017, Redmond, US

George Markomanolis
Computational Scientist
KAUST Supercomputing Laboratory
georgios.markomanolis@kaust.edu.sa
11 May 2017

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Outline

- Motivation
- What should be changed?
- How this framework works?
- Study cases
- Do we need new HPC I/O libraries?
- Conclusions

Leading KAUST Burst Buffer Early Access Program

- Investigation of the relationship between reactivity and aromaticity, Prof. Kuo-Wei Huang, Theo P Goncalves, Kristin Munkerup (Gaussian)
- Tuning of the I/O of a DART-MITGCM workflow at high number of jobs, Prof. Ibrahim Hoteit, Habib Toye Mahamadou Kele, Samuel Kortas (DART-MITGCM)
- Large domain of Saudi Arabia on WRF, Prof. Ibrahim Hoteit, Hari Dasari, Yesubabu Viswandhappli, George Markomanolis (WRF, MATLAB)
- Regional and global modeling of volcanic and dust aerosol impact, Georgiy Stenchikov, Sergey Osipov, Anatolii Anisimov, Mohamed Abdelkader (WRF, ROMS, WRF-CHEM, EMAC)
- High Performance on Burst Buffer for large scale simulations using PIDX and KARFS, Prof. Hong Im, Prof. Valerio Pascucci, Steve Petruzza, Sidharth Kumar, Duong Hoang, Bok Jik Lee, Francisco Hernandez Perez (PIDX, KARFS, IOR)
- KVL Development Project - Burst Buffer for In-Situ Analysis and Visualization, Madhu Srinivasan, Glendon Holst, Thomas Theussl (Paraview, Visualization tools)
- Performance of membrane spacers having different new geometries: A Direct Numerical Simulation (DNS) Investigations, Prof. Noreddine Ghaffour, Adnan Qamar, Rooh Khuram (ANSYS - Fluent, Tecplot)
- Introducing checkpointing in flow simulations with lattice-Boltzmann code, Prof. Tadeusz Patzek, Shiarhei Khirevich
- Data Offloading for Extreme Scale Simulation

Burst Buffer Training

2 hours video of KAUST Bust Buffer training with title
Burst Buffer: From Alpha to Omega
https://www.youtube.com/watch?v=8zLcZmiTweg
slides: https://goo.gl/2Juf24

Markomanolis, George (2017):
Getting started with the Burst Buffer. figshare.
https://doi.org/10.6084/m9.figshare.4871738

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Motivation

- Burst Buffer (BB) does not provide the expected performance… or we do not know how to use it?
- A user should be familiar with some technical details and most of them are science-focus researchers.
- We need a tool that a user can execute and extract the optimized parameters for his application and the used domain.
- **Disclaimer: KAUST site works with CLE v5.2 (May 2017)**

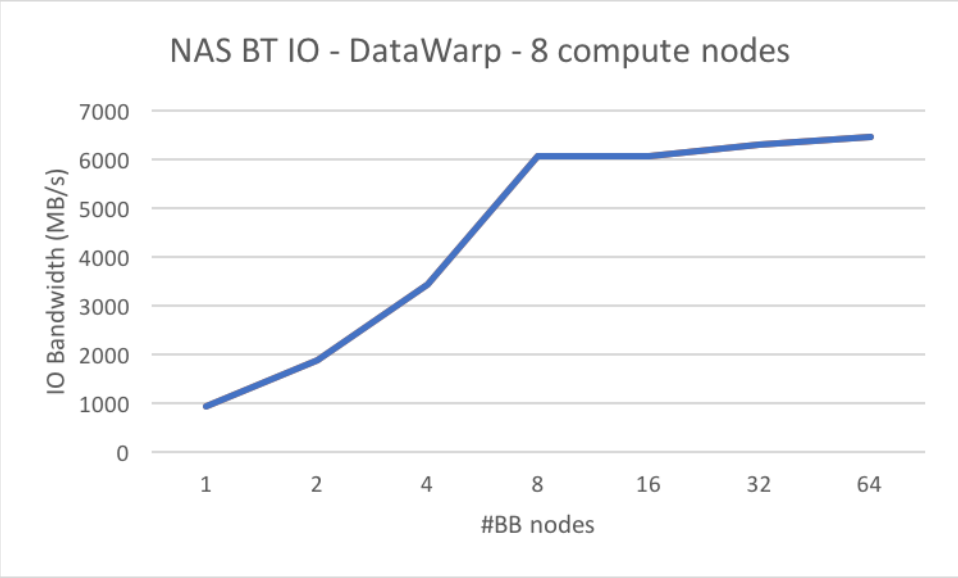
Applications

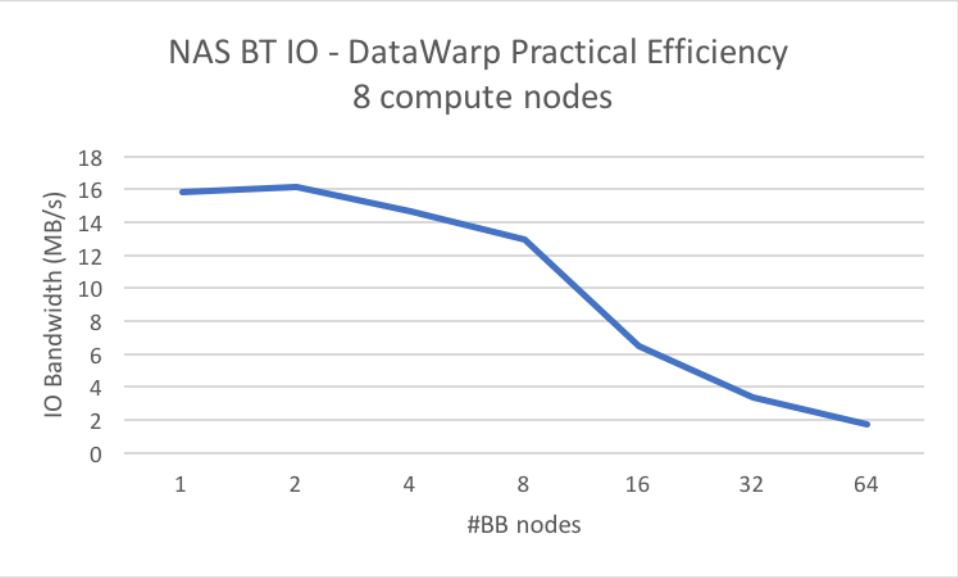
- NAS BT I/O
- Domain size: 1024 x 512 x 256
- 256 to 1024 MPI processes, 8 – 32 nodes
- Size of output file: 50 GB

- Weather Research and Forecasting Model coupling with Chemistry
- Small domain: 330 x 275 x 35
- Size of input file: 804 MB
- Size of output file: 2.9GB, it is saved every one hour of simulation
- Output file quite small
- For all the WRF-CHEM experiments we use 1280 MPI processes (40 nodes), as this is the optimum for the computation/communication
- For the default case, we stage-in all the files and we execute the simulation from BB

Burst Buffer nodes

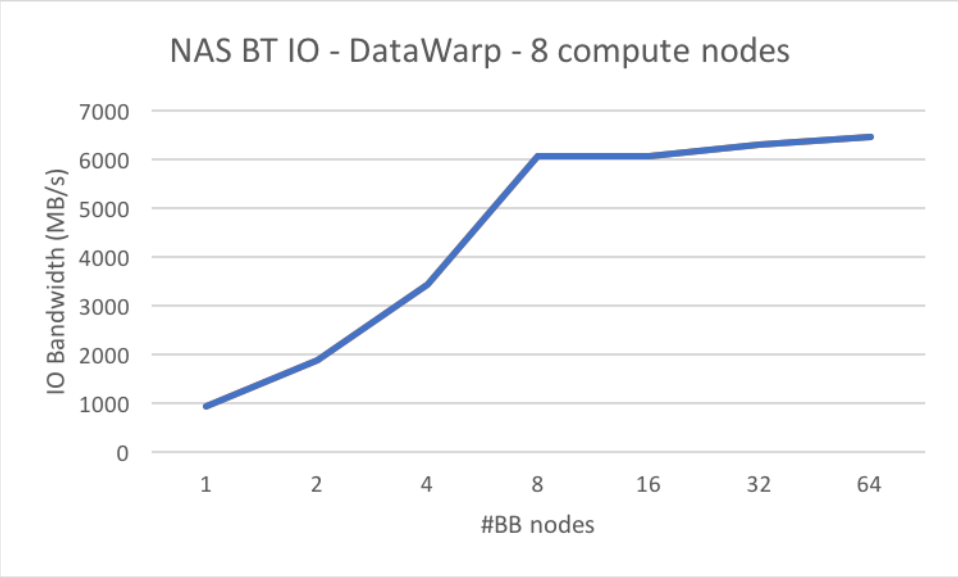
- A user tries to scale his application on Burst Buffer by just increasing the BB nodes and this does not always provide the best results.
- Increasing the BB nodes by 64 times, provide less than 8 times better performance and the practical efficiency is less than 2%!

