Coarrays in GNU Fortran

Alessandro Fanfarillo

fanfarillo@ing.uniroma2.it

June 24th, 2014

Introduction

Coarray Fortran (also known as CAF) is a syntactic extension of Fortran 95/2003 which has been included in the Fortran 2008 standard.

The main goal is to allow Fortran users to realize parallel programs without the burden of explicitly invoke communication functions or directives (MPI, OpenMP).

The secondary goal is to express parallelism in a "platform-agnostic" way (no explicit shared or distributed paradigm).

Coarrays are based on the Partitioned Global Address Space model (PGAS).

Compilers which support Coarrays:

- Cray Compiler (Gold standard - Commercial)
- Intel Compiler (Commercial)
- Rice Compiler (Free - Rice University)
- OpenUH (Free - University of Houston)
- G95 (Coarray support not totally free - Not up to date)

PGAS Languages

The PGAS model assumes a global memory address space that is logically partitioned and a portion of it is local to each process or thread.

It means that a process can directly access a memory portion owned by another process.

The model attempts to combine the advantages of a SPMD programming style for distributed memory systems (as employed by MPI) with the data referencing semantics of shared memory systems.

- Coarray Fortran.
- UPC (upc.gwu.edu).
- Titanium (titanium.cs.berkeley.edu).
- Chapel (Cray).
- X10 (IBM).

Coarray concepts

- A program is treated as if it were replicated at the start of execution, each replication is called an image.
- Each image executes asynchronously.
- An image has an image index, that is a number between one and the number of images, inclusive.
- A Coarray is indicated by trailing [ ].
- A Coarray could be a scalar or array, static or dynamic, and of intrinsic or derived type.
- A data object without trailing [ ] is local.
- Explicit synchronization statements are used to maintain program correctness.

Coarray example

```
real, dimension(10), codimension[*] :: x, y
integer :: num_img, me

num_img = num_images()
me = this_image()

! Some code here

x(2) = x(3)[7] ! get value from image 7

x(6)[4] = x(1) ! put value on image 4

x(:)[2] = y(:) ! put array on image 2

sync all

x(1:10:2) = y(1:10:2)[4] ! strided get from image 4

sync images (*)
```

Coarray support

- The most mature, efficient and complete implementation is provided by Cray.
- Cray runs on proprietary architecture.
- Intel provides a CAF implementation but it works only on Linux and Windows.
- Intel CAF has two modes: shared and distributed. For the distributed mode the Intel Cluster Toolkit is required.

GNU Fortran Libcaf

GFortran uses an external library to support Coarrays (libcaf).

Currently there are three libcaf implementations:

- MPI Based
- GASNet Based
- ARMCI Based (not updated)

The idea is to provide the MPI version as default and the GASNet/ARMCI as “expert version”.

GASNet

GASNet (Global Address Space Networking) provided by UC Berkeley.

- Efficient Remote Memory Access operations.
- Native network communication interfaces.
- Useful features like Active Messages and Strided Transfers.

Main issue: GASNet requires an explicit declaration of the total amount of remote memory to use (allocatable coarrays?).

Secondary issue: GASNet requires a non-trivial configuration during the installation and before the usage.

The GASNet version of Libcaf has not been deeply studied yet.

MPI Version - Current status

Supported Features:

- Coarray scalar and array transfers (efficient strided transfers)
- Synchronization
- Collectives
- Vector subscripts

Unsupported Features:

- Derived type coarrays with non-coarray allocatable components
- Atomics
- Critical
- Error handling

Hardware

- Eurora: Linux Cluster, 16 cores per node, Infiniband QDR QLogic (CINECA).
- PLX: IBM Dataplex, 12 cores per node, Infiniband QDR QLogic (CINECA).
- Yellowstone/Caldera: IBM Dataplex, 16 cores per node, Infiniband Mellanox (NCAR).
- Janus: Dell, 12 cores per node, Infiniband Mellanox (CU-Boulder).
- Hopper: Cray XE6, 24 cores per node, 3-D Torus Cray Gemini (NERSC).
- Edison: Cray XC30, 24 cores per node, Cray Aries (NERSC).

Only Yellowstone and Hopper used for this presentation.

Software

On Hopper:

- Cray: Cray (CCE) 8.2.1
- GFortran: GCC-4.10 experimental (gcc branch) + Mpich/6.0.1

On Yellowstone:

- Intel: Intel 14.0.2 + IntelMPI 4.0.3
- GFortran: GCC-4.10 experimental (gcc branch) + MPICH IBM opt. Mellanox IB

For Intel CAF, every coarray program runs in CAF Distributed Mode.

Test Suite Description

- CAF Psnap (Dan Nagle)
  - It is a network noise analyzer. It can be useful for a statistical study of the communication between processes.
- EPCC CAF Micro-benchmark suite (Edinburgh University)
  - It measures a set of basic parallel operations (get, put, strided get, strided put, sync, halo exchange).
- Burgers Solver (Damian Rouson)
  - It has nice scaling properties: it has a 87% weak scaling efficiency on 16384 cores and linear scaling (sometimes super-linear).
- CAF Himeno (Prof. Ryutaro Himeno - Bill Long - Dan Nagle)
  - It is a 3-D Poisson relaxation.
- Distributed Transpose (Bob Rogallo).
  - It is a 3-D Distributed Transpose.
- Dense matrix-vector multiplication (MxV).

Tests execution

- Every test has been run on Yellowstone/Caldera and Hopper.
- On Yellowstone we are able to run only Intel and GFortran.
- On Hopper we are able to run only GFortran and Cray.
- GFortran runs with whatever MPI implementation is available.
- On GFortran we used only the -Ofast flag.
- On Intel we used the following flags: -Ofast -coarray -switch no_launch
- On Cray we used the -O3 flag. We also loaded craype-hugepages2M and set XT_SYMMETRIC_HEAP_SIZE properly.

CAF Psnap - Yellowstone 32 cores

- This test shows the performance of the communication layer under the Coarrays interface.
- Psnap sends several times a single integer between "peer" processes (0-16, 1-17, ..).
- Using Intel CAF (and thus IntelMPI) transferring a single integer takes on average 1.018 msecs.
- Using GFortran/MPI (and thus IBM MPI) transferring a single integer takes on average 1.442 msecs.
- The time distribution is different (Gamma for Intel, Normal for IBM MPI).

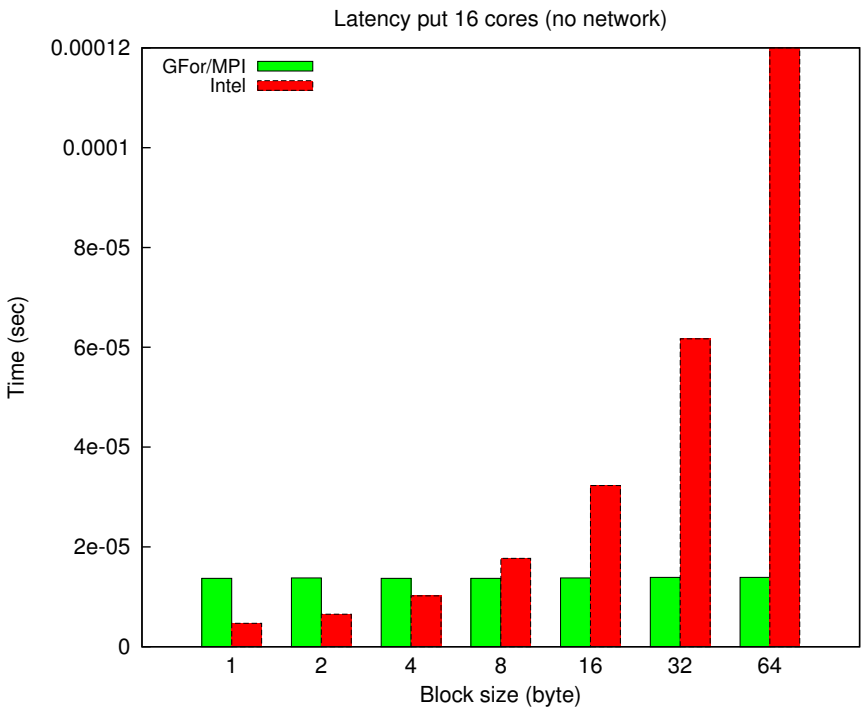
CAF Psnap - Hopper 48 cores

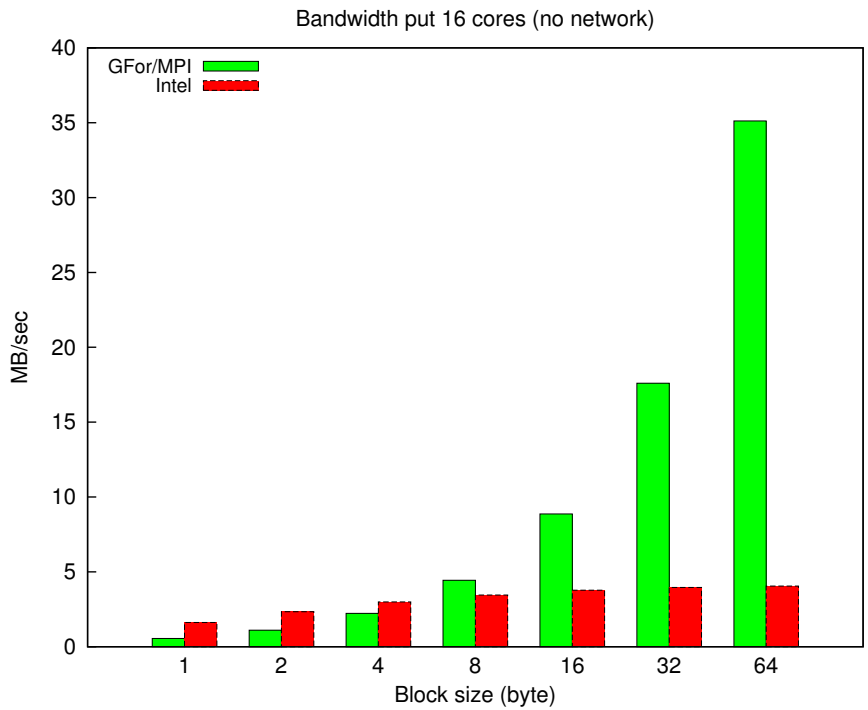
- Using Cray transferring a single integer takes on average 1.021 msecs.
- Using GFortran/MPI transferring a single integer takes on average 2.996 msecs.
- Both distributions look like heavy-tail.

