MISTRAL : bull environment

Application & Performance team

Pascale Girard
Cyril Mazauric

Introduction

Agenda

- **MPI environment**
  - bullxmpi, mxm and fca
- **feedback from the benchmark**
  - Call for tender rules
  - Optimization strategy
  - Two examples : EMAC and ICON
- **best-practices on setting the environment for bullxmpi**
  - How to find variables to tune your application ?
  - Examples

MPI environment

Architect of an Open World™

MPI environment

bullxmpi_mlx

- **you will find on mistral**
  - bullxMPI_mlx : openmpi 1.6.5 compiled with mellanox tools
  - bullxMPI_mlx_mt : idem + thread multiple support

- **How to use bullxmpi ?**

```
module load compiler intel
module load mxm/<version>
module load fca/<version>
module load bullxmpi_mlx/<version>
```

MPI environment

Mellanox tools

Fabric Collective Accelerator (FCA)

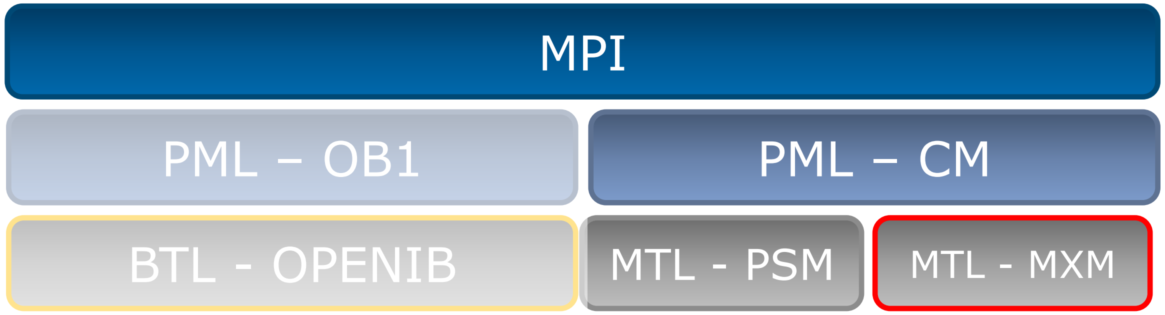
- FCA is a Mellanox MPI-integrated software package that utilizes CORE-Direct technology for implementing the MPI collectives communications. FCA can be used with all major commercial and open-source MPI solutions that exist and being used for high-performance applications. FCA with CORE-Direct technology accelerates the MPI collectives runtime, increases the CPU availability to the application and allows overlap of communications and computations with asynchronous collective operations.
  - Significantly reduce MPI collective operations runtime
  - Efficient collective communication flow optimized to job and topology
  - Increase CPU availability and efficiency
  - Eliminate congestion caused by collective traffic
  - No need for any additional hardware to install or manage
  - No space/power/cooling penalty
  - Seamless integration with MPI libraries and job schedulers
  - Future proof with embedded support for MPI-3

Messaging Accelerator (MXM)

- Mellanox Messaging Accelerator (MXM) provides enhancements to parallel communication libraries by fully utilizing the underlying networking infrastructure provided by Mellanox HCA/switch hardware. This includes a variety of enhancements that take advantage of Mellanox networking hardware including:
  - Multiple transport support including RC, XRC and UD
  - Proper management of HCA resources and memory structures
  - Efficient memory registration
  - One-sided communication semantics
  - Connection management
  - Receive side tag matching
  - Intra-node shared memory communication

MPI environment

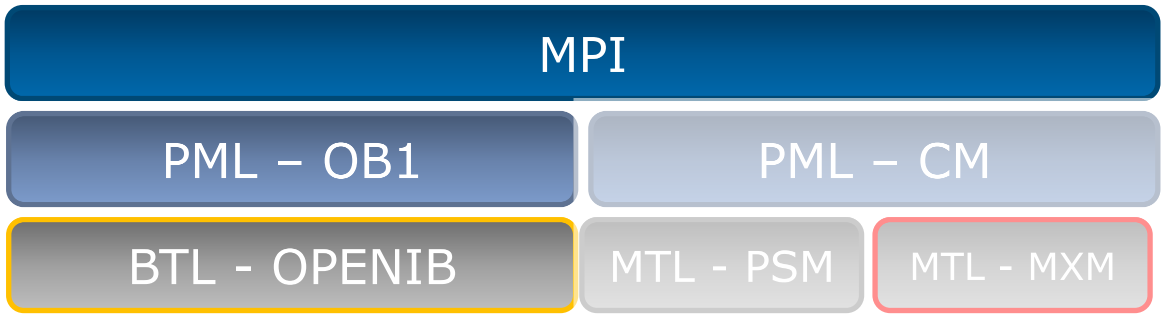
openmpi and mellanox tools ?



```
export OMPI_MCA_pml=cm
export OMPI_MCA_mtl=mxm
```

MPI environment

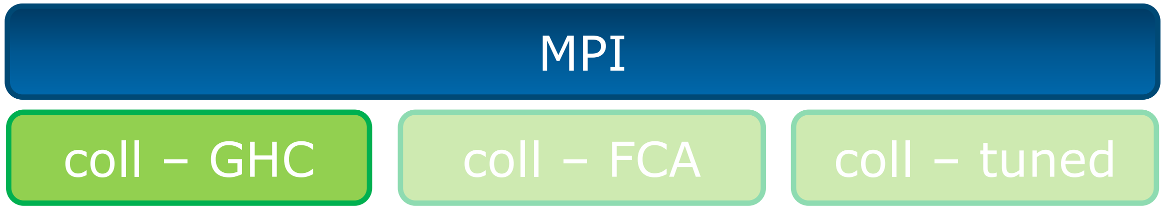
openmpi and mellanox tools ?



```
export OMPI_MCA_mtl=^mxm
export OMPI_MCA_pml=^cm
export OMPI_MCA_pml=ob1
```