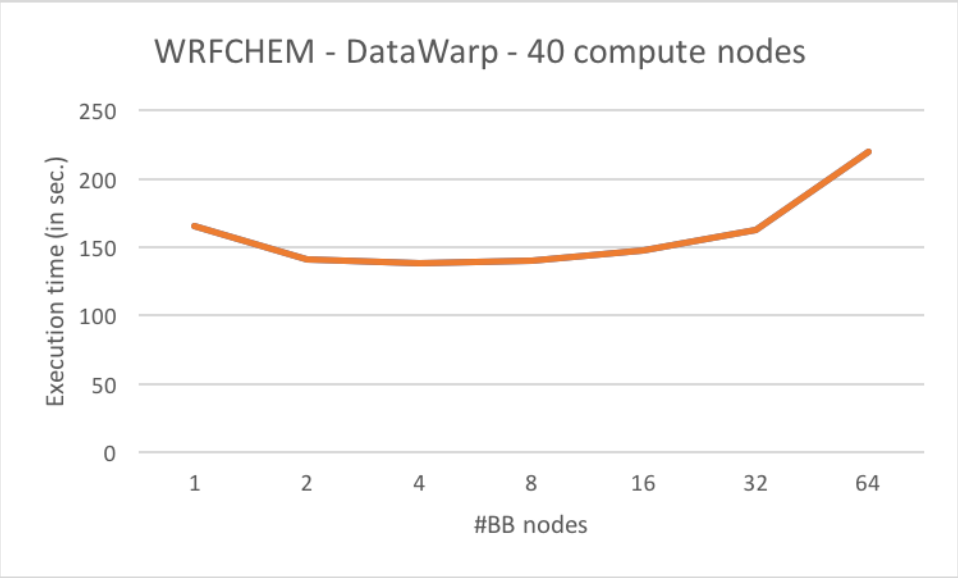




Burst Buffer nodes

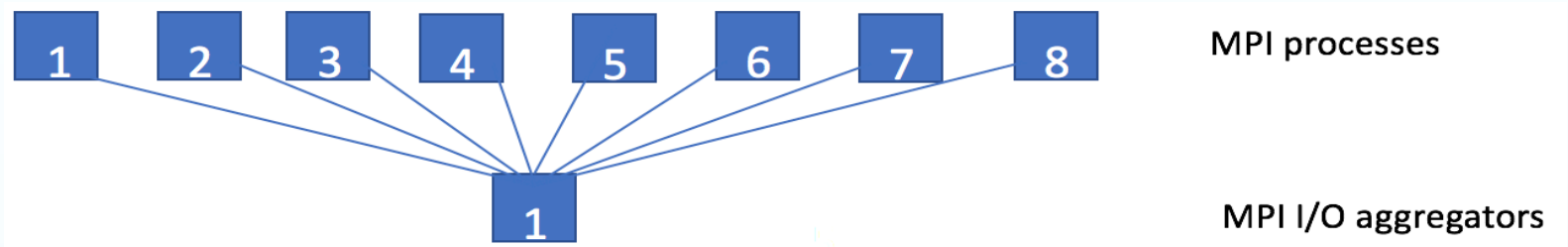
- A user tries to scale his application on Burst Buffer by just increasing the BB nodes and this does not always provide the best results.
- Trying to scale WRF-CHEM, the execution time increases while we increase the number of BB nodes.

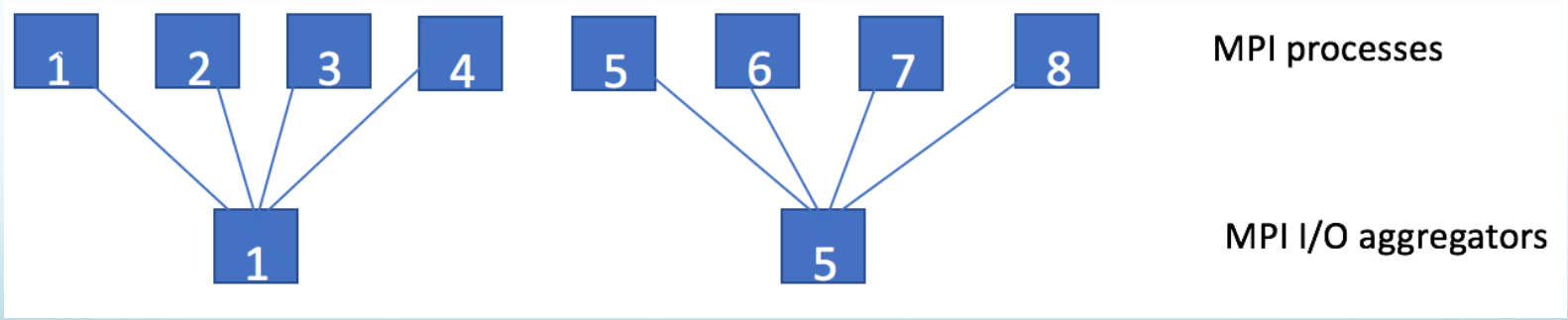




Collective Buffering – MPI I/O aggregators

- During a collective write, the buffers on the aggregated nodes are buffered through MPI, then these nodes write the data to the I/O servers.
- The default MPI I/O aggregators on Burst Buffer, is the number of BB nodes. So, if we have 8 MPI processes and 1 BB node, then we have 1 MPI I/O aggregator, if we have 2 BB nodes, then we have 2 MPI I/O aggregators.



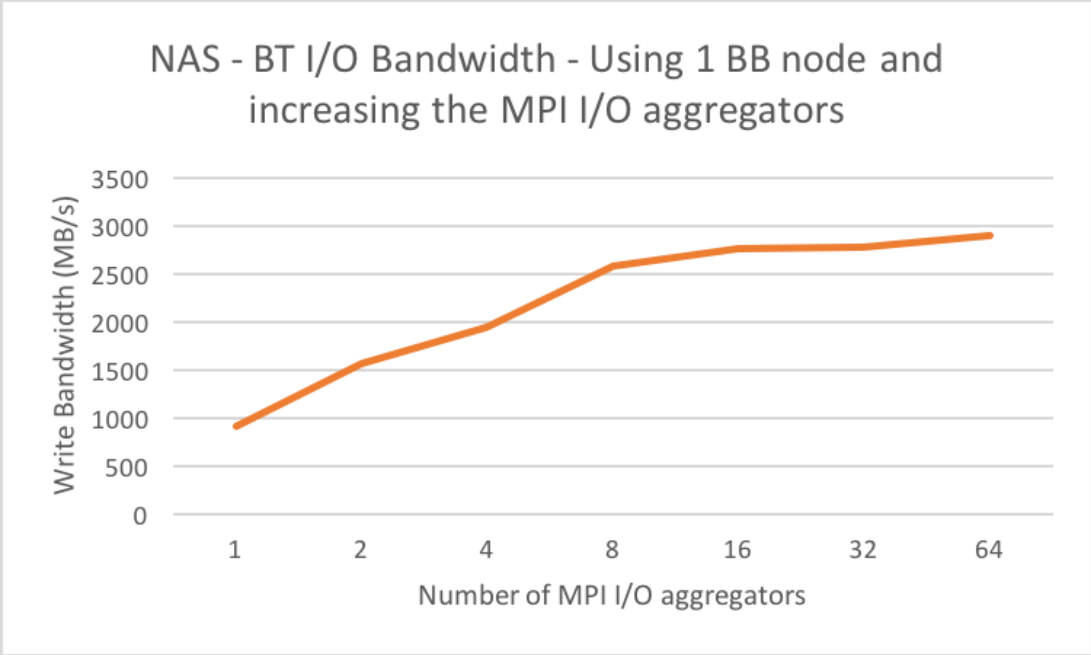


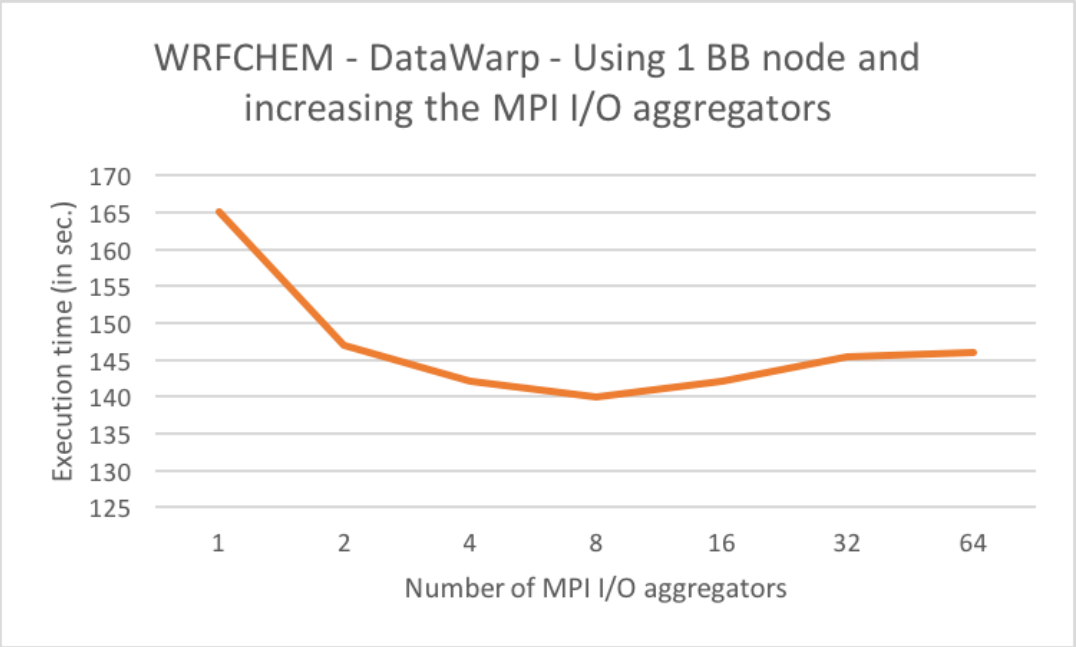
export MPICH_MPIIO_HINTS="wrfrst*:**cb_nodes**=80,wrfout*:**cb_nodes**=40"

This environment variable is supported on DataWarp since Cray-MPICH v7.4

Collective Buffering – MPI I/O aggregators II

- Using optimized MPI I/O aggregators improved the performance up to 3,11 times on just one BB node for NAS BT I/O and 15.15% for WRF-CHEM.
- For NAS BT I/O, we achieved best performance with 64 MPI I/O aggregators while for WRF-CHEM with 8 MPI I/O aggregators.