EPCC CAF Micro-benchmark suite

- Single point-to-point: image 1 interacts with image n.
  - Put, Get, Strided Put, Strided Get.
- Multiple point-to-point: image i interacts with image i+n/2.
  - Put, Get, Strided Put, Strided Get.
- Synchronization: several pattern of synchronization. (Not shown)
  - Sync all, sync images.
- Halo swap: exchange of the six external faces of a 3D array. (Not shown)

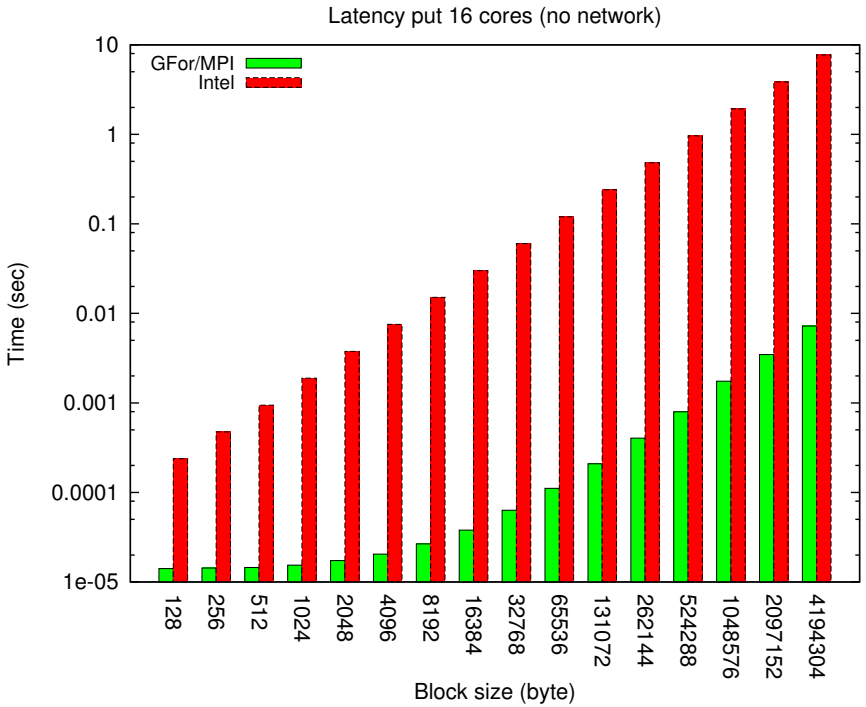
Latency/Bw Single Put 16 cores Yellowstone (Small)

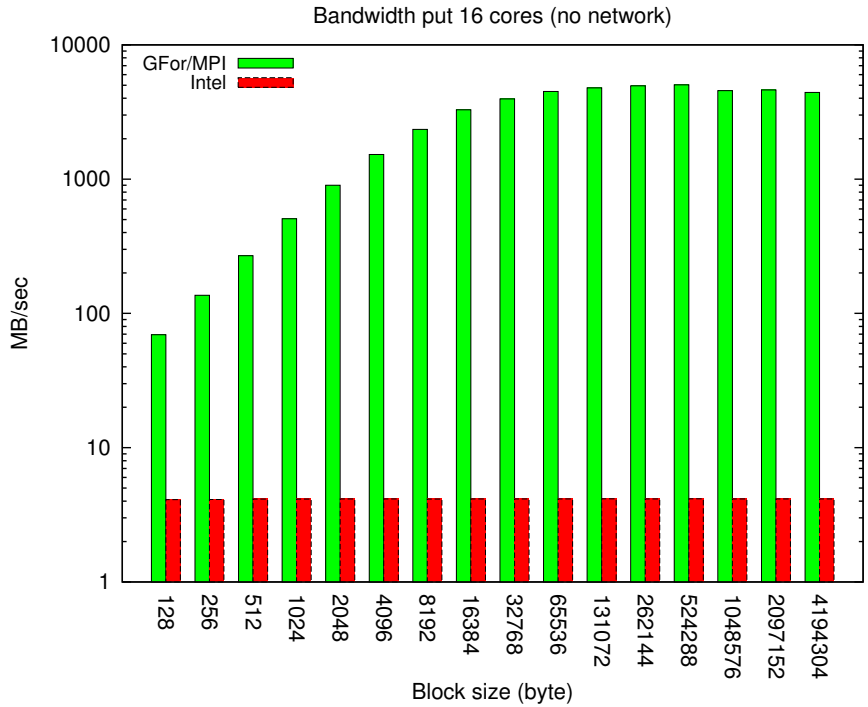




Latency: lower is better - Bandwidth: higher is better

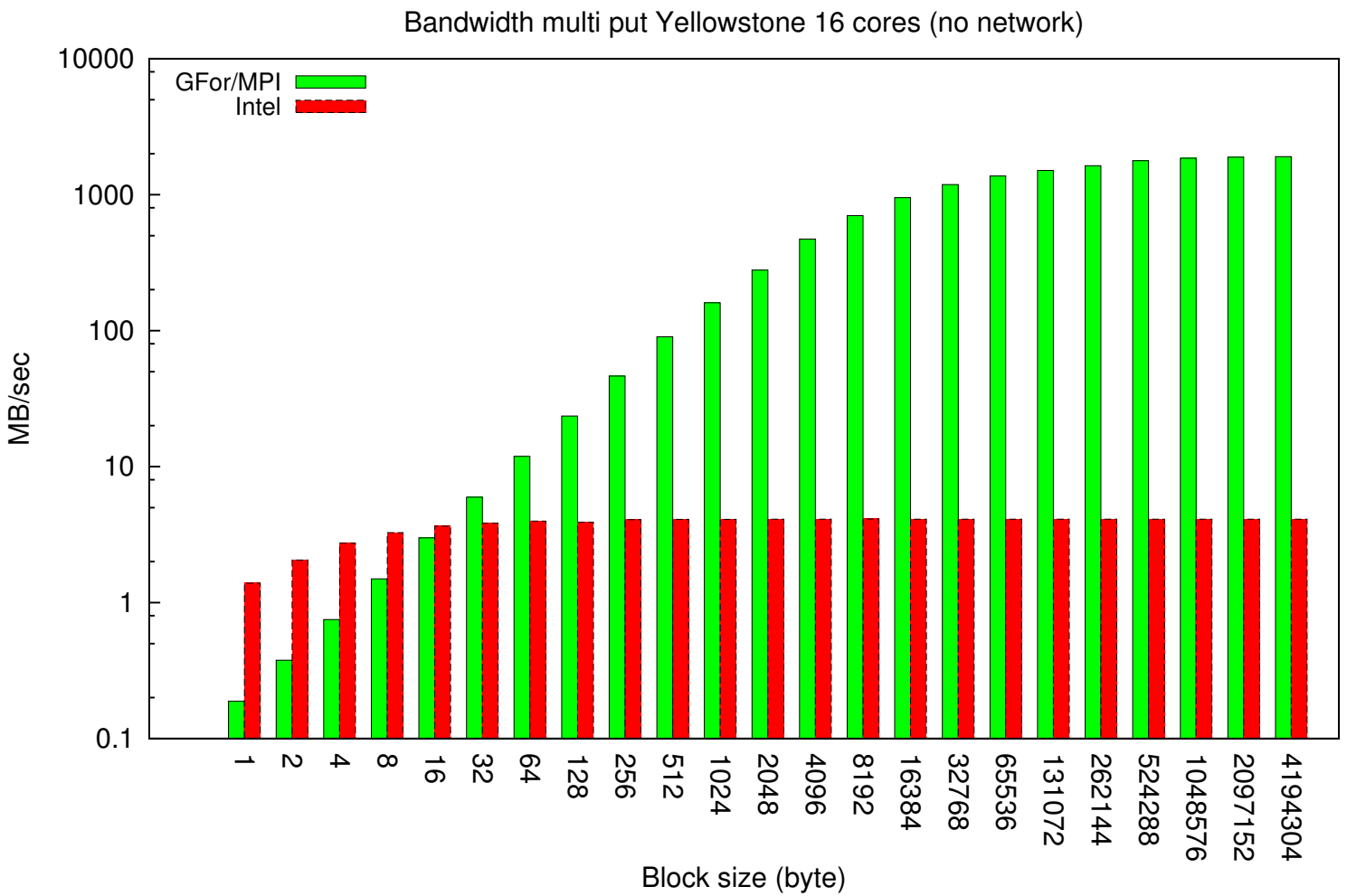
Latency/Bw Single Put 16 cores Yellowstone (Big)