MPI environment

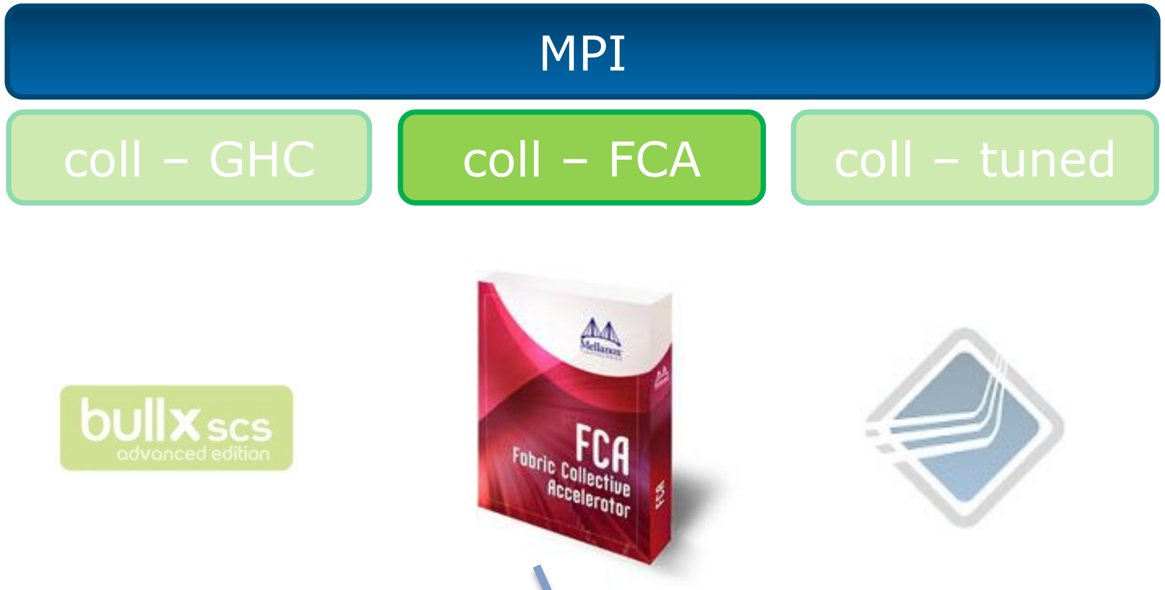
openmpi and mellanox tools ?



GHC has been developed to optimized collectives operations between islands when the pruning factor is important

MPI environment

openmpi and mellanox tools ?



FCA has been developed by Mellanox to optimized collectives operations : AllGather, AllReduce, Barrier, Bcast

MPI environment

openmpi and mellanox tools ?

bullxmpi

bullx scs advanced edition

MPI

coll – GHC | coll – FCA | coll – tuned

bullx scs advanced edition

tuned is the default openmpi module developped to optimized collectives operations