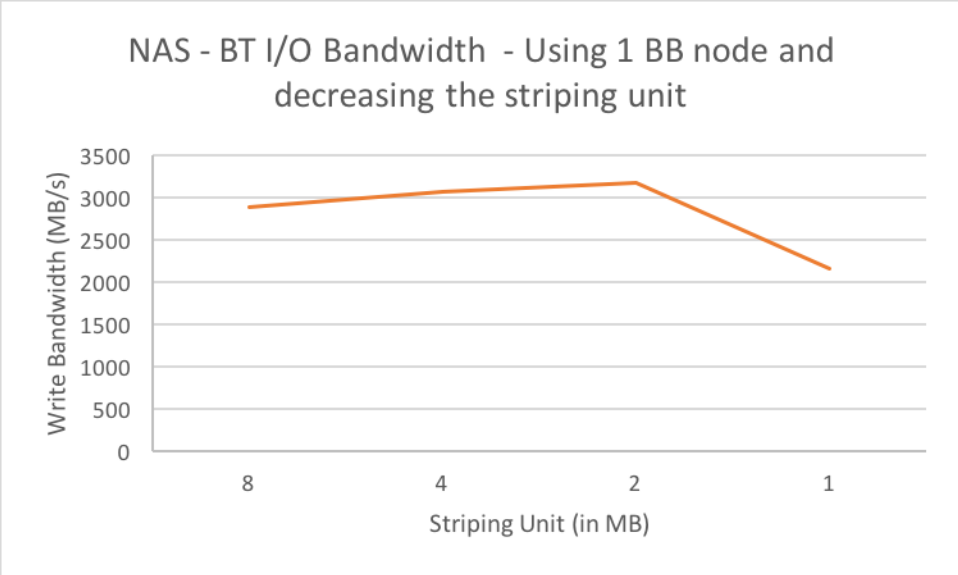


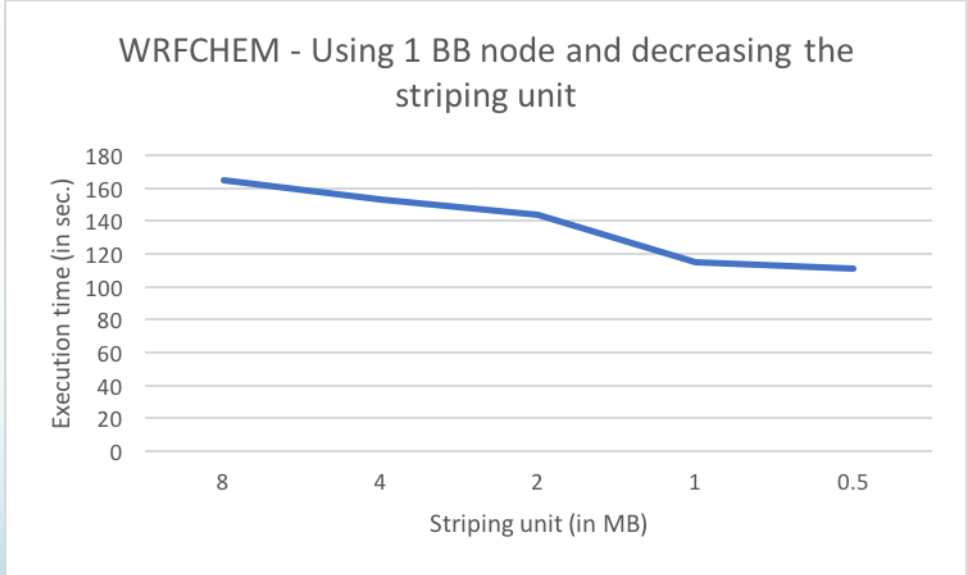


Use 64 MPI I/O aggregatos for the file btio.nc: export MPICH_MPIIO_HINTS=btio.nc:**cb_nodes=64**

Striping Unit

- The stripe units are the segments of sequential data written to or read from a disk before the operation continues to the next disk
- For NAS BT IO, decreasing the striping unit up to 2 MB, increases the performance by 10%, while for WRF-CHEM, decreasing the striping unit to 0.5 MB, increases the performance by 32%.





Change striping unit of file btio.nc to 2MB:
export MPICH_MPIIO_HINTS="btio.nc:cb_nodes=64:**striping_unit=2097152"**

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Striping Buffer

- Striping buffer improves the performance, specially with cases with large I/O, for example in one case of WRF, doubling the striping buffer improved the performance by 25%.

MPI I/O Statistics

- CRAY-MPICH supports MPI I/O statistics by using MPICH_MPIIO_STATS=1 or 2

```
MPIIO write access patterns for wrfout_d01_2007-04-03_00_00_00
  independent writes                = 2
  collective writes                 = 552960
  independent writers               = 1
  aggregators                       = 1
  stripe count                = 1
  stripe size                       = 8388608
  system writes               = 797
  stripe sized writes               = 114
  total bytes for writes      = 3045341799=2904 MiB
  ave system write size       = 3821006
  read-modify-write count           = 0
  read-modify-write bytes           = 0
  number of write gaps = 3
  ave write gap size                = 8351037
```

797 system writes
14.3% striped writes (low)

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Framework preparation I

- **Fill in the required information in the beginning of the ExPBB script**
- export executable="btio”
- #Declare option for the executable (leave empty if no arguments)

  export arguments="inputbt1.data"
- #Declare path file with output information such as MPI statistics

  export output_file=”output.txt"
- #Declare the minimum requested Burst Buffer size in GB

  export min_bb_size=1
- #Should the tool investigate different number of MPI/OpenMP processes? (0 for no, 1 for yes)

  export resources=1

Framework preparation II

- #MIN nodes that you need for your problem (less or equal to the reserved nodes)

  export min_nodes=100
- #Declare stage-in folder, full path

  export stage_in="/project/k01/markomg/development/expbb"
- #Declare stage-out folder, full path

  export stage_out="/project/k01/markomg/back2"
- #Do you want to use only Lustre (0), only Burst Buffer (1), or both (2)?

  export filesystem=1
- **The executable is required to have been compiled with the Darshan profiling tool**

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Important MPI environment variables

- export MPICH_ENV_DISPLAY=1
  - Displays all settings used by the MPI during execution
- export MPICH_VERSION_DISPLAY=1
  - Displays MPI version
- export MPICH_MPIIO_HINTS_DISPLAY=1
  - Displays all the available I/O hints and their values
- export MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1
  - Display the ranks that are performing aggregation when using MI-I/O collective buffering
- export MPICH_MPIIO_STATS=1 or 2
  - Statistics on the actual read/write operations after collective buffering
- export MPICH_MPIIO_HINTS="..."
  - Declare I/O hints

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Execution of ExPBB

- If your submission script is called run.sh, then execute:

  expbb run.sh

- Then the following will happen:
  - A parser will extract the compute resources from the original script and it will add the corresponding #DW commands in a copy of the original script. From the requested GBs the number of minimum BB nodes will be calculated.
  - The previous important MPI environment variables are added to all the new generated submission scripts
  - Two executions will take place, one on Lustre and one on BB. This happens for two reasons, first to extract the basic execution time for comparison reasons, and second to extract the default striping unit and buffer for each case.

Execution of ExPBB II

- Then the tool will create a new submissions script depending on the number of the BB nodes, for example on Shaheen II we have 268 BB nodes, if we need 4 BB nodes minimum, then there will be scripts for 4, 8, 16, 32, 64, 128, and 256 BB nodes, **if** they are available and not drained.
- Each of the script includes extra code before and after the *srun* command, where loops change the values of the parameters, where their range depends on the default values extracted on the first BB execution.
- After the srun command a command to a parser is called, where it reads the Darshan performance data and tests the following rules:
  - Is the I/O for the studied file faster than the previous execution?
    - If yes, is the total execution time faster?
      - If yes, continue to the next value of the studied parameter
      - If not, extract from Darshan data, if there is a small file with shared access that consumes significant time, start the hybrid execution.
- The first script will be submitted with the minimum requested nodes and it will start investigating the results.
- All the results will be written in txt files that are easily accessible

ExPBB example – Original script

Original script

```
#!/bin/bash -x

#SBATCH --partition=workq
#SBATCH -t 06
#SBATCH -A k01
#SBATCH  --ntasks=256
#SBATCH --ntasks-per-node=32
#SBATCH --ntasks-per-socket=16
#SBATCH   -J btio
#SBATCH -o btio_out_%j
#SBATCH -e btio_err_%j

echo $SLURM_SUBMIT_DIR >> inputbt1.data

time srun --ntasks=256 --ntasks-per-node=32 \
--threads-per-core=1 --hint=nomultithread \
./btio inputbt1.data

exit 0
```