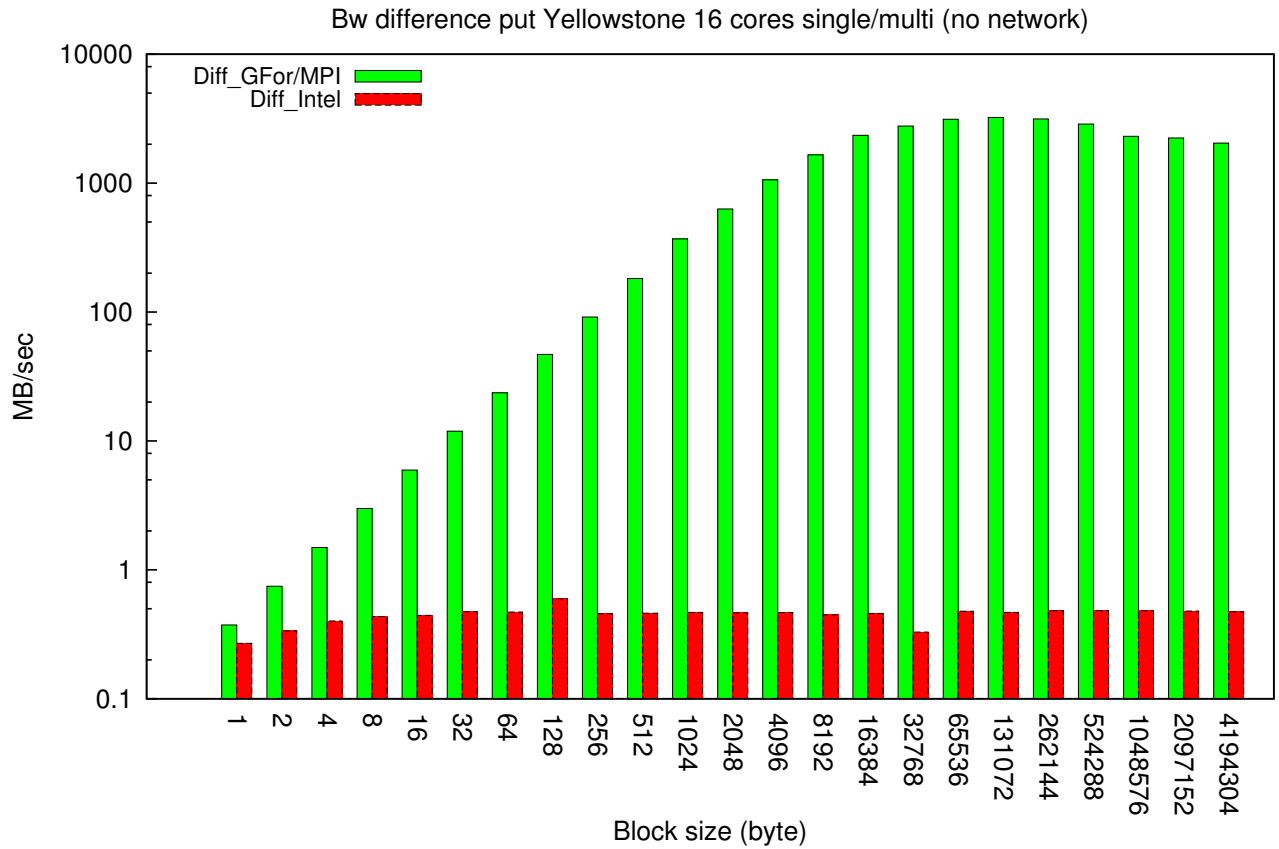


Latency: lower is better - Bandwidth: higher is better

Bw Multi Put 16 cores Yellowstone

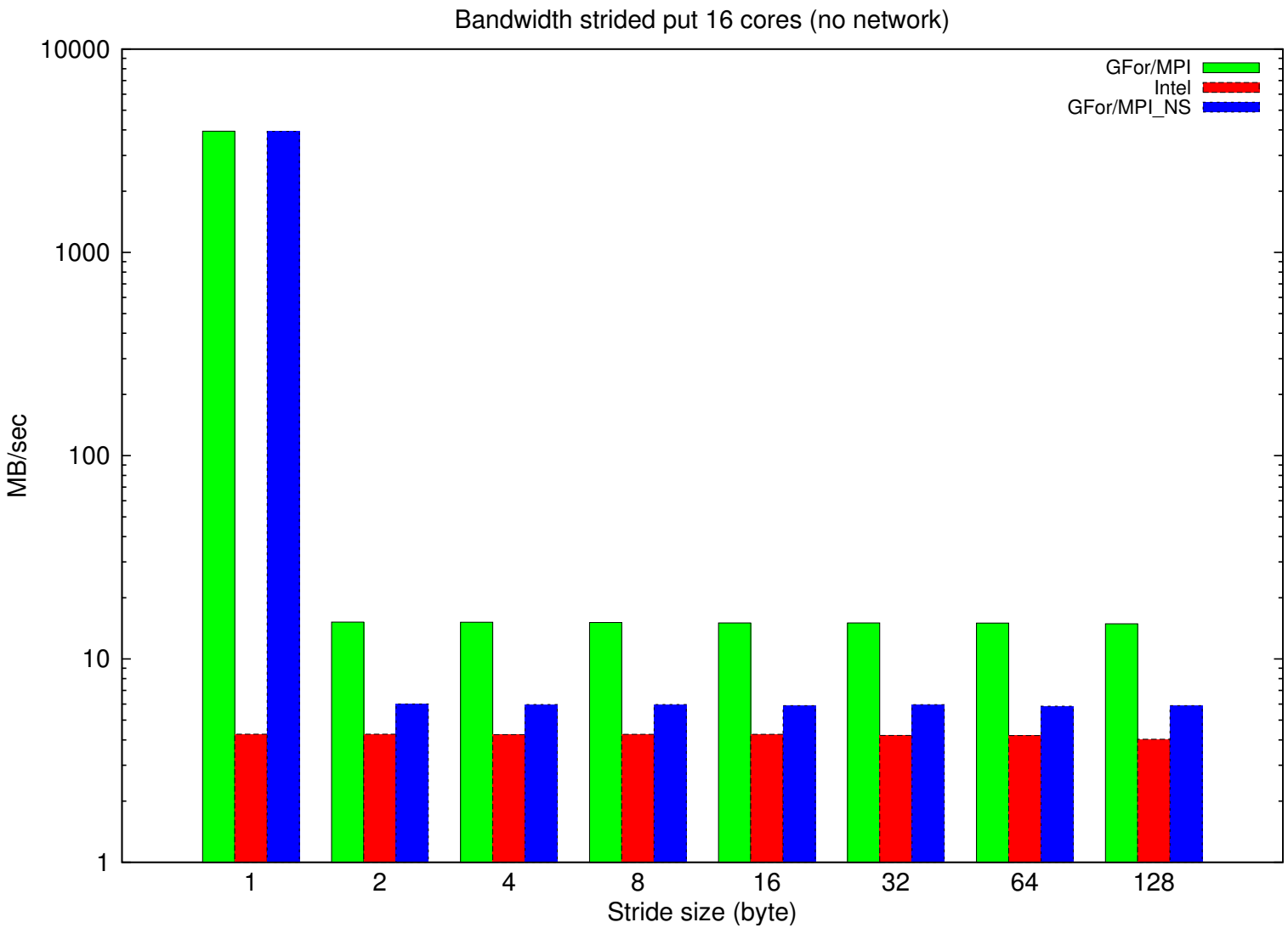


Bw Difference Single/Multi Put 16 cores Yellowstone

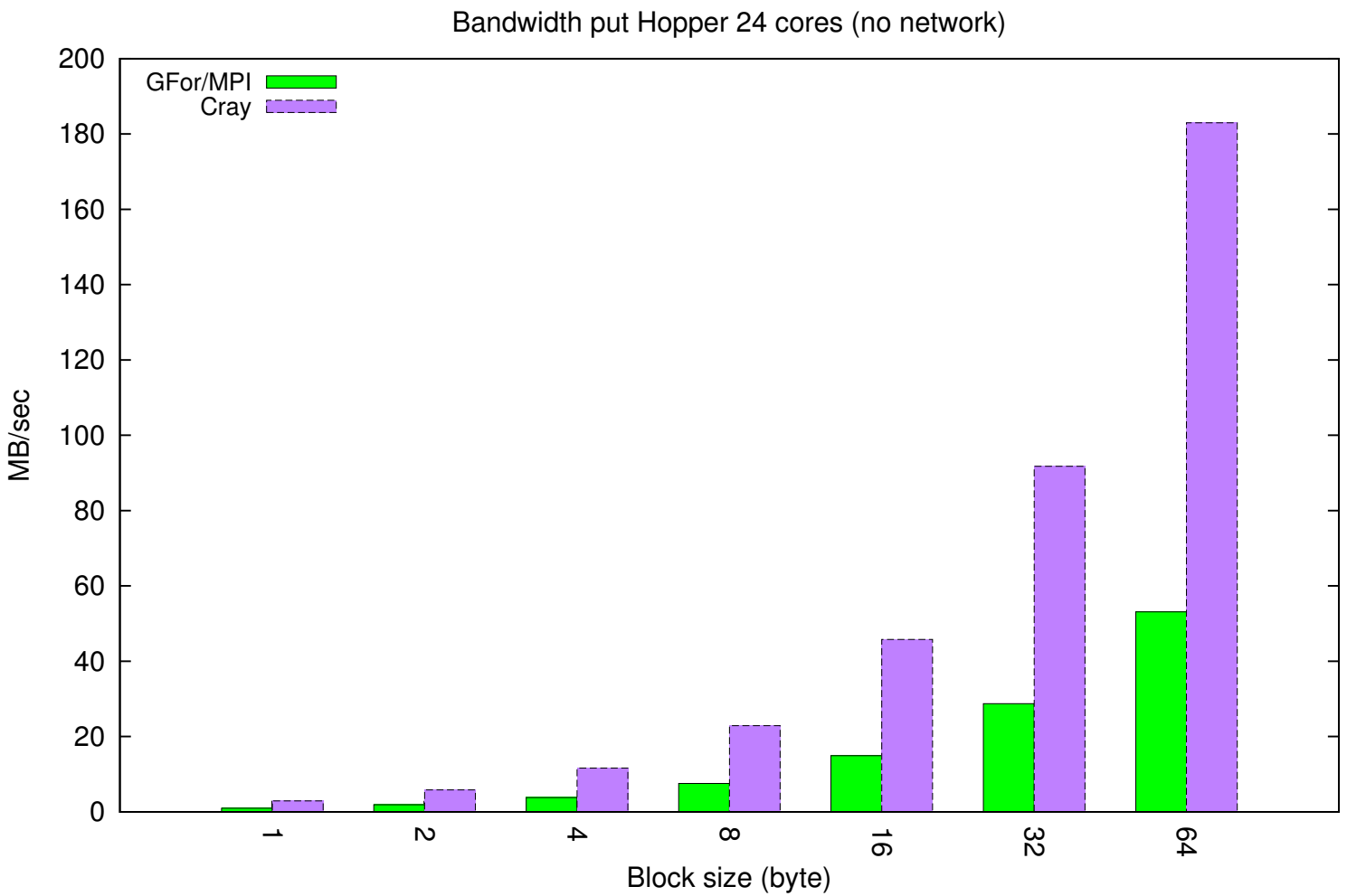


A constant trend means that the network contention does not impact (too much) the performance.

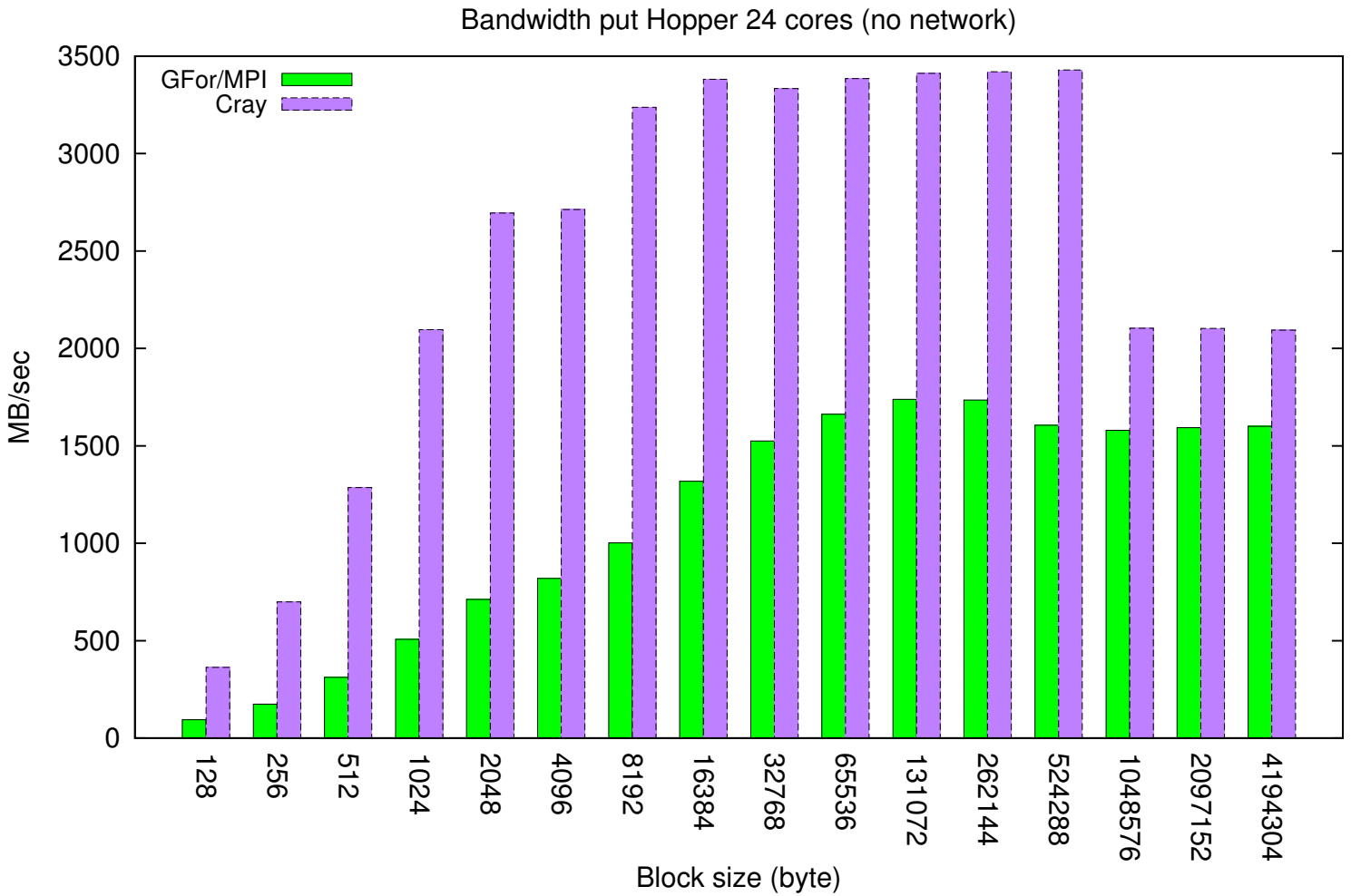
Bw Single Strided Put 16 cores Yellowstone