Atos Worldwide IT Partner

MXM and FCA impact on

DKRZ

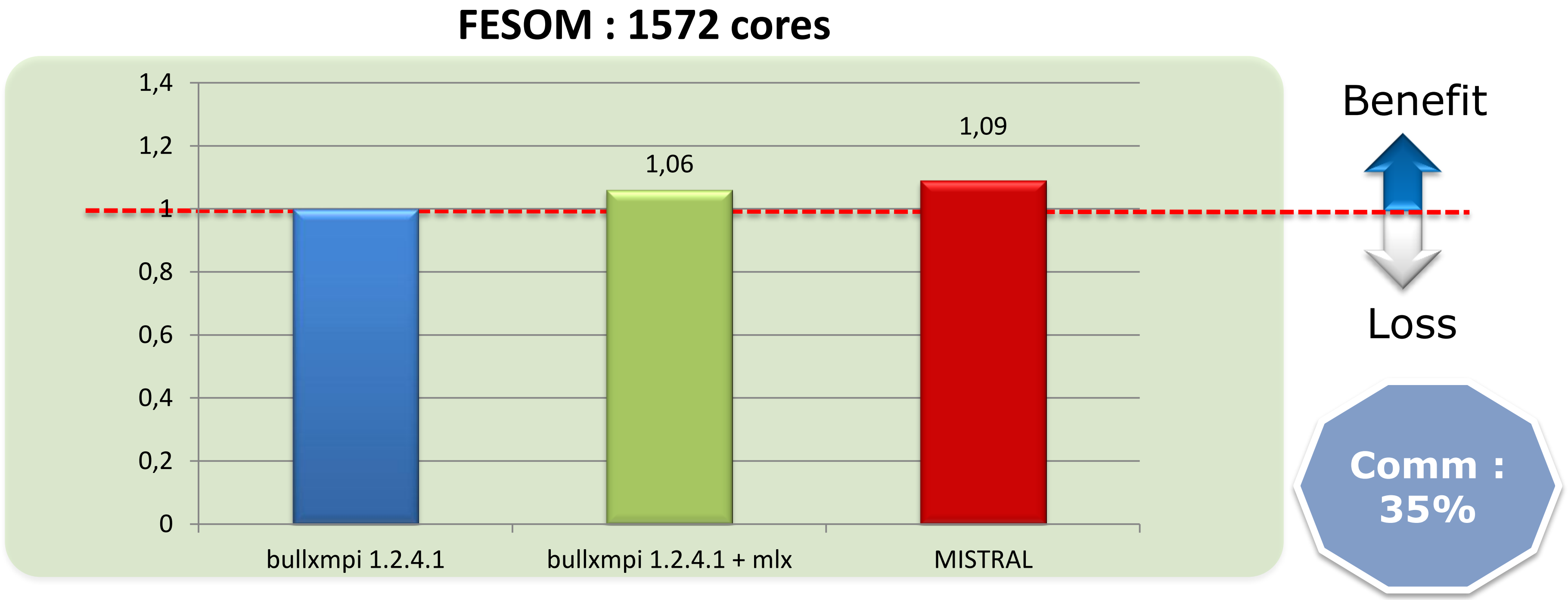
DEUTSCHES KLIMARECHENZENTRUM

Bull

Architect of an Open World™

MPI environment

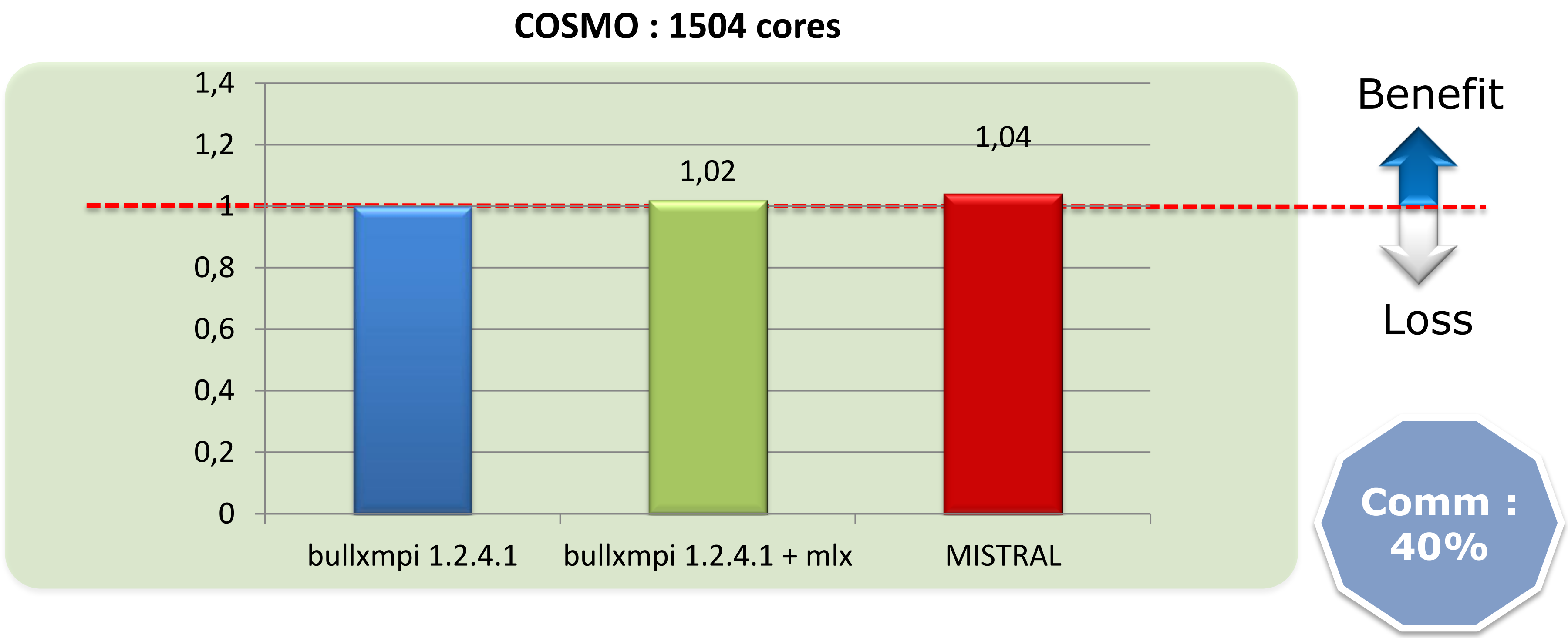
FESOM 1572 cores



▶ **Best performance measured with MXM and FCA**

MPI environment

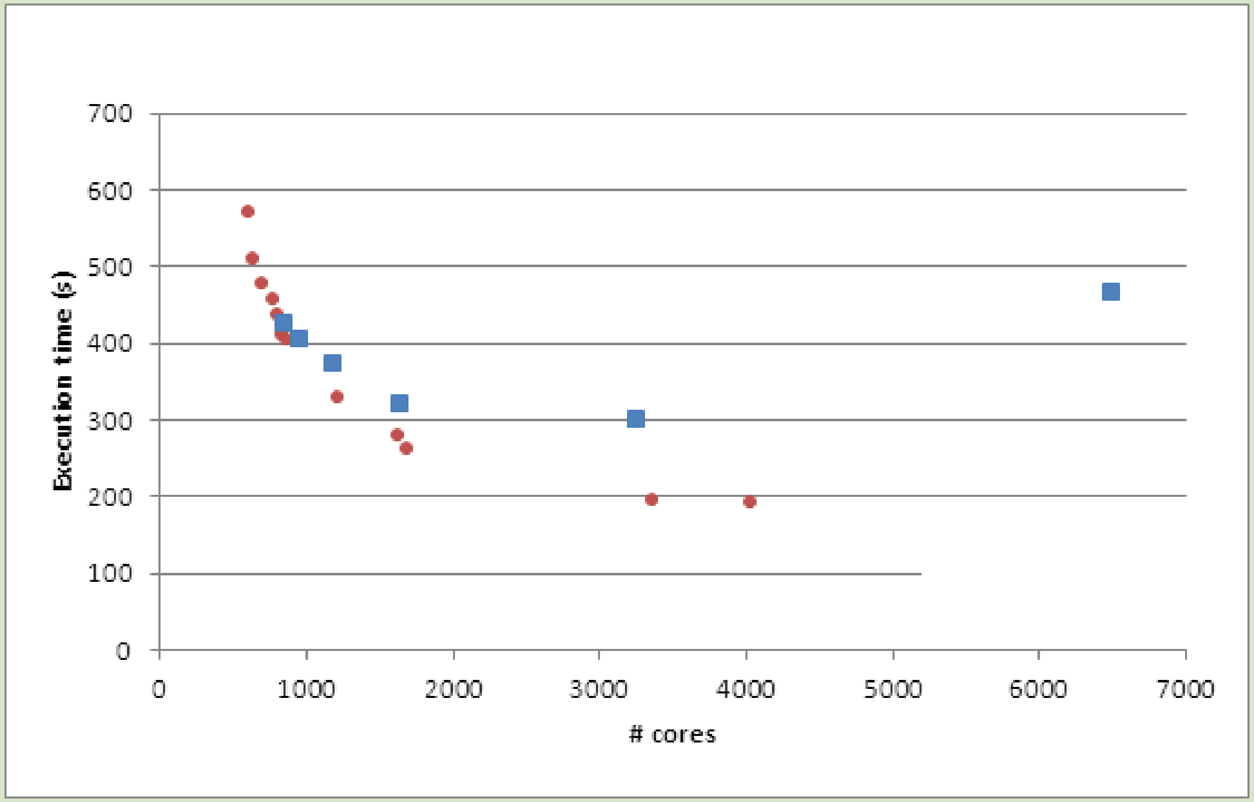
COSMO 1504 cores



▸ **Best performance measured with MXM (no FCA)**

MPI environment

COSMO

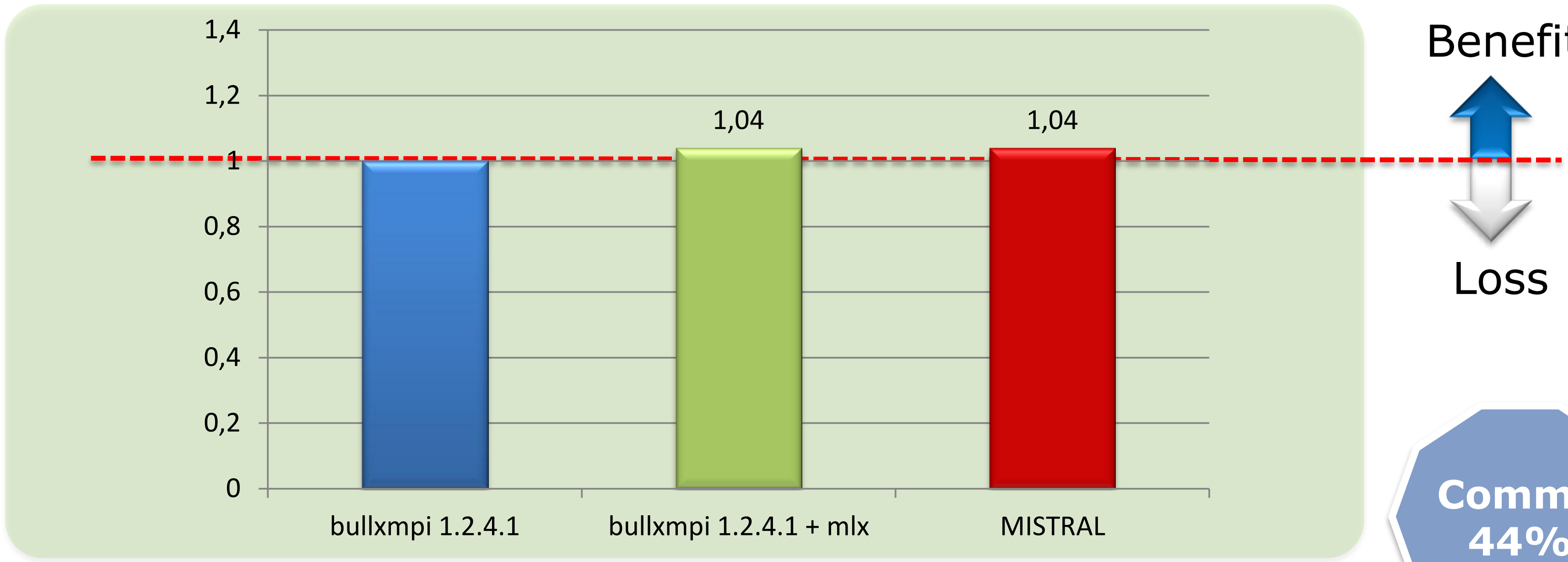


▶ **Best scalability with bullxmpi installed on MISTRAL**

MPI environment

EMAC

EMAC : 258 cores



Comm : 44%

▶ **Best performance measured with MXM and FCA**

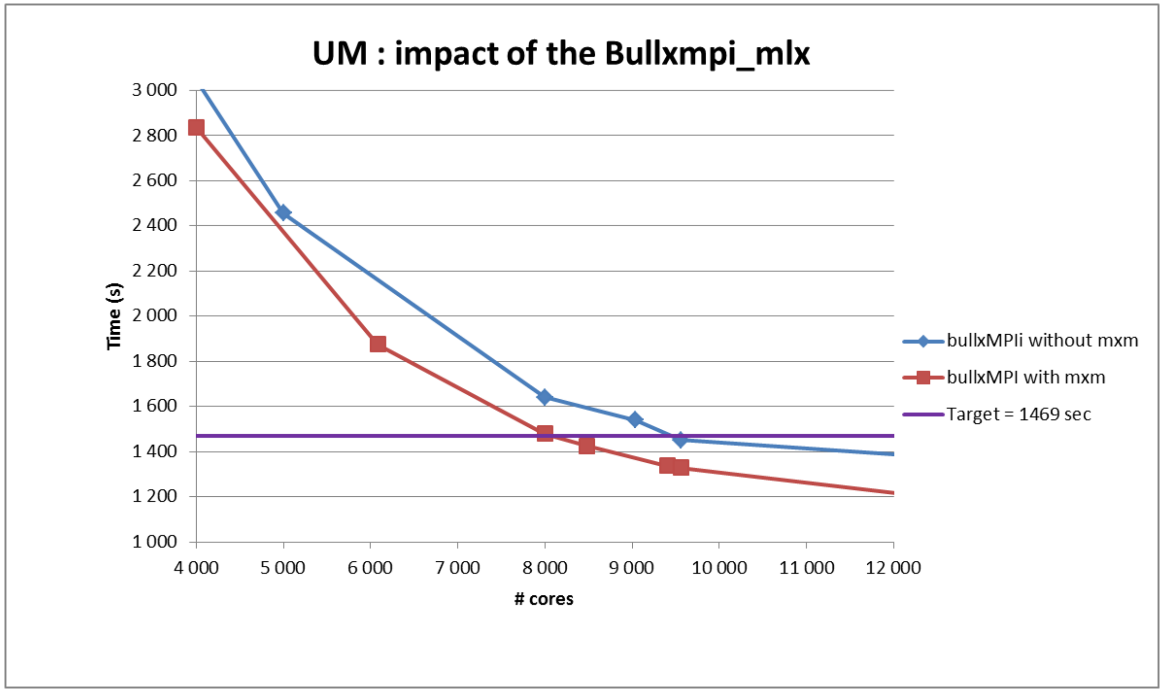
Another example

Met Office

Bull
Architect of an Open World™

MPI environment

UM : how many nodes should we used to reach 1469 sec ?