ExPBB example – Converted script

Script converted with ExpBB
Code not final, to be modified in the released version

```
#!/bin/bash -x

#SBATCH --partition=workq
#SBATCH -t 384
#SBATCH -A k01
#SBATCH  --ntasks=256
#SBATCH --ntasks-per-node=32
#SBATCH --ntasks-per-socket=16
#SBATCH   -J btio
#SBATCH -o btio_out_%j
#SBATCH -e btio_err_%j
#DW jobdw type=scratch acces
#DW stage_in type=directory
#DW stage_out type=directory

export err_file=btio_err_$SL
export study_file="btio.nc"
export MPICH_ENV_DISPLAY=1
export MPICH_VERSION_DISPLAY
export MPICH_MPIIO_HINTS_DIS
export MPICH_MPIIO_STATS=2   g"`
export MPICH_MPIIO_AGGREGATO

cd $DW_JOB_STRIPED
chmod +x btio                                   ize)); expbb_stripe_s
export folder=${SLURM_SUBMIT
export best_run_id=${SLURM_J
export execu=btio
export exp_id=1                                 //g"`
export run_id=1
cp del_tmp inputbt1.data
echo $DW_JOB_STRIPED >> inpu                    //g"`
                                                  |
let total_tasks=256
let nodes=8
export MPICH_MPIIO_HINTS="btio.nc"
let expbb_mpi_tasks=256
let expbb_omp_tasks=1
        if [ "$expbb_mpi_tasks" -gt "$nodes" ]; then
                                                for (( expbb_stripe_buffer=$(($default_stripe_buffer/8)); expbb_stripe_buffer<=$((4*$de
        fi      fault_stripe_buffer)); expbb_stripe_buffer=2*$expbb_stripe_buffer ));
        expor                                   do
        end_i           if [ "$expbb_stripe_buffer" -eq "$(($default_stripe_buffer/8))" ] && [ $exp_id -gt 1 ]; then
        for (   export temp_stripe_buffer=$((4*$default_stripe_                 if [ $com -gt 0 ]; then
        do      export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HI export best_run_id=${SLURM_JOBID}_$(($run_id-1))
                fi                                                      echo $best_run_id > ${SLURM_SUBMIT_DIR}/best_run_id.txt
                if [ "$expbb_stripe_buffer" -gt "$(($de                 break
                export temp_stripe_buffer=$(($expbb_stripe_buff         else
                export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HI export best_run_id=${SLURM_JOBID}_$run_id
                fi                                                      export check_all=`paste $folder/$execu/$year/$month/$day/${SLURM_JOBID}_$run_id $folder/$execu/$year/$month/$day/${best
                                                export MPICH_MF         _run_id} | awk '{if((($6-$12)/$12)>0.05) print 1; else print 0;}'`
                        export START=$(date +%s.%N)                     if [ $check_all -gt 0 ]; then
                        time srun --ntasks=$expbb_mpi_t                 cd
                pbb_omp_tasks} --threads-per-core=1 --hint=nomu         export myself=`scontrol show job ${SLURM_JOBID} | grep Command | awk 'BEGIN{FS="/"} {print $NF}
                        END=$(date +%s.%N)                      '`
                        DIFF=$(echo $END - $START | bc)                 cp $myself hybrid_$myself
                chmod +x parse_darshan.sh                               sed s/cd $DW_JOB_STRIPED// -i hybrid_
                ./parse_darshan.sh $SLURM_JOBID $run_id         echo $best_run_id > ${SLURM_SUBMIT_DIR}/hybrid_run_id.txt
                let year=`date +%Y`                                     else
                let month=`date +%-m`                           echo $best_run_id > ${SLURM_SUBMIT_DIR}/best_run_id.txt
                let day=`date +%-d`                                     fi
                export total_io=`cat $folder/$execu/$ye         fi
                nt s}'`                                         fi
                export percentage=`echo "scale=2; 100*$                 let run_id=${run_id}+1
                echo "The percentage of the total execu         if [ "$exp_id" -gt "1" ]; then
                export line=`grep -n "| MPIIO" ${SLURM_ export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HINTS" | sed "s/:striping_buffer=${expbb_stripe_buffer}//g"`
        " | tail -n 1 | awk 'BEGIN{FS=":"} {print $1}'`       fi
        sed -n $line,$(($line+14))p ${SLURM_SUBMIT_DIR}                 grep "| MPIIO" ${SLURM_SUBMIT_DIR}/$err_file | grep patt | grep -v darshan | awk '{print $NF}' | uniq >
                echo "BB_nodes=1 IO aggregators         pe_buffer time $DIFF" >> ${SLURM_SUBMIT_DIR}/re> ${SLURM_SUBMIT_DIR}/pio_files_${SLURM_JOBID}.txt
                if [ $run_id -gt 1 ]; then              let exp_id=$exp_id+1
        export com=`paste $folder/$execu/$year/$month/$                 done;
        d} | awk '{if(($4-$10)<0) print -1; else print          done;
done


exit 0
```

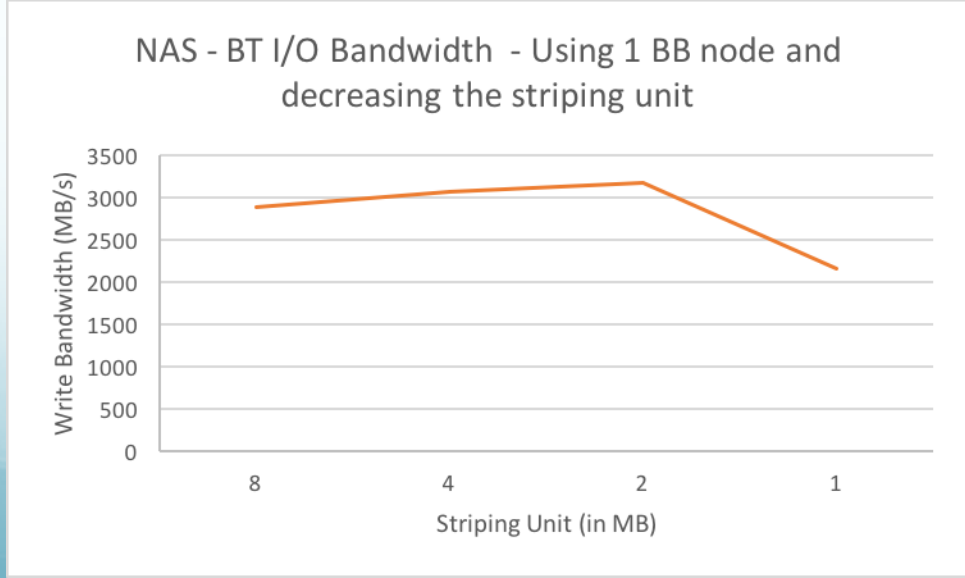
NAS BT I/O - Understanding striping unit

2897 MB/s

```
| MPIIO write access patterns for
|/var/opt/cray/dws/mounts/batch/3129772/ss//btio.nc
|   independent writes      = 11
|   collective writes       = 40960
|   independent writers     = 1
|   aggregators             = 64
|   stripe count            = 1
|   stripe size             = 8388608
|   system writes           = 6411
|   stripe sized writes     = 6400
|   total bytes for writes  = 53687091532 = 51200 MiB = 50 GiB
|   ave system write size   = 8374214
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 21
|   ave write gap size      = 23336707978
```

2165 MB/s

```
| MPIIO write access patterns for
|/var/opt/cray/dws/mounts/batch/3151099/ss//btio.nc
|   independent writes      = 11
|   collective writes       = 40960
|   independent writers     = 1
|   aggregators             = 64
|   stripe count            = 1
|   stripe size             = 1048576
|   system writes           = 51211
|   stripe sized writes     = 51200
|   total bytes for writes  = 53687091532 = 51200 MiB = 50 GiB
|   ave system write size   = 1048350
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 21
|   ave write gap size      = 23297910666
```

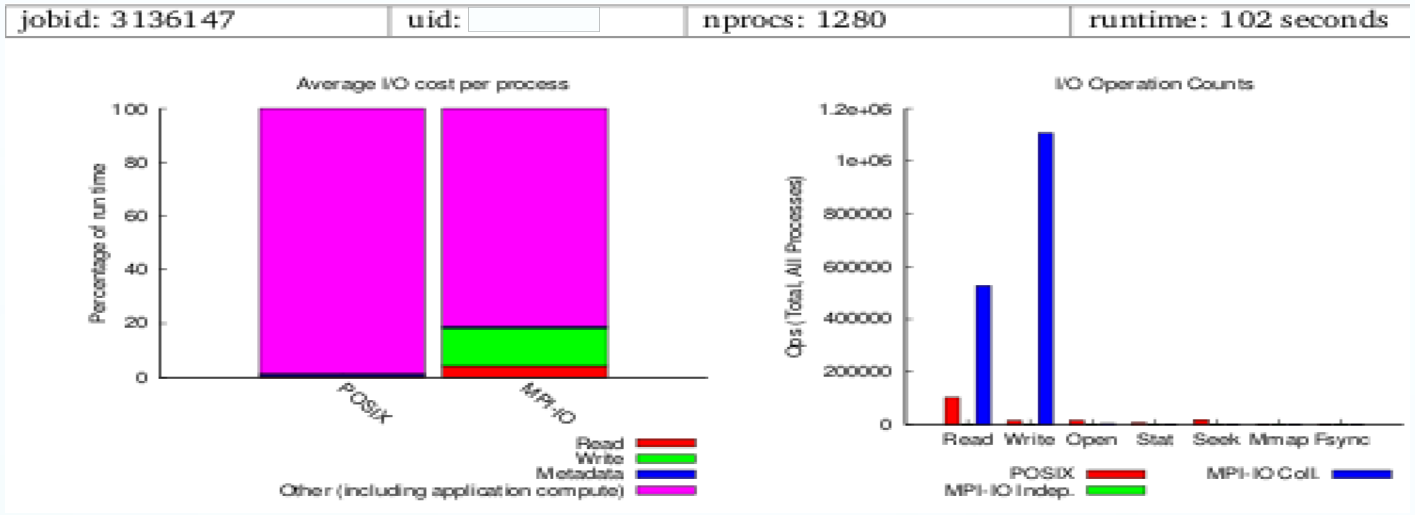


By **decreasing** the **striping unit** by 8 times, the **system writes** were increased by 8 times.