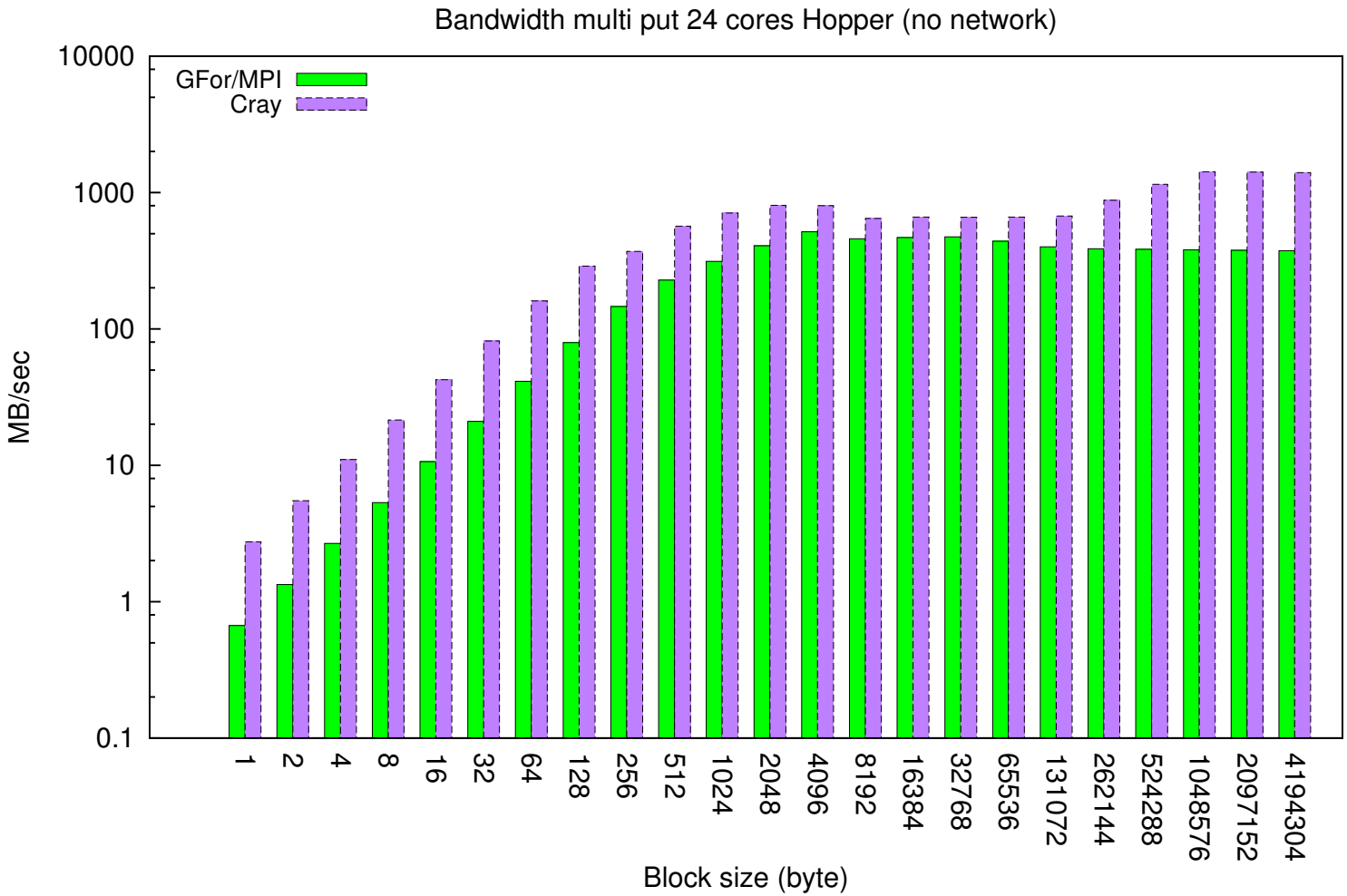
Bandwidth Single Put 24 cores Hopper (Small)



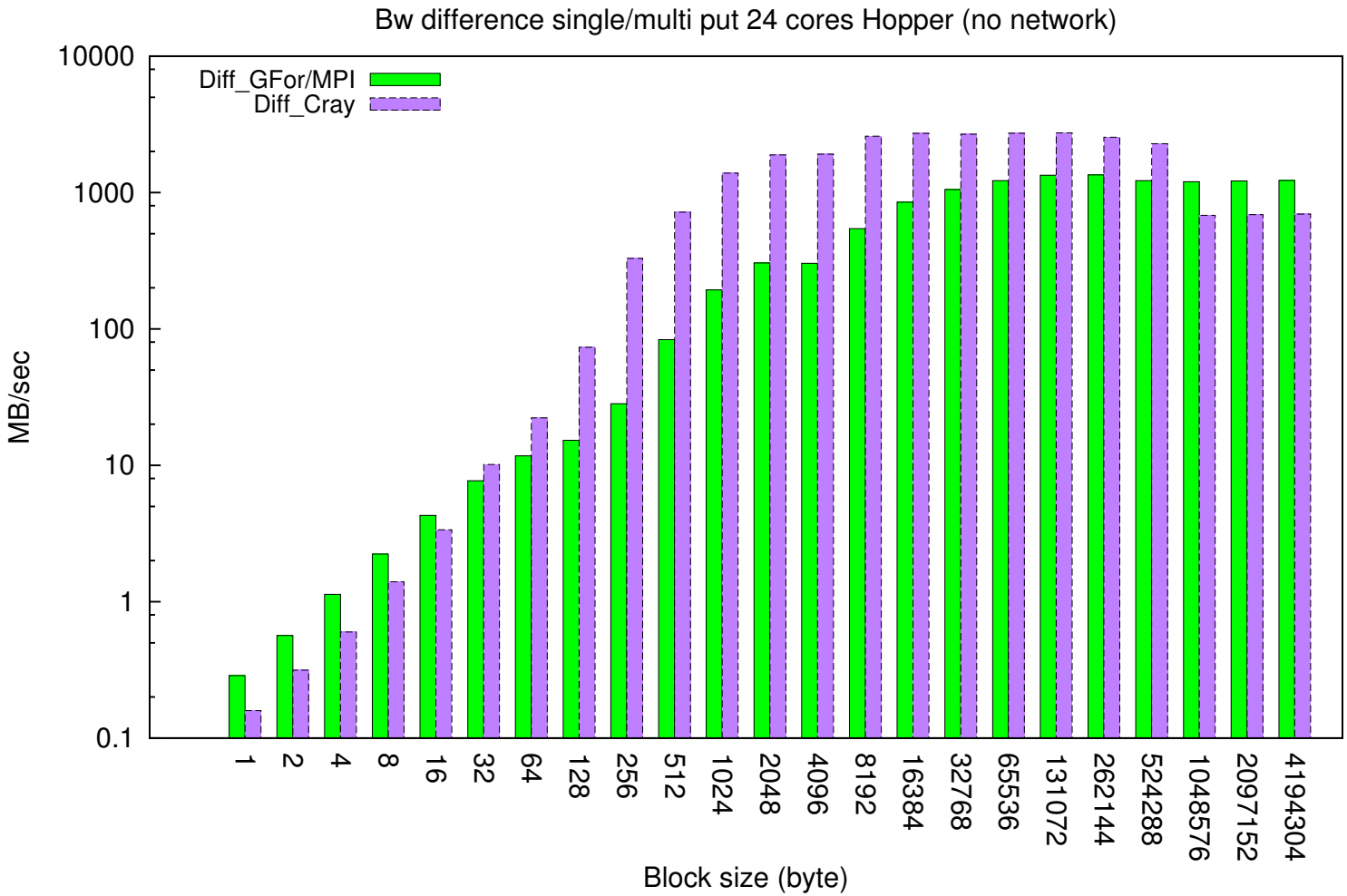
Bandwidth Single Put 24 cores Hopper (Big)