- **Without mxm = 500 nodes are necessary to decrease the time execution down to 1469 sec.**

- **With   mxm = 412 nodes !**

feedback from benchmark

Architect of an Open World™

Feedback from the benchmark

Performance optimization strategy

- **ASIS version**
  - Code analyse : input parameters tuning
  - Number of OpenMP threads and MPI tasks
  - Intel compiler options
  - Environment parameters : transparent huge page, drop cache, memory tuning, Turbo mode, hyperthreading…
  - Profiling MPI : MPI tuning using BullxMPI & MXM (and FCA), specific setting (Bcast algorithm, buffer size, MXM tuning…)
  - Use external libraries : memory alignment (aalloc…), BullxLib, MKL

- **Optimized version**
  - Identify hotspot (gprof, perftop, Allinea tools, Intel tools : vtune…)
  - Try to adapt the code to the Intel technology

  This work has been stopped when our commitments have been achieved… but this kind of work should be continued

Feedback from the benchmark

First example : EMAC

- **EMAC**
  - ECHAM5 / Messy Atmosphere Chemistry model
  - Code in Fortran, parallelized with MPI, using NETCDF and HDF5 libraries

- **Best environment setting**
  - Specific compiler options : -fp-model strict -O2 -xCORE-AVX2 –ftz
  - Use transparent huge page
  - Use memory tuning :
    - MALLOC_MMAP_MAX_=0
    - MALLOC_TRIM_THRESHOLD_=-1
  - MPI communications
    - 32% of the execution time is spent in MPI communications
    - 86% of communication time = MPI_Bcast
    - Use BullxMPI and MXM (no GHC, no FCA)
    - Bcast tuning : OMPI_MCA_coll_tuned_bcast_algorithm=2