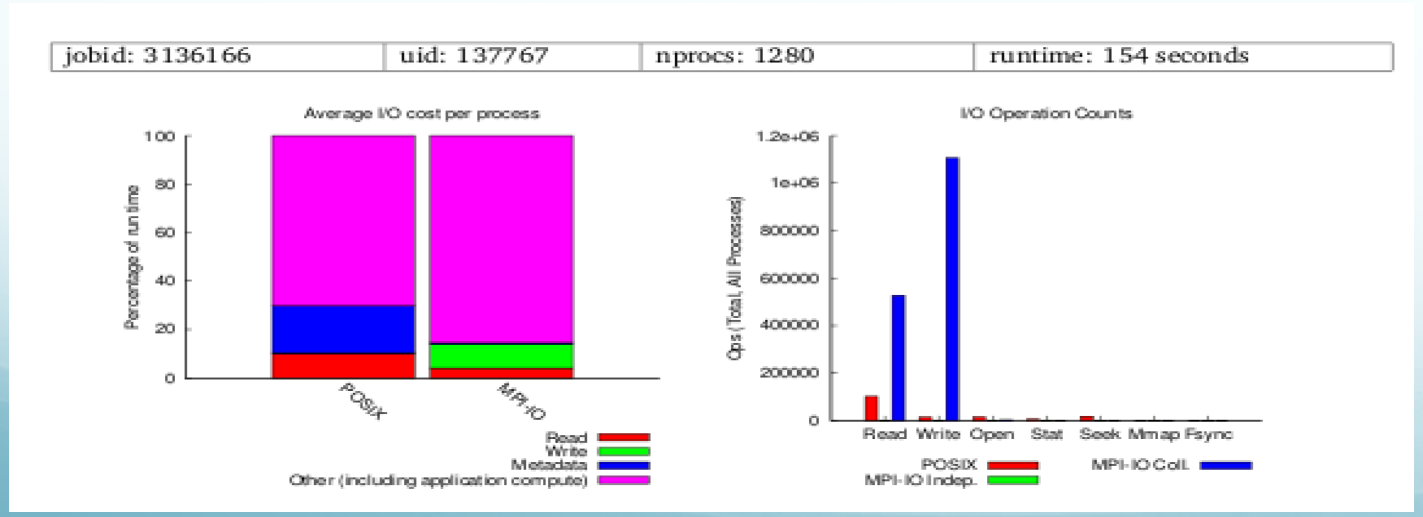
Doubling the number of **BB nodes** from 1 to 2, the I/O bandwidth from 2165 MB/s becomes 3850 MB/s, **78.2%** improvement.

Understanding metadata issue I

WRF-CHEM



2 BB nodes



40 BB nodes

By increasing the number of BB nodes, the **metadata** issue was significant, and the total execution time was increased by **50%**! In this case, we should execute hybrid simulation

Understanding metadata issue II

WRF-CHEM

Variance in Shared Files

File Suffix	Processes	Fastest			Slowest			σ	
		Rank	Time	Bytes	Rank	Time	Bytes	Time	Bytes
...-04-03_01_00_00	1280	1019	7.357200	0	0	8.011398	182M	0.107	2.11e+07
...-04-03_00_00_00	1280	43	7.202004	0	0	7.237024	182M	0.1	2.11e+07
...ss/wrfinput_d01	1280	1	3.520925	0	1279	5.043351	0	0.0457	2.03e+07
.../namelist.input	1280	1042	0.360680	22K	198	0.671558	22K	0.0474	5.51e+03
...e_lat.formatted	1280	129	0.001137	536	325	0.309750	536	0.0836	0
...ozone.formatted	1280	116	0.009035	531K	1057	0.260530	531K	0.0751	0
...._plev.formatted	1280	1141	0.011253	708	1029	0.224185	708	0.0348	0

2 BB nodes

Variance in Shared Files

File Suffix	Processes	Fastest			Slowest			σ	
		Rank	Time	Bytes	Rank	Time	Bytes	Time	Bytes
.../namelist.input	1280	153	20.415039	22K	0	36.595398	215K	3.81	5.51e+03
...._plev.formatted	1280	1212	0.157083	708	88	9.920437	708	0.839	0
...-04-03_01_00_00	1280	1019	7.685493	0	0	8.316938	19M	0.061	6.3e+06
...-04-03_00_00_00	1280	979	7.503696	0	0	7.511409	19M	0.0554	6.3e+06
...ss/wrfinput_d01	1280	4	5.759557	0	1279	7.327778	0	0.0648	6.06e+06
...e_lat.formatted	1280	1212	0.011889	536	182	6.146939	536	1.86	0
...ozone.formatted	1280	1258	0.013096	531K	580	2.014465	531K	0.609	0

40 BB nodes

A shared file called namelist.input with size of 12KB, needs 36.6 seconds to be read from the slowest process!

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

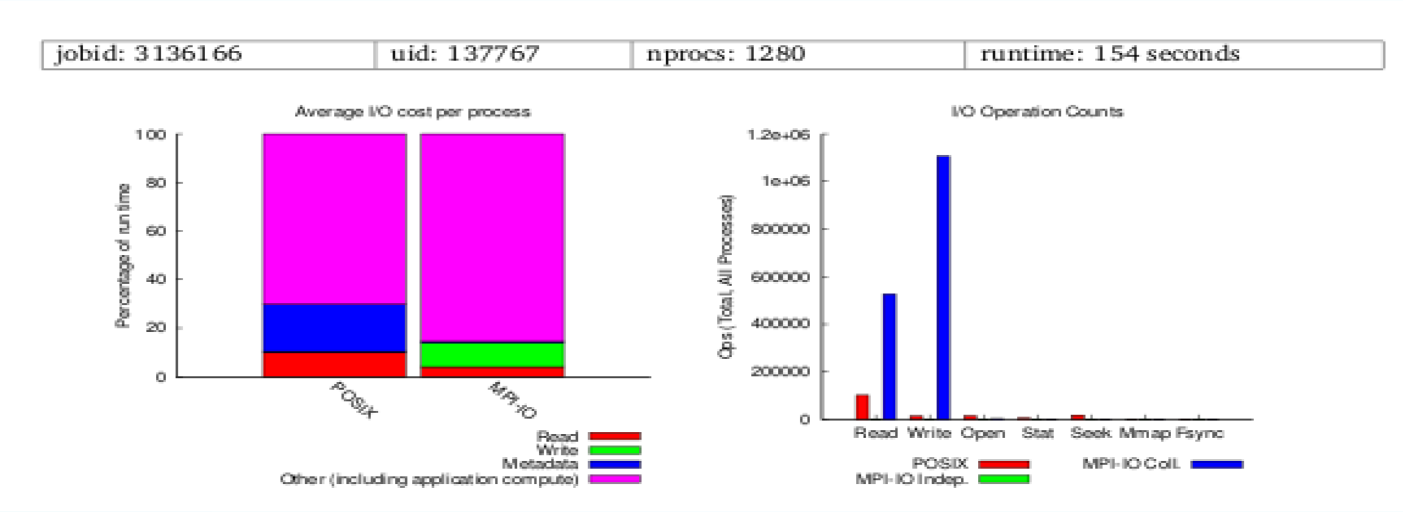
Hybrid simulation

- The user should create a script where is declared what should be changed to use Burst Buffer **only** for the large files

- The execution occurs from the Lustre. Small files are saved on Lustre, large files on BB (does the application support it?)

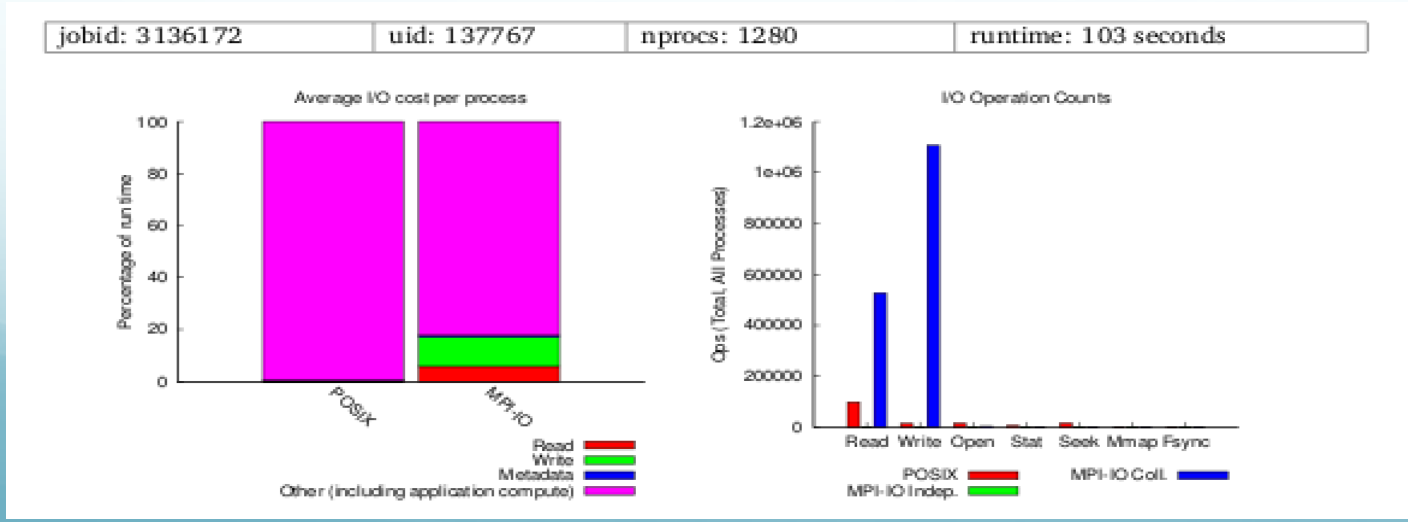
Hybrid simulation - Darshan I

WRF-CHEM



40 BB nodes

Hybrid simulation, decreased the execution time **50%**.