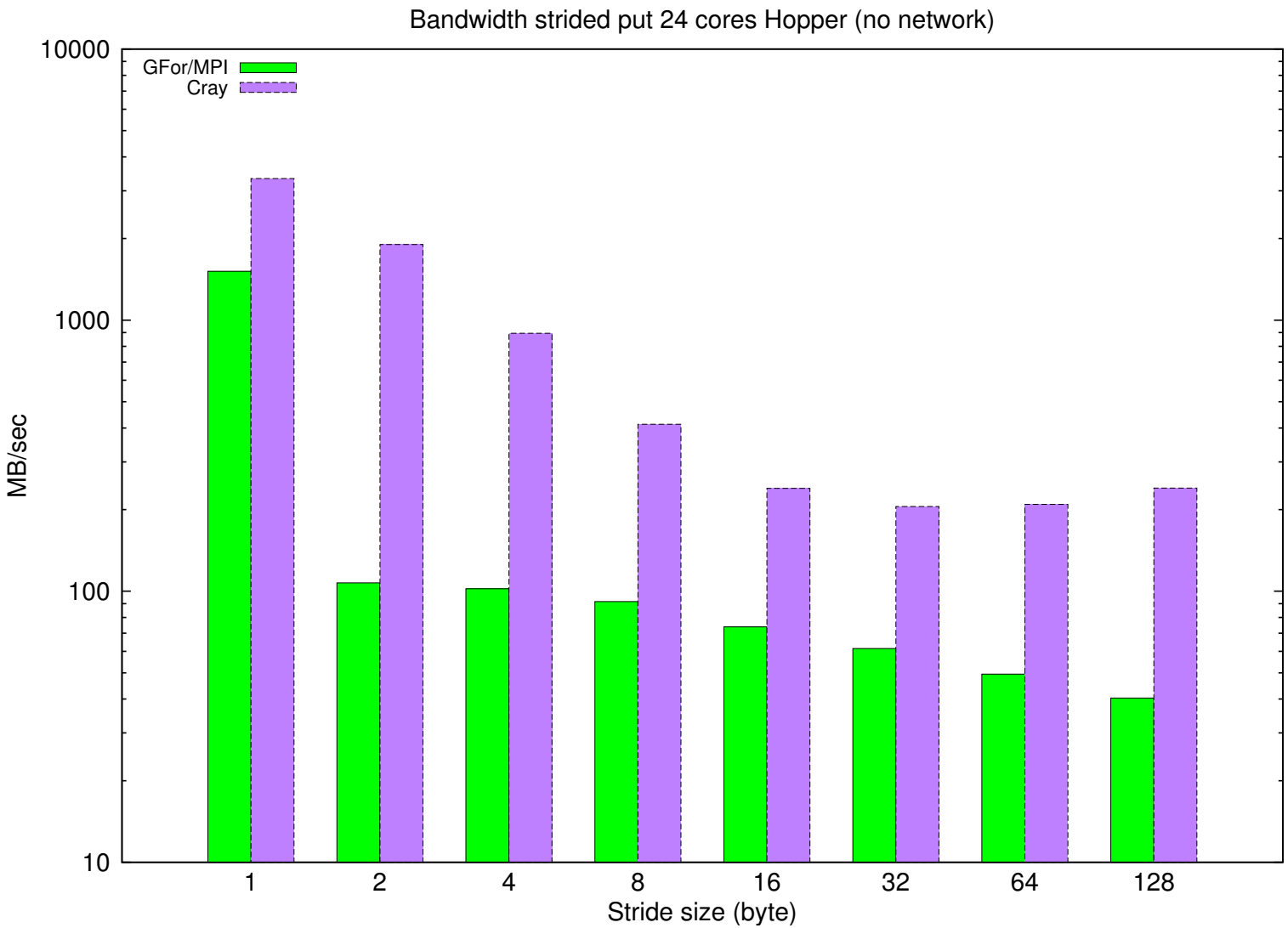
Bw Multi Put 24 cores Hopper



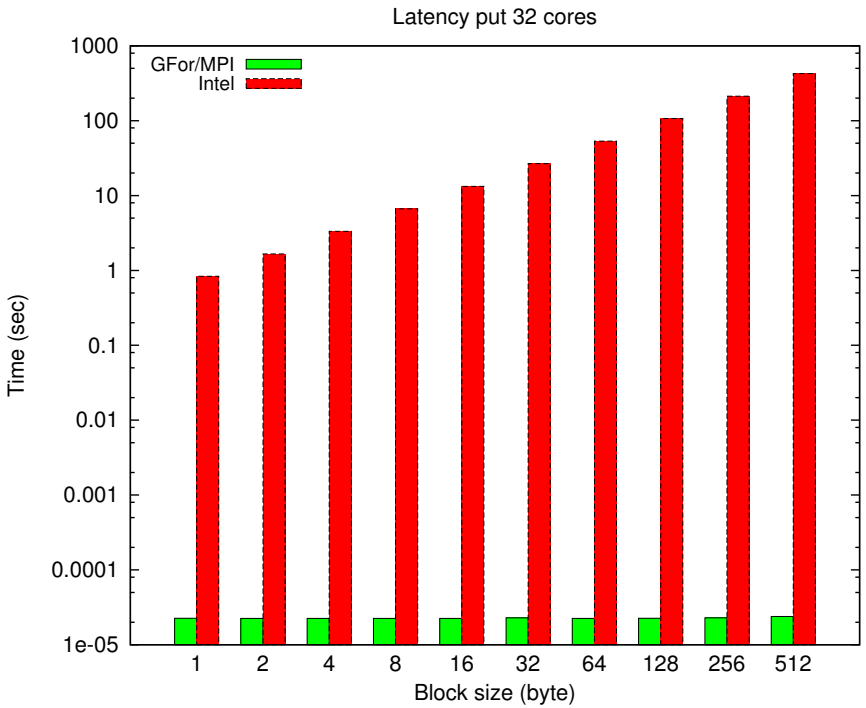
Bw Difference Single/Multi Put 24 cores Hopper

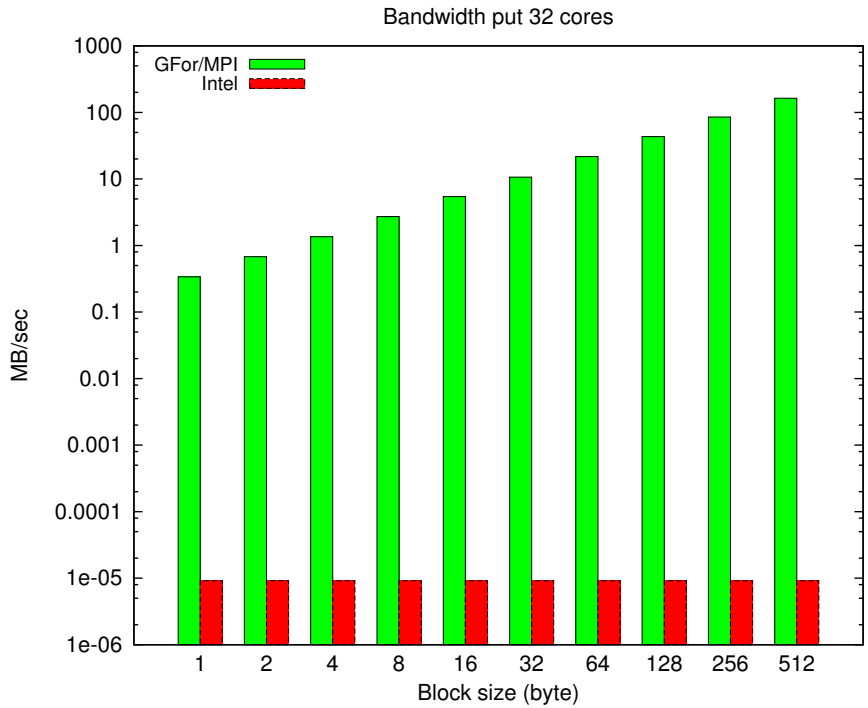


Bw Single Strided Put 24 cores Hopper



Latency/Bw Single Put 32 cores Yellowstone





Latency: lower is better - Bandwidth: higher is better

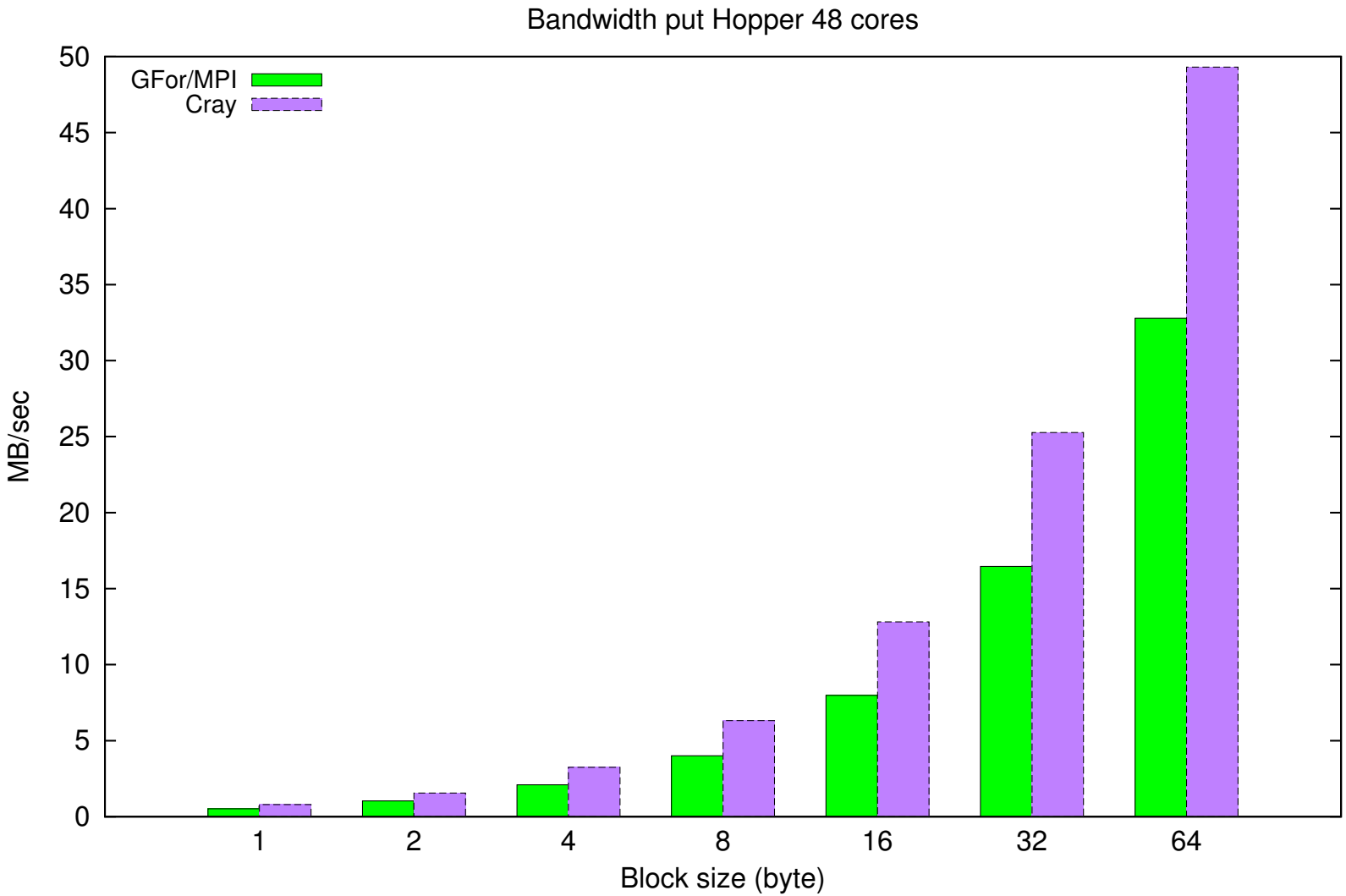
Note: Latency is expressed in seconds and Bw in MB/Sec.