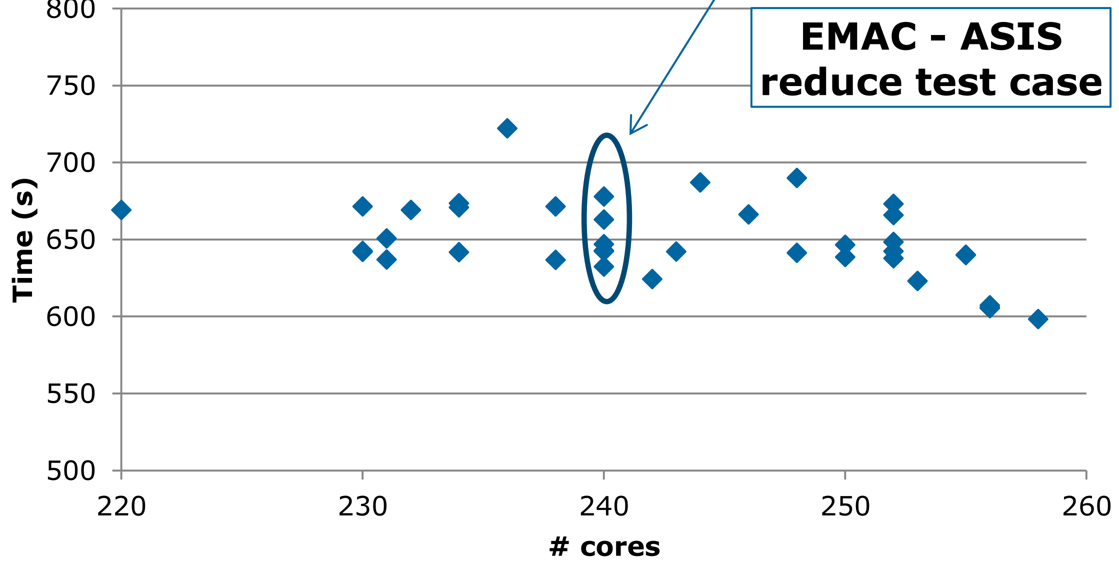
Feedback from the benchmark

First example : EMAC

- ▶ **Best environment found (suite)**
  - – Domain decomposition :
    number of MPI tasks = NP_X x NP_Y

240 = **16 x 15**
= 15 x16
= 12 x 20
= 10 x 24
= 8 x 30
= 6 x 40

EMAC - ASIS reduce test case



Feedback from the benchmark

First example : EMAC

- **BullxLIB library**
  - About 15% of the execution time spent in POWR8I4 function (when the application raise a real to integer exponent)
  - About 8% in EXP.L

This library is focused on the optimisation of

- X**(-4)
- X**3

Feedback from the benchmark

First example : EMAC

- **BullxLIB library**
  - About 15% of the execution time spent in POWR8I4 function (when the application raise a real to integer exponent)
  - About 8% in EXP.L

This library is focused on the optimisation of

- X**(-4)
- X**3

- **Code modification : Optimized version**
  - Loop reordering because in Fortran arrays are stored in column-major order.
    This implies that the matrix A will be stored in memory as (a11,a21,a12,a22) (in contrast, in a row-major order like C language, the order would be a11, a12, a21, a22).
    Thus, accessing the elements of A column-wise is most efficient.
  - Consolidation of two loops (to increase the data reuse)
  - Decrease the number of division and multiplication by factorisation

Feedback from the benchmark

First example : EMAC

- **ASIS**
  - Target time = 14 000 s
  - Achieved using
    - 11 nodes without Turbo mode (258 cores)
    - or 10 nodes with Turbo mode (240 cores)

- **Optimized**
  - Target time = 14 000 s
  - Achieved using 8 nodes with Turbo mode (176 cores)

Gain on OPTIM = 27%

Feedback from the benchmark

Second example : ICON

- **ICON**
  - ICOsahedral Non-hydrostatic model
  - Code in Fortran, parallelized with MPI and OpenMP, using NETCDF and HDF5 libraries
  - 2 test cases : APE (target time = 600s) and LAM (16 000 cores)

- **Best environment setting**
  - Specific compiler options :
    -ip -ansi-alias –pad -fast-transcendentals -align array64byte -xCORE-AVX2
  - Use transparent huge page
  - Zone reclaim
  - Use memory tuning :
    - MALLOC_MMAP_MAX_=0
    - MALLOC_TRIM_THRESHOLD_=-1
  - Use record buffer for I/O
    - FORT_BUFFERED=true
    - decfort_dump_flag=true
  - MPI communications
    - Use BullxMPI and MXM (no GHC, no FCA)

Feedback from the benchmark

Second example : ICON

- **Best environment setting (suite)**
  - OpenMP tuning
    - OMP_STACKSIZE=64M
    - KMP_BLOCKTIME=1
    - OMP_WAIT_POLICY=PASSIVE
  - OpenMP threads / MPI tasks and hyperthreading

Test case	#nodes	Execution time (s)							
		HT ON					HT OFF		
		24 MPI x 2 OMP	12 MPI x 4 OMP	8 MPI x 6 OMP	6 MPI x 8 OMP	4 MPI x 12 OMP	24 MPI x 1 OMP	12 MPI x 2 OMP	6 MPI x 4 OMP
APE	141	834	831	821	837	848		882	883
	190	630	615	628	625	637			
	192	610	604	612	610	631			
	194	617	601	606	603	628			
	282	443	425	426	433	441			
	388	391	327	328	325	336	455	330	328
	564	390	248	245	238	241	402	241	237
	667	121	111	111	111		133	120	104
LAM	776	392	235	199	194	209	400	217	188

Feedback from the benchmark

Second example : ICON

- **Code modification : ASIS version**
  - Adding guideline to force the vectorization on loops (OMP directives “omp simd”)
  - Adding guideline to align data

- **Code modification : Optimized version**
  - The main bottleneck is related to the memory : data structure is critical. We were not able to modify it…
  - Few modifications :
    - Remove some not needed if statements
    - Replace some if statements by merge
    - Split some loops (to improve the data reuse)

Feedback from the benchmark

Second example : ICON

- **ASIS (APE)**
  - Target time = 600 s
  - Achieved using 198 nodes with Turbo mode & hyperthreading (8 MPI x 4 OMP per node)

- **Optimized (APE)**
  - Target time = 600 s
  - Achieved using 194 nodes with Turbo mode & hyperthreading (8 MPI x 4 OMP per node)