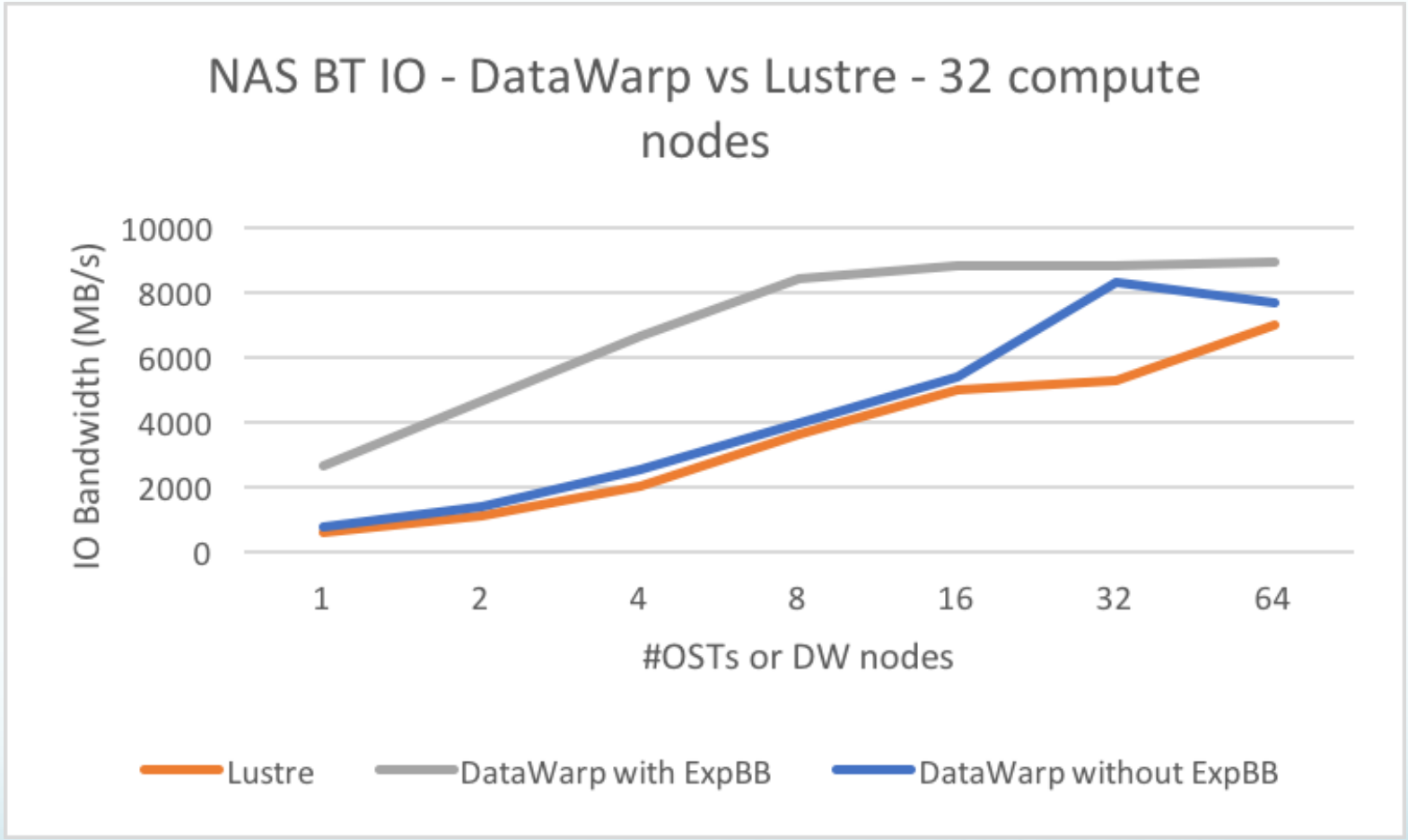
Hybrid, 40 BB nodes

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Rules

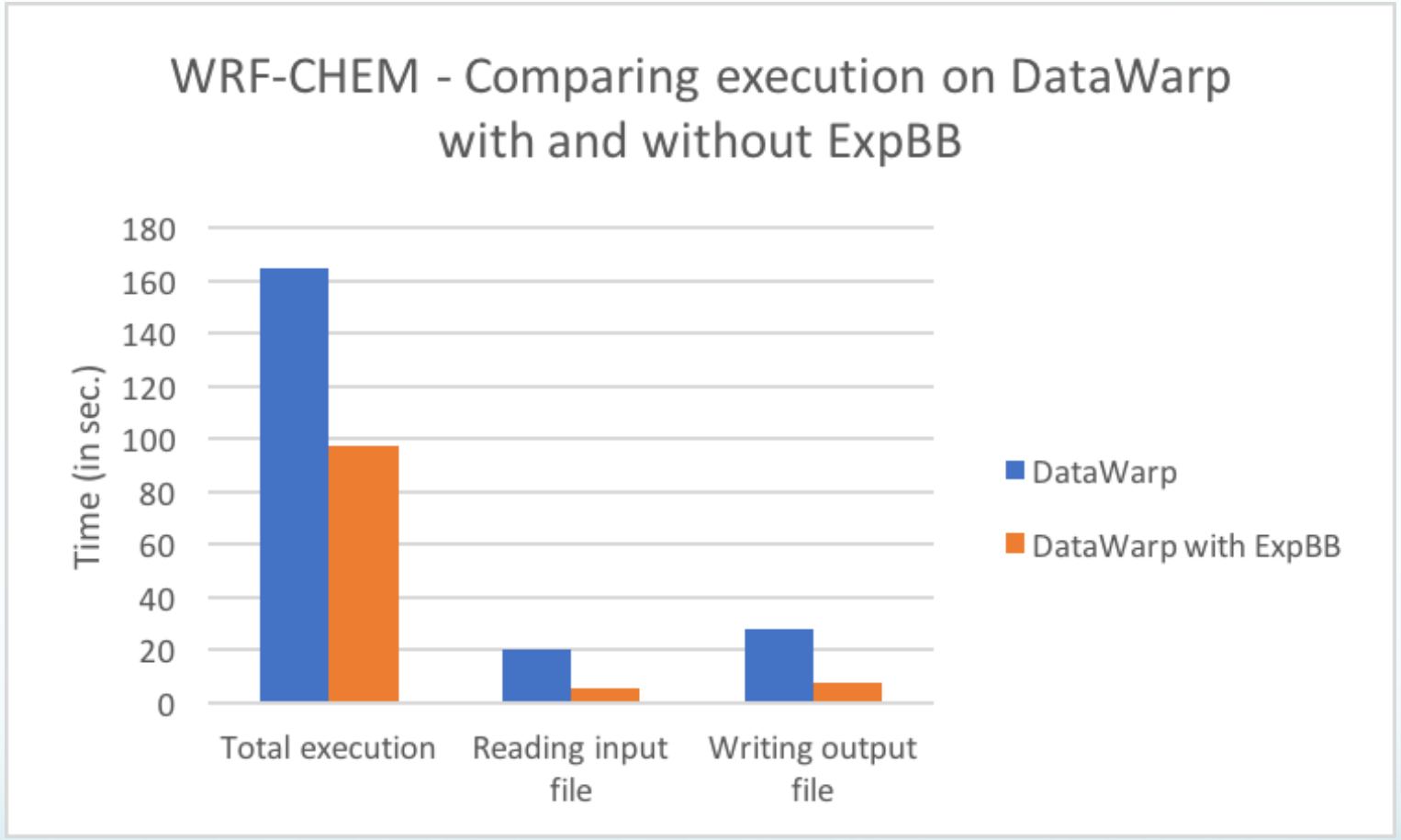
- If the performance becomes worse while we decrease the striping unit and the number of system write/reads is significant large, then submit a submission script with double number of BB nodes with updated starting values for parameters, and stop the current execution.
- If the performance becomes worse when we increase the number of BB nodes, and there is according to Darshan, a small shared file that all the processes access to it and it takes significant time to read, then execute hybrid simulation.

Results I



We observe that for 8 BB nodes, with ExPBB framework, we have better performance than 64 BB nodes without ExPBB or Lustre. The maximum speedup compared to default BB execution, is 4,52. Moreover, 8 BB nodes have better performance than 64 OSTs

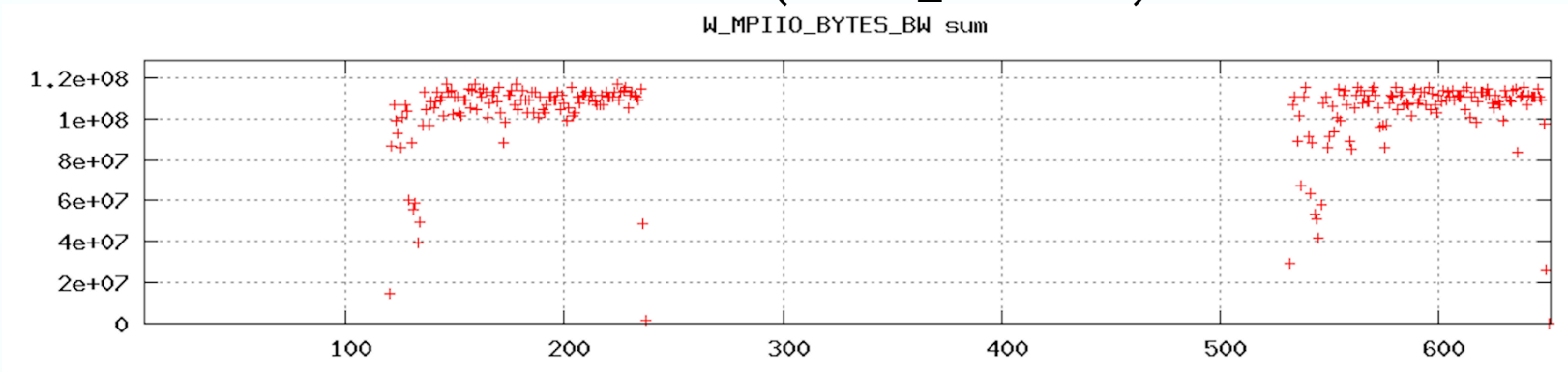
Results II



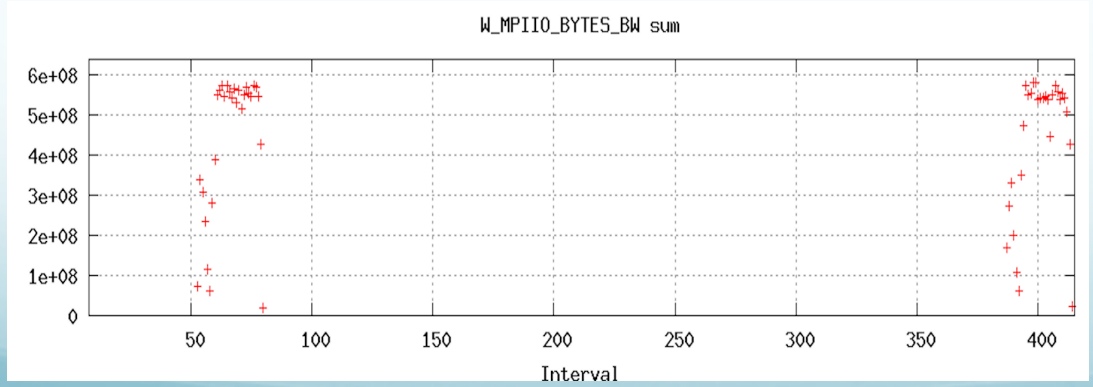
The total execution time is improved 1,7 times with ExPBB and the I/O is improved up to 3,8 times for 1 BB node. Finally, the total execution time is 13.4% faster than Lustre with 64 OSTs.