Latency/Bw Intel 32 cores Yellowstone

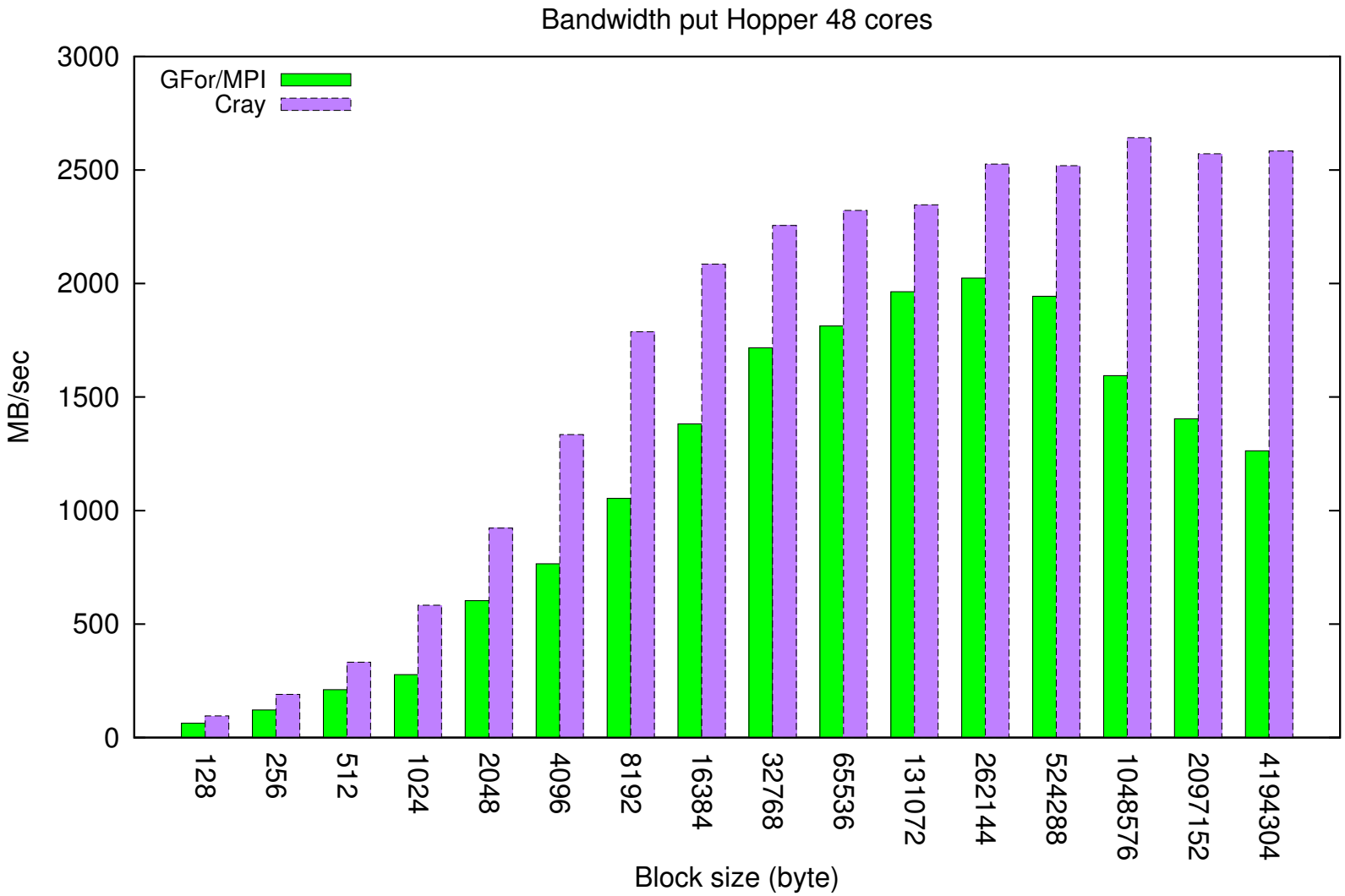
With 2 nodes we have a weird behavior from the Intel compiler.
The benchmark gets an internal error during the subput test for every configuration of nodes.

blksize (Bytes)	latency (sec)	bwidth (MB/s)
1	0.837	0.9133E-05
2	1.67	0.9178E-05
4	3.35	0.9157E-05
8	6.66	0.9159E-05
16	13.4	0.9156E-05
32	26.8	0.9155E-05
64	53.4	0.9148E-05
128	107.	0.9150E-05
256	214.	0.9149E-05
512	428.	0.9141E-05

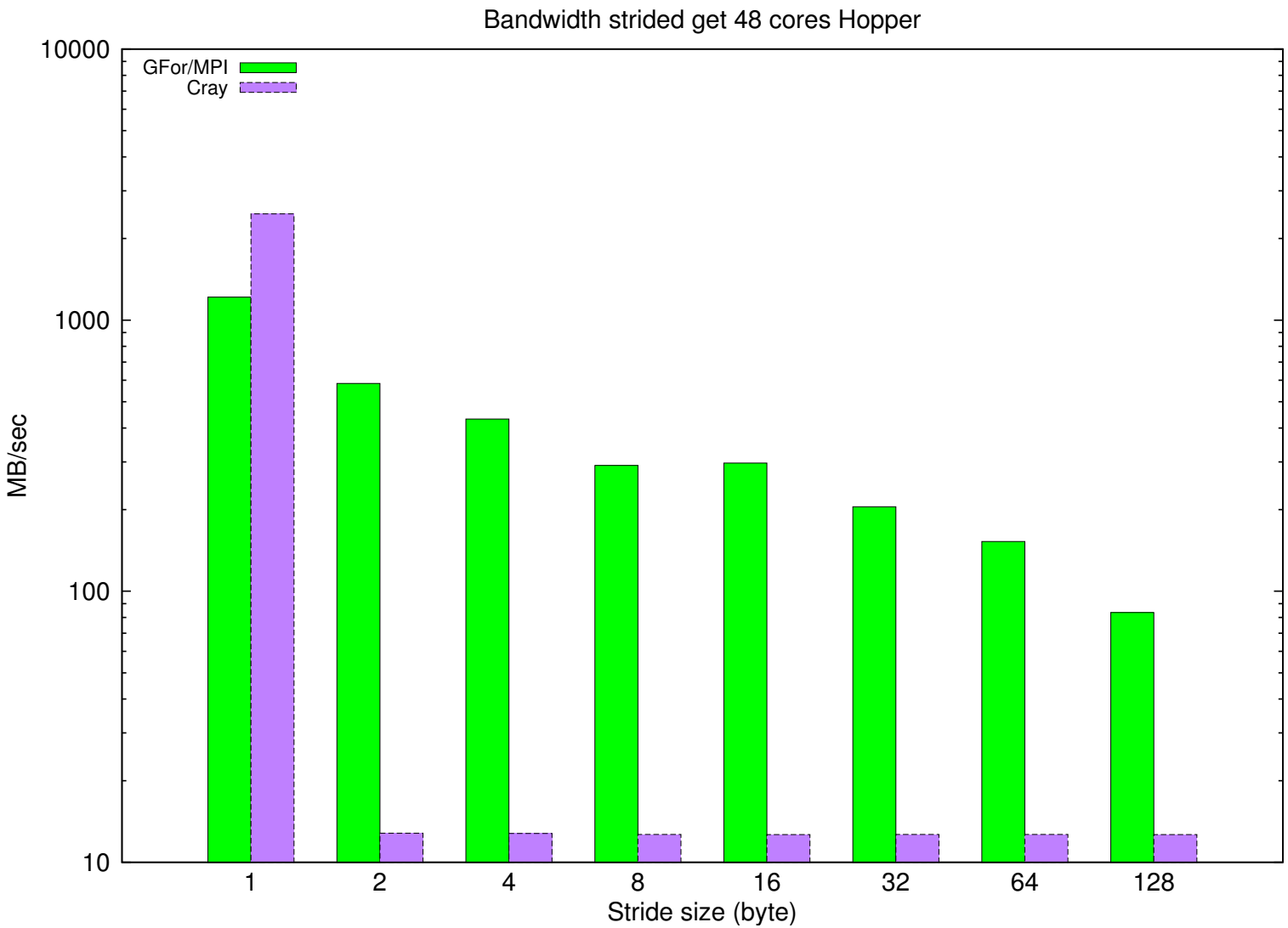
Bw Single Put 48 cores Hopper (small)



Bw Single Put 48 cores Hopper (big)



Bw Single Strided Get 48 cores Hopper



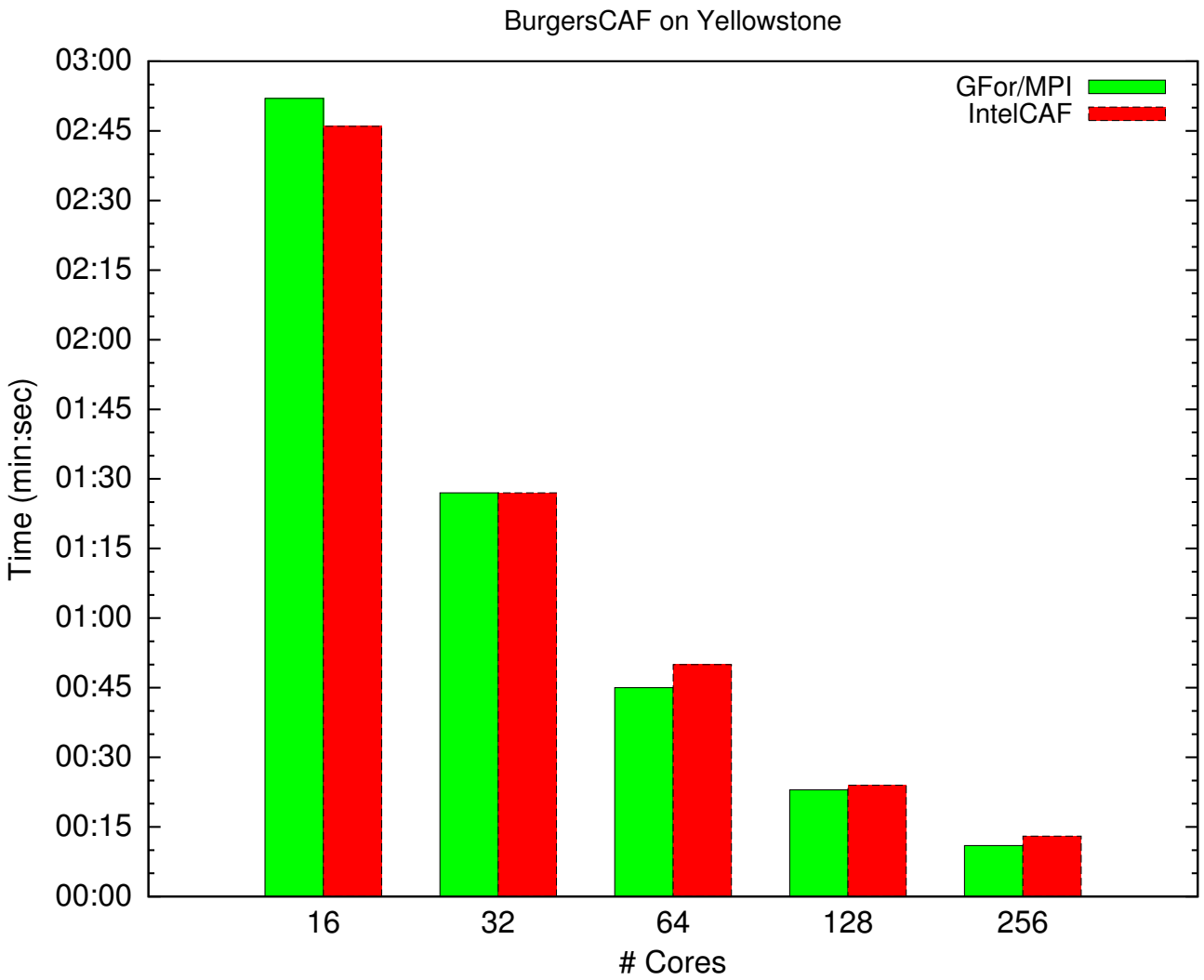
Bw Single Strided Put 48 cores Hopper (2)

- During a strided transfer Cray sends a predefined amount of data (perhaps elem-by-elem).
- GFortran uses the MPI Derived Data Types.
- Only the bandwidth is not enough for a fair comparison.
- Derived data types require a lot of memory (a Yellowstone's compute node does not have enough memory for running the entire benchmark).
- A good way to compare the two CAF implementation is through a Time/Memory trade-off.
- We plan to improve the strided transfer very soon.