Gain on OPTIM = 2%

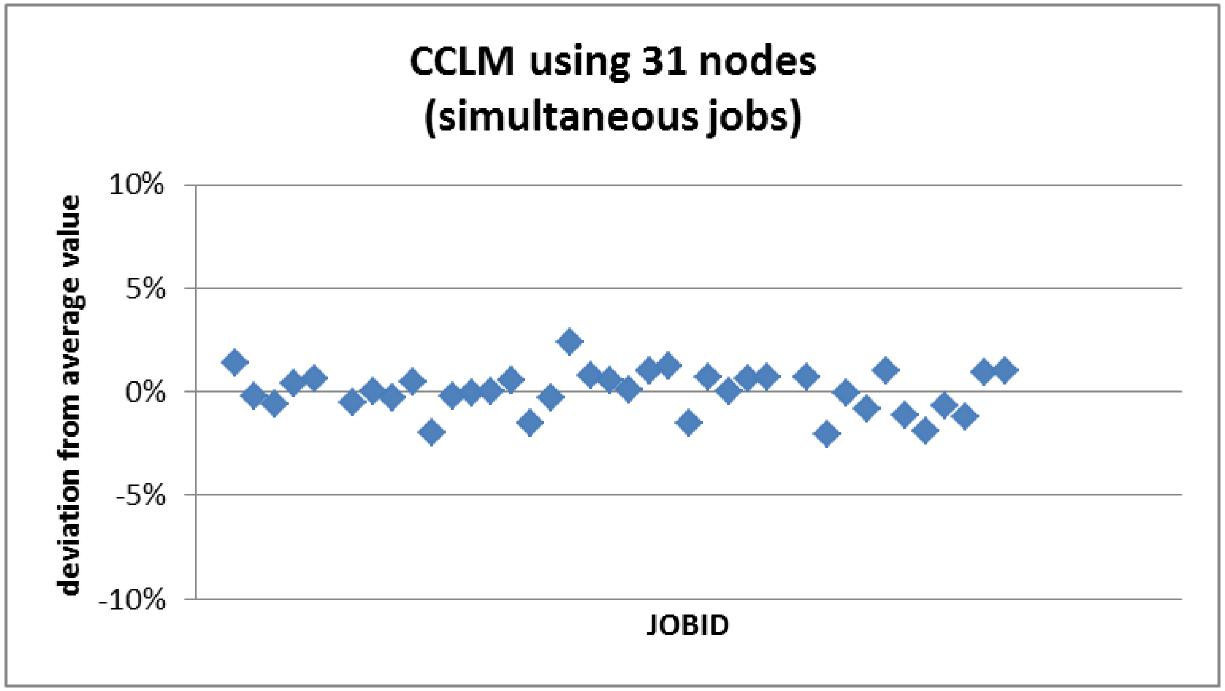
best-practices on setting the environment for bullxmpi

Architect of an Open World™

Why interconnect is a fat tree ?

Cosmo

- cosmo on 31 nodes is our reference. With 40 occurrences mistral is full (almost)



- Deviation is lower than 5%

how to find variables to tune your application ?

bullxmpi

- **Once environment is correctly loaded**

```
module load compiler intel
module load mxm/<version>
module load fca/<version>
module load bullxmpi_mlx/<version>
```

- **MXM (messaging accelerator)**

```
mxm_dump_config –t
```

- **BTL(openmpi default device)**

```
ompi_info –all
```

BTL

```
export OMPI_MCA_mtl=^mxm
export OMPI_MCA_pml=^cm
export OMPI_MCA_pml=ob1
```

how to find variables to tune your application ?

bullxmpi

- **FCA (Mellanox collectives accelerator)**

  ompi_info –all

FCA

```
export OMPI_MCA_coll=^ghc
export OMPI_MCA_coll_fca_priority=95
export OMPI_MCA_coll_fca_enable=1
```

- **tuned (openmpi default)**

  ompi_info –all -mca coll_tuned_use_dynamic_rules

tuned

```
export OMPI_MCA_coll=^ghc,fca
```

A MPI setting by default ?

bullxmpi

- **a good environment : bullxmpi with mxm**

```
module load compiler intel
module load mxm/<version>
module load fca/<version>
module load bullxmpi_mlx/<version>
export OMPI_MCA_pml=cm
export OMPI_MCA_mtl=mxm
export OMPI_MCA_coll=^ghc
export MXM_RDMA_PORTS=mlx5_0:1
```

mxm will improve the scalability

A MPI setting by default ?

bullxmpi

- **a good environment : bullxmpi with mxm**

```
module load compiler intel
module load mxm/<version>
module load fca/<version>
module load bullxmpi_mlx/<version>
export OMPI_MCA_pml=cm
export OMPI_MCA_mtl=mxm
export OMPI_MCA_coll=^ghc
export MXM_RDMA_PORTS=mlx5_0:1
```

FDR interconnect is a full fat tree without island. GHC is not efficient and should not be used

A MPI setting by default ?

bullxmpi

▸ **a good environment : bullxmpi with mxm**

```
module load compiler intel
module load mxm/<version>
module load fca/<version>
module load bullxmpi_mlx/<version>
export OMPI_MCA_pml=cm
export OMPI_MCA_mtl=mxm
export OMPI_MCA_coll=^ghc
export MXM_RDMA_PORTS=mlx5_0:1
```

MXM will use the correct infiniband device : mlx5

A MPI setting by default ?

bullxmpi

▶ **a good environment : bullxmpi with mxm**

```
module load compiler intel
module load mxm/<version>
module load fca/<version>
module load bullxmpi_mlx/<version>
export OMPI_MCA_pml=cm
export OMPI_MCA_mtl=mxm
export OMPI_MCA_coll=^ghc
export MXM_RDMA_PORTS=mlx5_0:1
```

you will use mxm and tuned for collectives. FCA could be used with the correct setting

How to find a good MPI tuning ?