Results III

WRF-CHEM – cray_mpiio_summary tool (MPIIO_STATS=2)

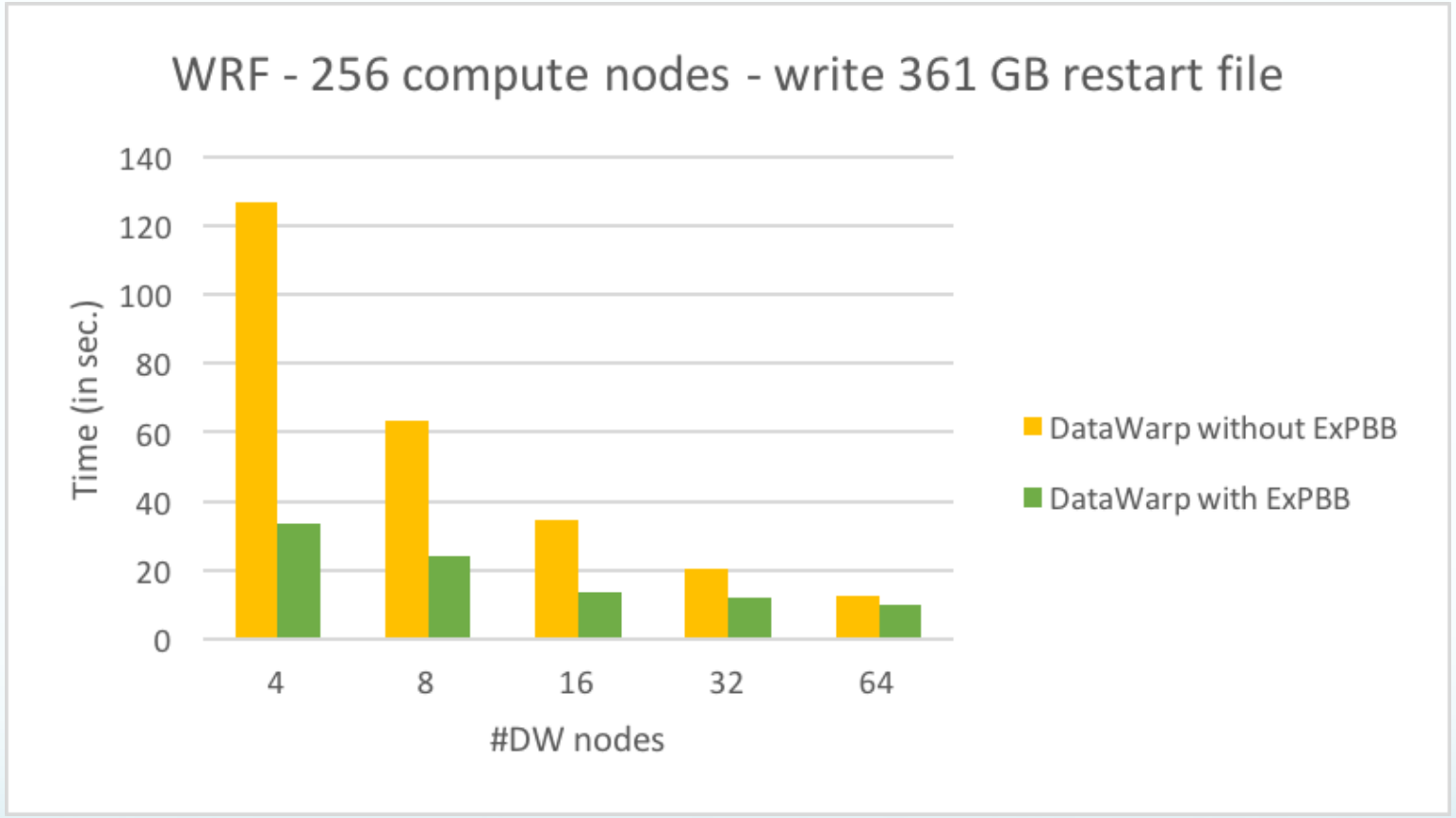


BB without ExPBB



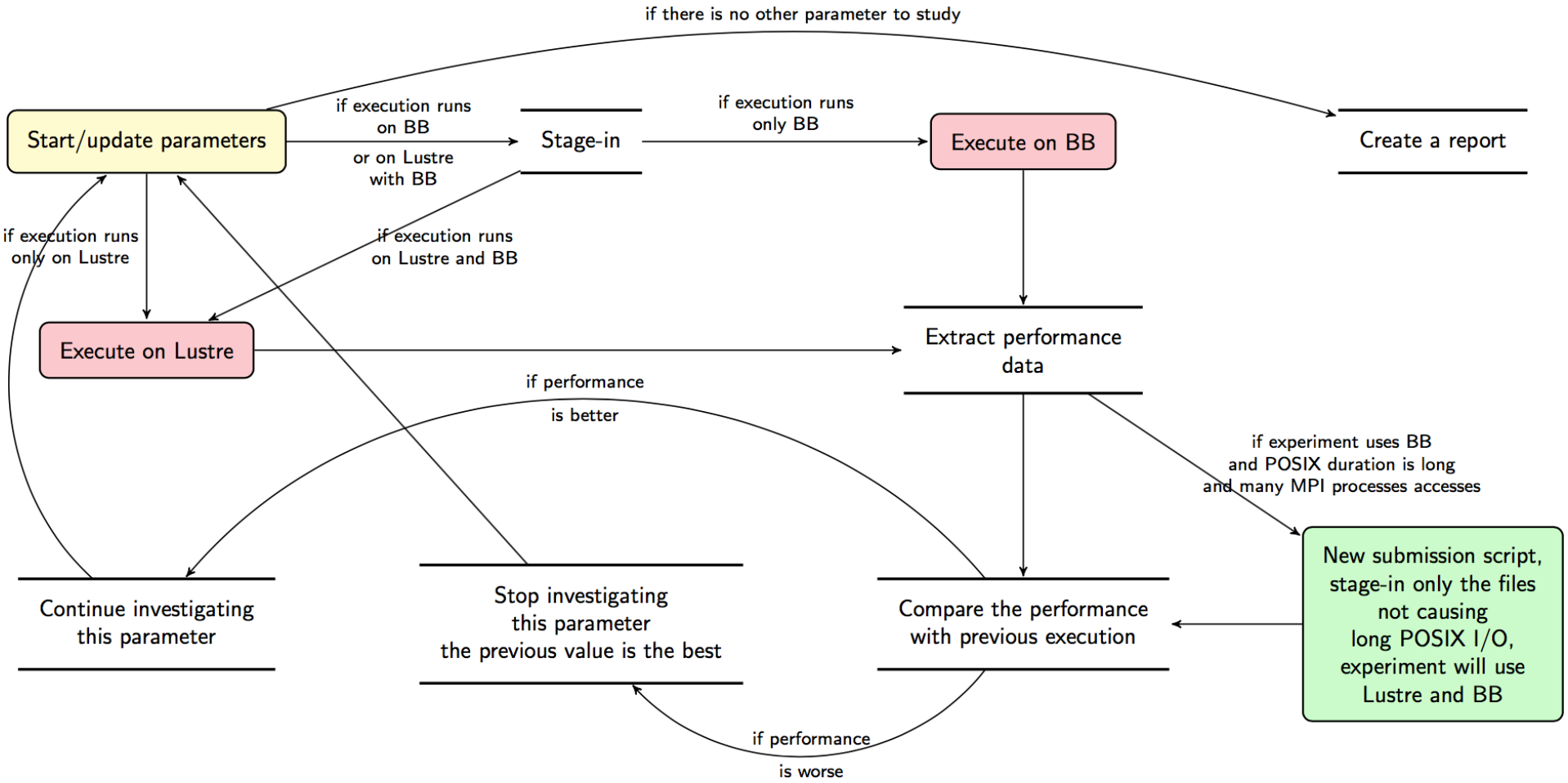
BB with ExPBB

Results IV



The I/O was improved with ExPBB between 1,28 till 3,8 times.
The execution on 16 BB nodes with ExPBB is faster than 64 BB nodes without ExpBB
MPICH_MPIIO_HINTS="wrfi*:cb_nodes=128:striping_unit=4194304,
wrfo*:cb_nodes=256:striping_unit=4194304, wrfr*:cb_nodes=256:striping_unit=4194304"