BurgersCAF

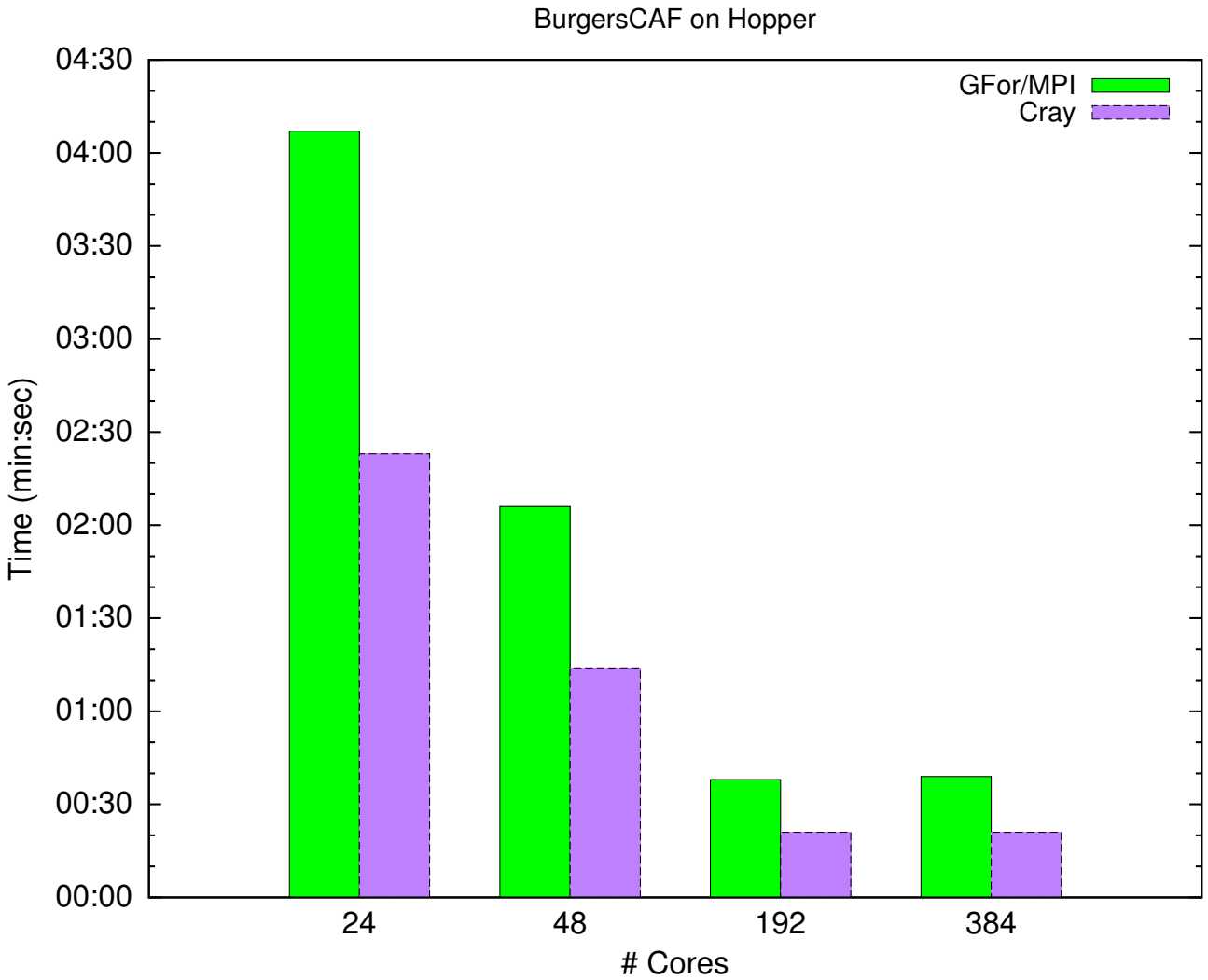
- It uses Coarrays mainly for scalar transfers.
- It communicates with neighbour images which are usually placed on the same node.
- The nature of the algorithm influences the performance a lot.

BurgersCAF - Yellowstone



(Lower is better)

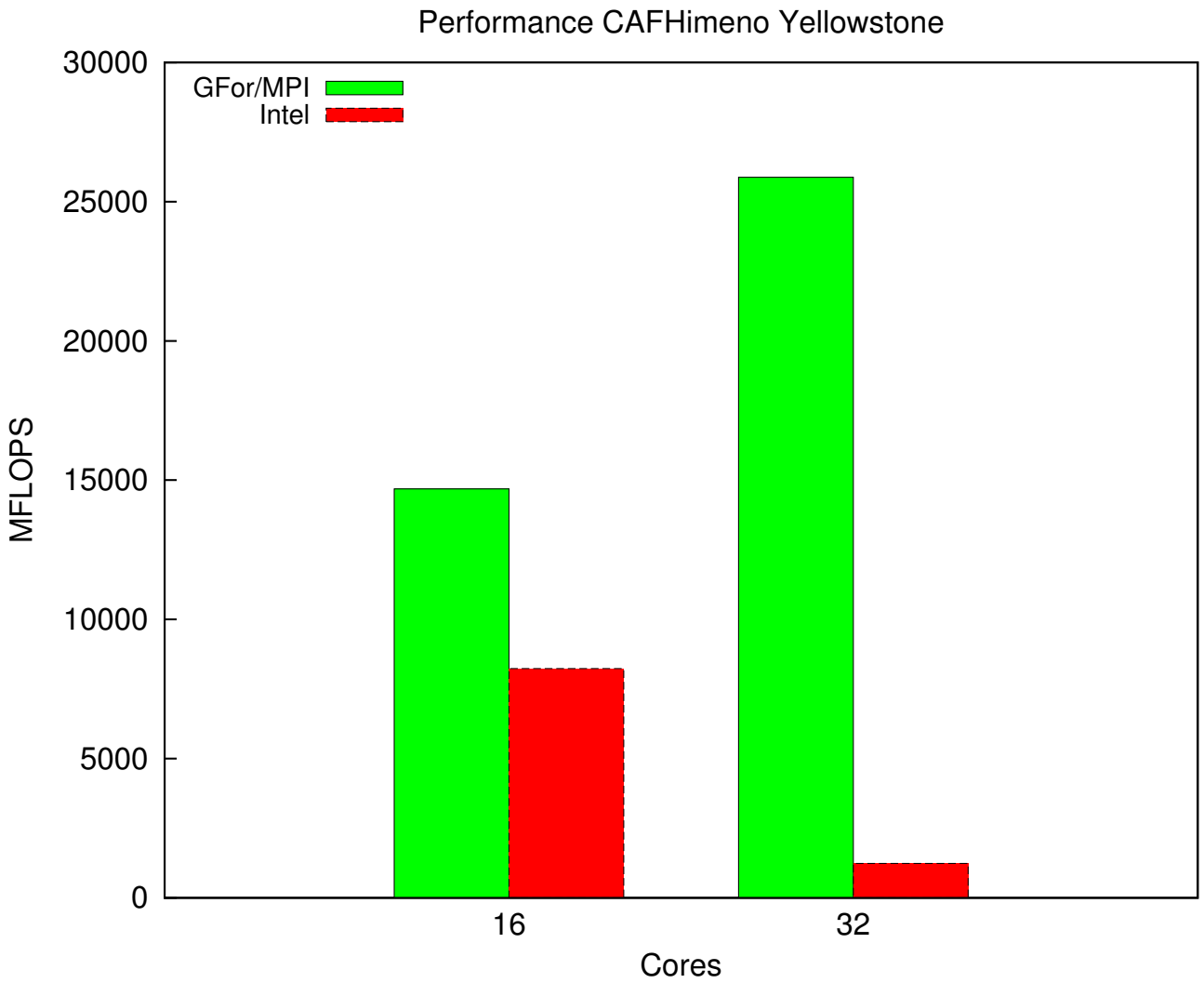
BurgersCAF - Hopper



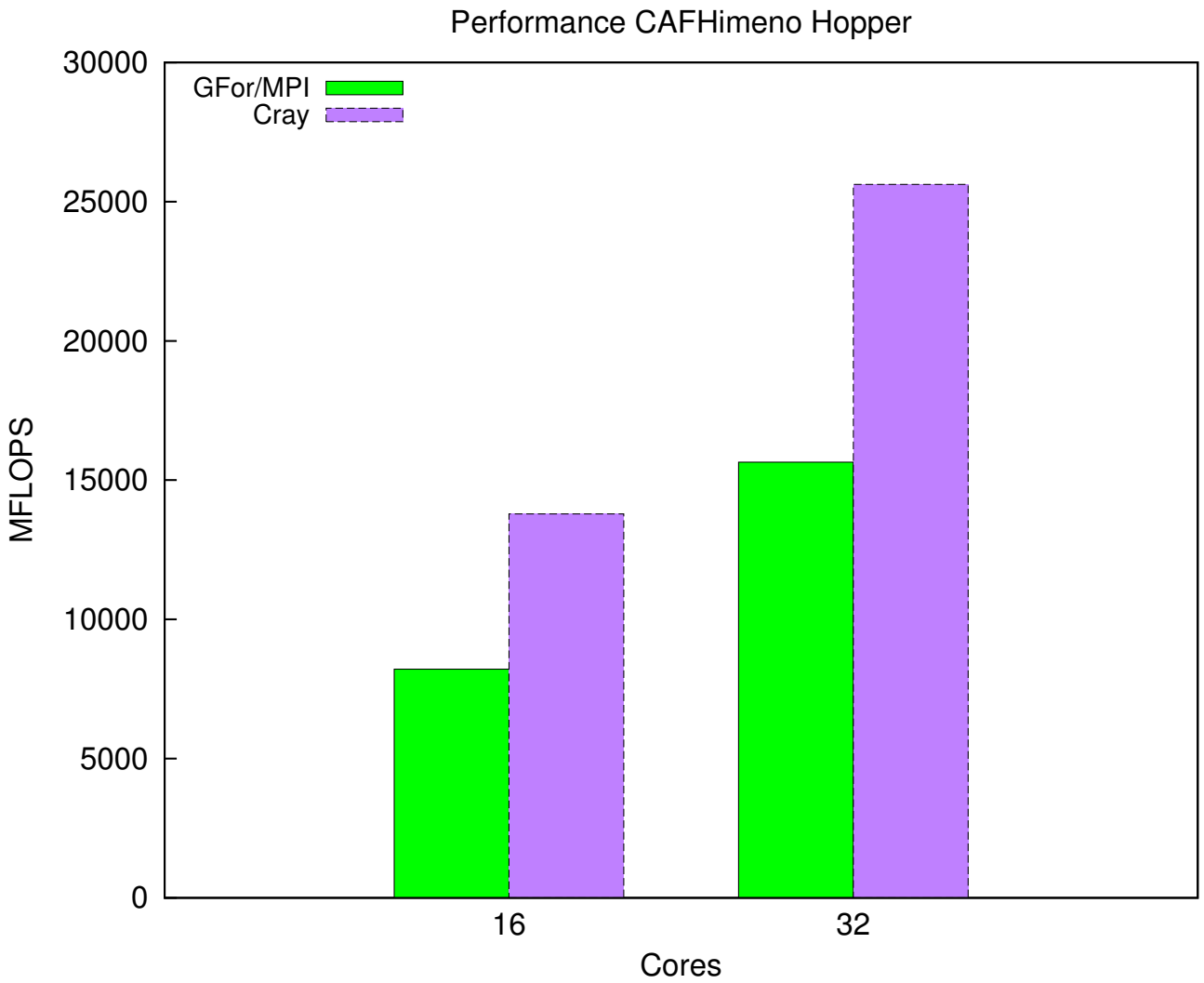
CAF Himeno

- It uses the Jacobi method for a 3D Poisson relaxation.
- Intel requires more than 30 minutes to complete the test with 64 cores.
- Cray requires several tuning arrangements in order to run (the 32 cores test has been run with 8 images on each node).
- GFortran was the easiest to run.