Profiling

- **You need to use a MPI profiler which will show you which MPI functions are used.**

```
MPI Communications Statistics:
==============================

Total Parallel time : 7281.543846 s
Total Communication time : 551.369430 s
Ratio :   8 %

Accumulated data of all processes:
==================================
                         Min [R]           Max [R]            Average       Stand dev.

Parallel time:          910.185775 [7]     910.239525 [0]        910.192981        0.017595 (  0.00 %)
Communication time: 21.771790 [0]     150.005858 [7]         68.921179       58.516123 ( 84.90 %)
……
MPI_Allreduce: 303.233854 s ( 55.00 %)
          number           16730 [0]          16730 [0]          16730.00
          time (s)      4.767301 [0]      106.838756 [6]          37.904232
          size (b)         135232 [0]         135232 [0]          135232.00
```

How to find a good MPI tuning ?

Example : MPI_Wait

- **The main MPI funtion is MPI_Wait**
  - Is it logical ?
    - nothing to do ☺
  - Bad synchronization ?
    - change the mpi tasks binding to improve the performance
    - srun –m cyclic (round-robin distribution inter nodes)
    - srun –m cyclic:cyclic (round-robin inter and intra nodes)
  - messages are waiting because the messages sizes are too small ?
    - try to pack your messages to take advantage of the interconnect bandwidth

How to find a good MPI tuning ?

Example : MPI_Barrier

- **The main MPI funtion is MPI_Barrier**
  - Are they really necessary ?
    - Usually, MPI_Barrier are used to clean the output files ☺, removing all barrier could improve the performance or highlight which MPI functions are hidden.

      you can create a library which replace the MPI_Barrier call by nothing

      int MPI_Barrier(MPI_Comm comm) { return 0; }

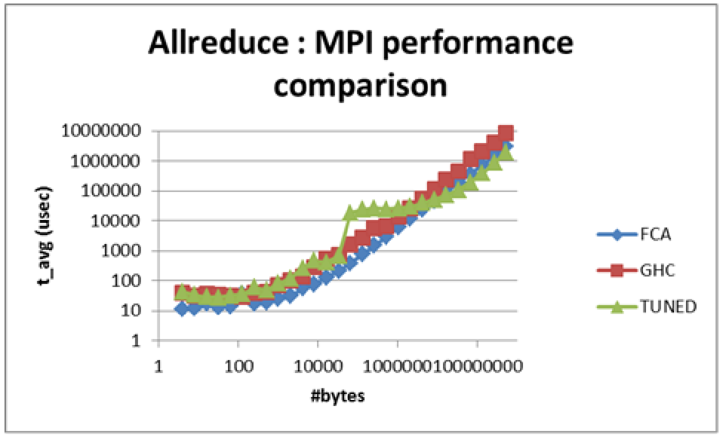
How to find a good MPI tuning ?

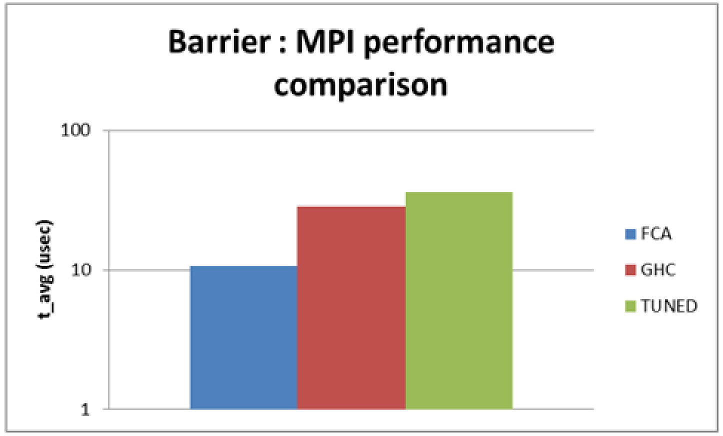
Example : MPI_allreduce

▶ **The main MPI funtion is MPI_allReduce (or another collectives operation)**

- fca
  - check with “ompi_info --all” if the variable coll_fca_enable_allreduce is set to 1

```
MCA coll: parameter "coll_fca_enable_allreduce" (current value: "1", data source: default, level: 9 dev/all, type: int)
        [1|0|] Enable/Disable FCA Allreduce support
```





How to find a good MPI tuning ?

Example : MPI_allreduce

- But as we know IMB is not the real life !

- tuned
  - test all tuned algorithm

MCA coll: informational "coll_tuned_allreduce_algorithm_count" (current value: "**5**….)
MCA coll: parameter "**coll_tuned_allreduce_algorithm**"
Which allreduce algorithm is used. Can be locked down to any of: 0 ignore, 1 basic linear, 2 nonoverlapping (tuned reduce + tuned bcast), 3 recursive doubling, 4 ring, 5 segmented ring

```
export OMPI_MCA_coll_tuned_use_dynamic_rules=1
export OMPI_MCA_ coll_tuned_allreduce_algorithm={1 2 3 4 or 5}
```

Thanks

For more information please contact: Pascale Girard or Cyril Mazauric

Pascale.Girard@atos.net
Cyril.Mazauric@atos.net