ExPBB workflow

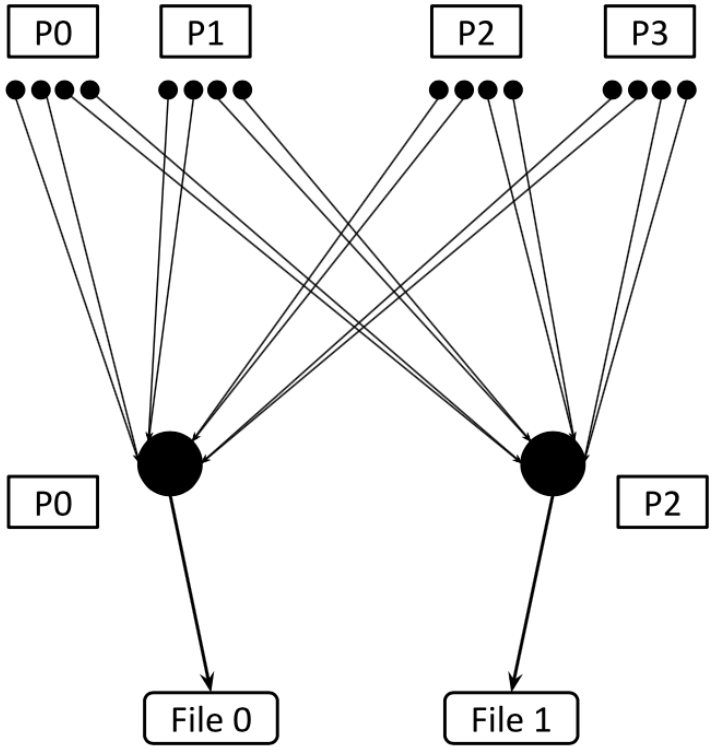


جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

PIDX

- PIDX is an efficient parallel I/O library that reads and writes multiresolution IDX data files
- It can provide high scalability up to 768k cores
- Successful integration with several simulation codes
  - KARFS (KAUST Adaptive Reacting Flow Solvers) on Shaheen II
  - Uintah with production runs on Mira
  - S3D
- Developed by University of Utah

PIDX structure

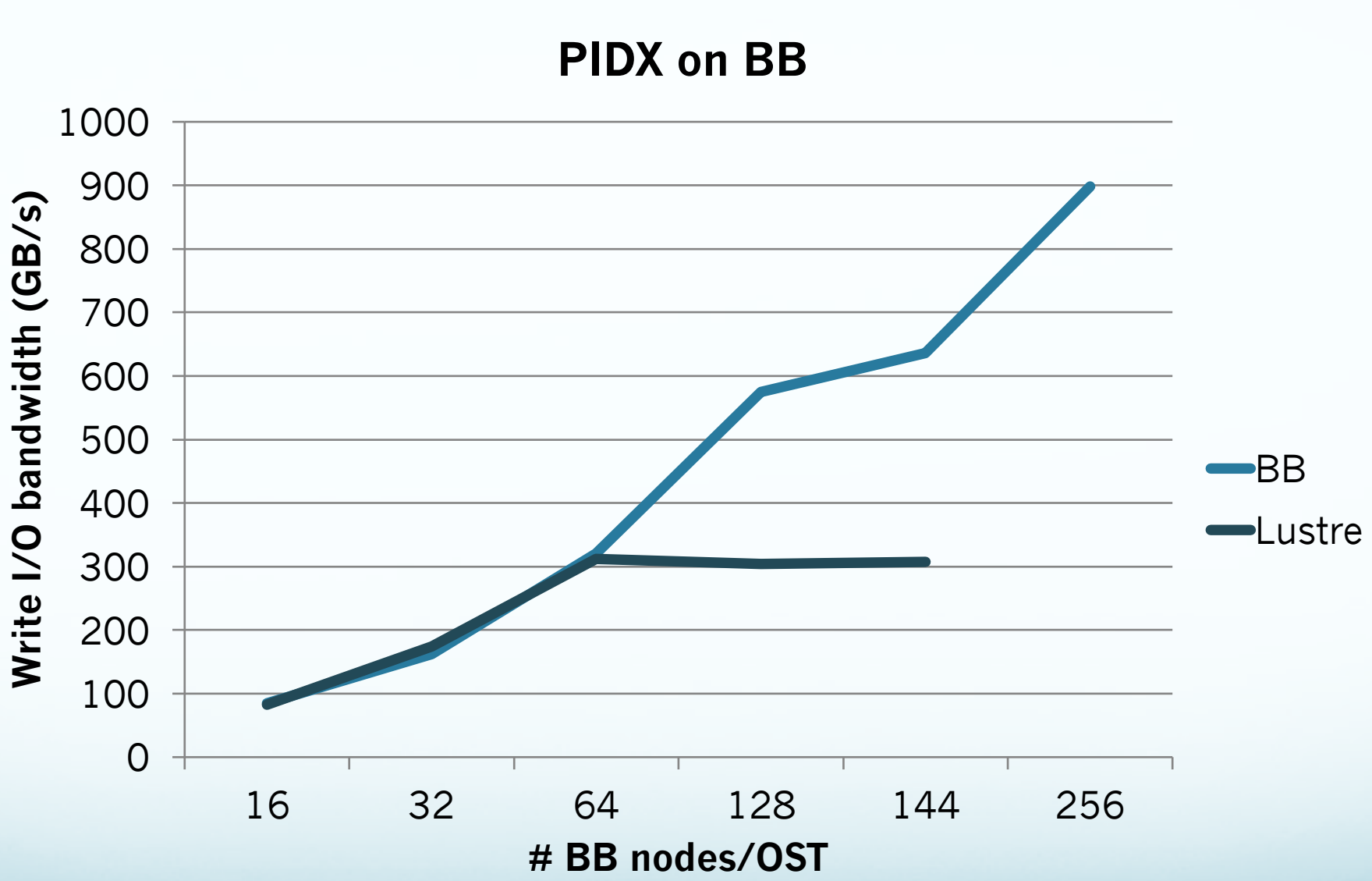


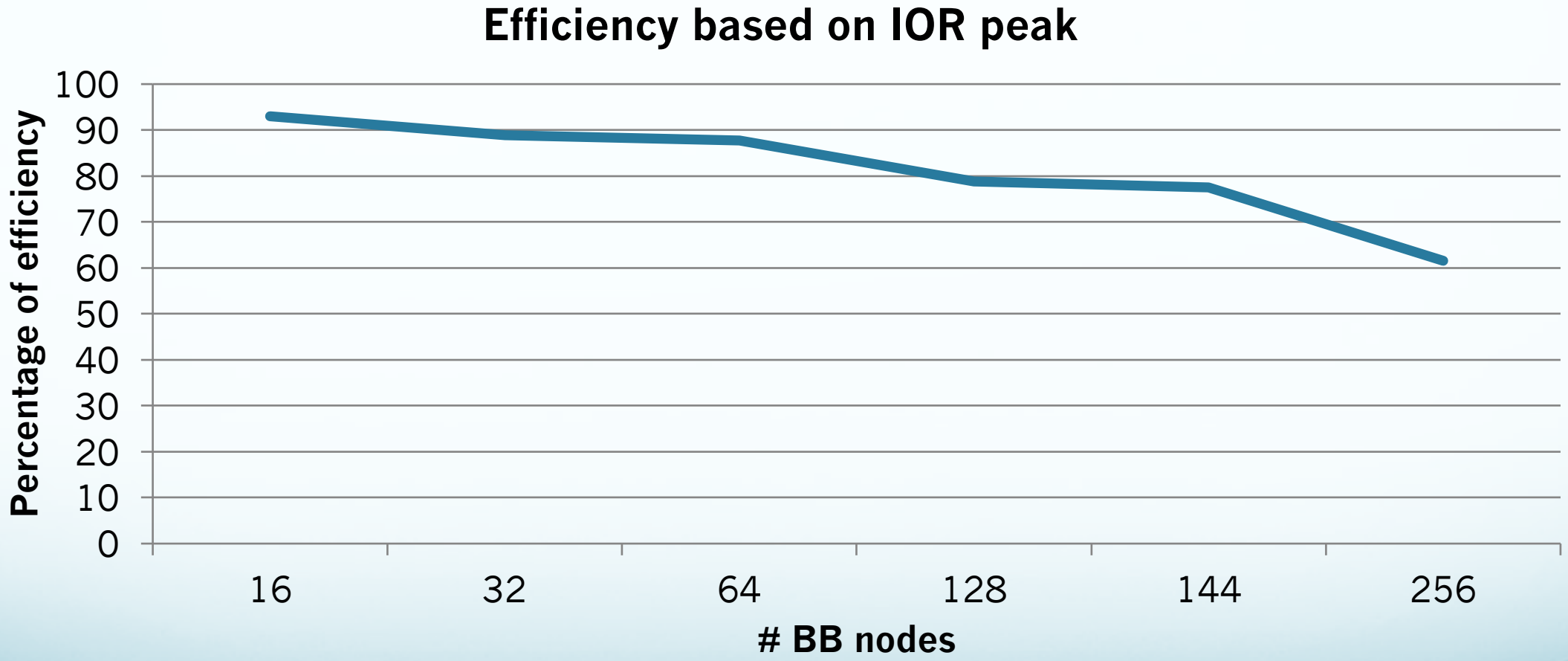
Synchronous reducing mode of communication using one-sided MPI

To balance the data movement, the aggregators are uniformly placed across the ranks.

File locking contention is reduced by having an aggregator to write to only one file

File count are tunable (based on the file system in use).


PIDX on BB
Write I/O bandwidth (GB/s)
1000
900
800
700
600
500
400
300
200
100
0
16
32
64
128
144
256
BB nodes/OST
BB
Lustre


Efficiency based on IOR peak
Percentage of efficiency
100
90
80
70
60
50
40
30
20
10
0
16
32
64
128
144
256
BB nodes

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Conclusions – Future work

- Many parameters need be investigated for the optimum performance
- Be patient, probably you will not achieve the best performance immediately
- Be careful to compile your application with the appropriate Cray MPICH
- CLE 6.0 will solve some BB issues of KAUST installation
- Always check the output of the application, used MPI version etc.
- Check MPICH_MPIIO_TIMERS with CRAY-MPICH 7.5.1
- Investigate Darshan XT
- Prepare next ExPBB version with online learning

JUN
20

HPC Storage Breakfast - Frankfurt (during ISC 2017)

by Seagate & Cray

Free

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Thank you!
Questions?

georgios.markomanolis@kaust.edu.sa