CAF Himeno - Yellowstone



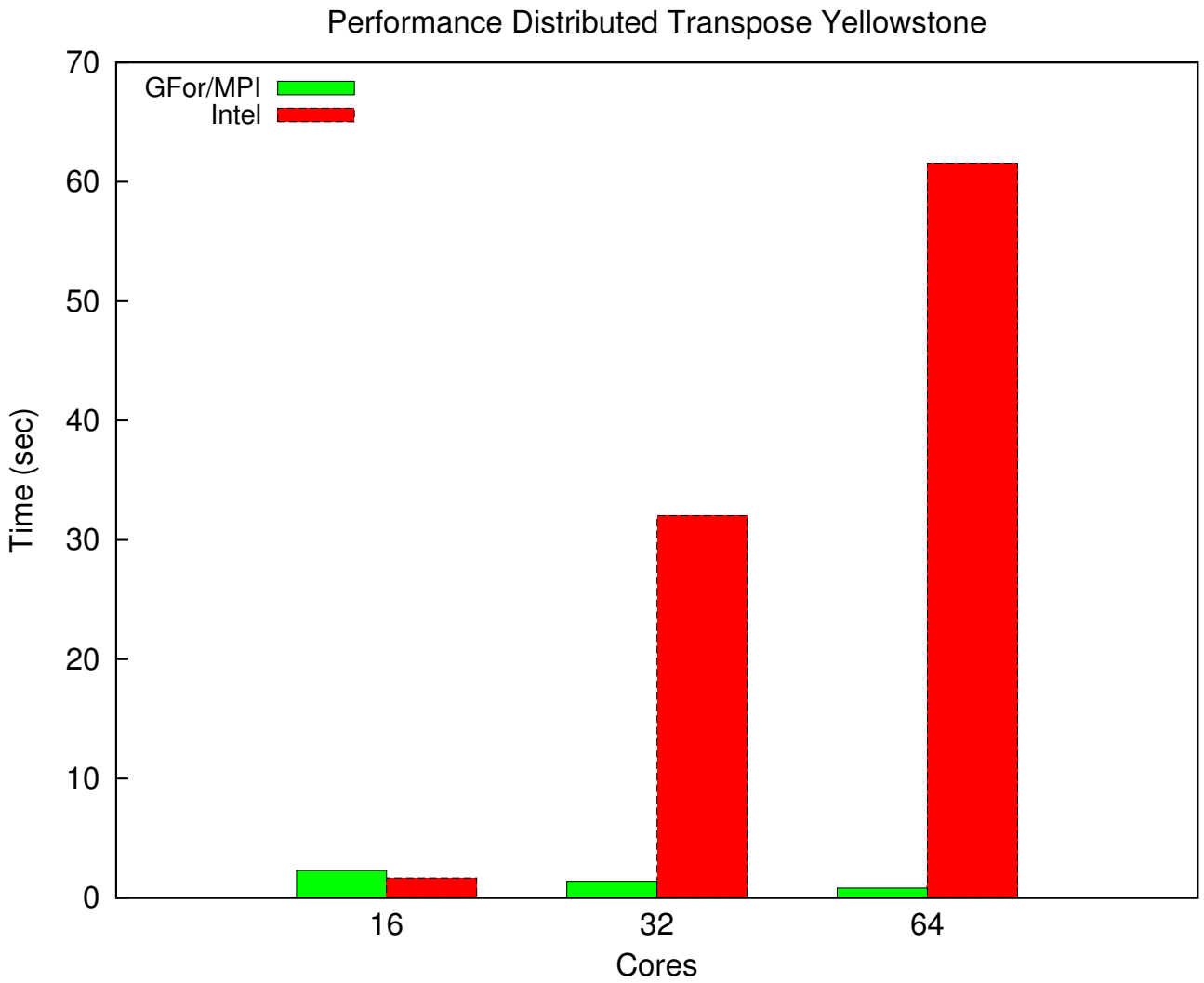
CAF Himeno - Hopper



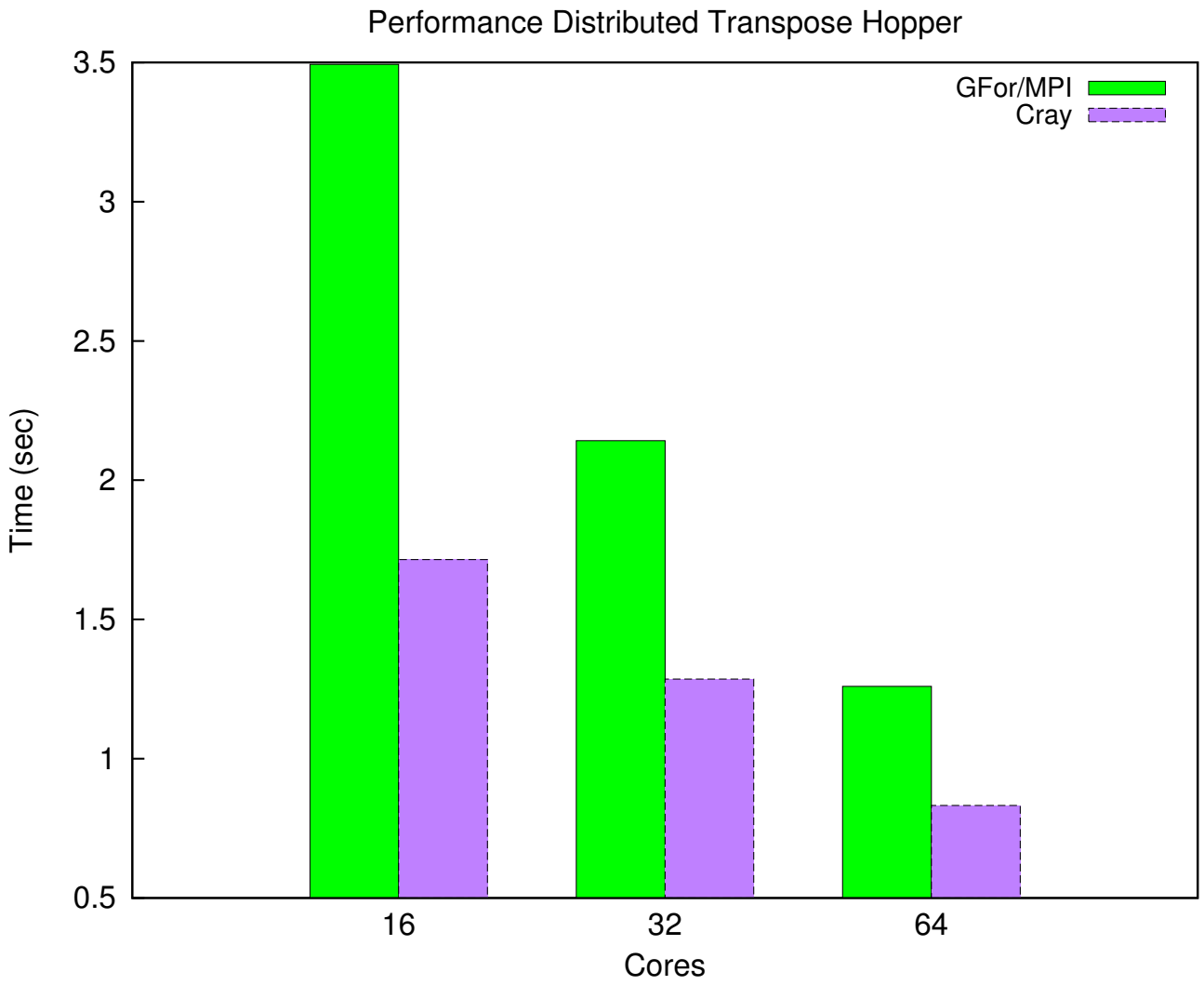
Distributed Transpose

- 3-D Distributed Transpose.
- It is performed sending element-by-element.
- The problem size has been fixed at 1024,1024,256.
- It is a good communication test case.

Distributed Transpose - Yellowstone



Distributed Transpose - Hopper



Distributed MxV

- A group of rows is sent to each image.
- The usual array syntax has been used (it implies strided array transfer).
- The bottleneck is communication (broadcast and gather).
- With a single MxV the computational part is too small for getting benefits from a parallel approach.

Distributed MxV - Executions

Yellowstone 16 cores (single node)

GFor/MPI	Intel
0:40 (m:s)	5:09 (m:s)

Hopper 16 cores (single node)

GFor/MPI	Cray
1:54 (m:s)	0:08 (m:s)

Conclusions - GFortran vs. Intel

- Intel shows better performance than GFortran only during scalar transfers within the same node.
- Intel pays a huge penalty when it uses the network (multi node).
- GFortran shows better performance than Intel on array transfers within the same node.
- GFortran shows better performance than Intel in every configuration which involves the network.

Conclusions - GFortran vs. Cray

- Cray has better performance than GFortran within the same node.
- Cray has better performance than GFortran during contiguous array transfer on multi node (using the network).
- GFortran has better performance than Cray for strided transfers on multi node. (!!!)
- Cray requires some tuning (env vars and modules).

Conclusions

- GFortran provides a stable, easy to use and efficient implementation of Coarrays.
- GFortran provides a valid and free alternative to commercial compilers.
- Coarrays in GFortran can be used on any architecture able to compile GCC and a standard MPI implementation.

Future developments

- On the compiler side the multi image Coarray support is already on GCC 4.10 trunk (It also supports OpenMPv4).
- On the library side we plan to release the MPI version very soon.
- Improve the strided transfer support and add all the missing features expected by the standard.
- Improve the experimental GASNet version.

The Team

Development Team:

- Tobias Burnus
- Alessandro Fanfarillo

Support Team:

- Valeria Cardellini
- Salvatore Filippone
- Dan Nagle
- Damian Rouson

Acknowledgment

- CINECA (Grant HyPSBLAS)
- Google (Project GSoC 2014)
- UCAR/NCAR
- NERSC (Grant OpenCoarrays)

Thanks

Suggestions/Feedback/Questions?