Introduction to coarray Fortran using GNU Fortran

Alessandro Fanfarillo

University of Rome Tor Vergata

fanfarillo@ing.uniroma2.it

December 19th, 2014

Modern Fortran

FORTRAN 77 is usually what people think when you say “Fortran”.

Since Fortran 90 the language has evolved in order to address the modern programming needs.

- Fortran 90: free form, recursion, dynamic memory, modules, derived types and much more...
- Fortran 95: Pure and elemental routines.
- Fortran 2003: Object-oriented programming and C interoperability.
- Fortran 2008: Parallel programming with coarrays and do concurrent.

Introduction to coarray

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)
- **GNU Fortran (Free - GCC)**
- Rice Compiler (Free - Rice University)
- OpenUH (Free - University of Houston)

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).

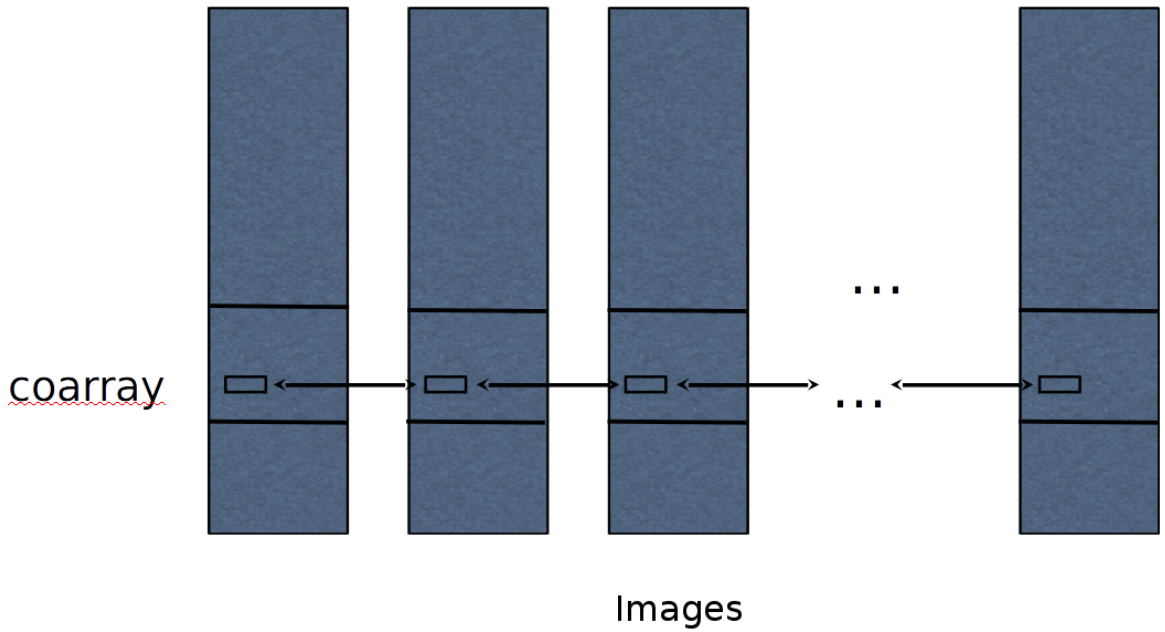
Coarray concepts

- A program is treated as if it were replicated at the start of execution (SPMD), 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.

Memory basics

When we declare a coarray variable the following statements are true:

- The coarray variable exists on each image.
- The coarray name is the same on each image.
- The size is the same on each image.



Coarray declaration

```
! Scalar coarray
integer :: x[*]

! Array coarray
real, dimension(n) :: a[*]

! Another array declaration
real, dimension(n), codimension[*] :: a

! Scalar coarray corank 3
integer :: cx[10,10,*]

! Array coarray corank 3
! different cobounds
real :: c(m,n) :: [0:10,10,*]

! Allocatable coarray
real, allocatable :: mat(:,:)[:]
allocate(mat(m,n)[*])

! Derived type scalar coarray
type(mytype) :: xc[*]
```

Coarray example

```
real, dimension(10), codimension[*] :: x, y
integer :: num_img, me
integer,dimension(5) :: idx

num_img = num_images();  me = this_image()
idx = (/1,2,3,9,10/)

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
! Remote-to-remote array transfer
if (me == 1) then
  y(:)[num_img] = x(:)[4]
  sync images(num_img)
elseif (me == num_img) then
  sync images([1])
end if

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

y(idx) = x(idx)[8] ! Get using vector subscript
```

Segments

- A segment is a piece of code between synchronization points.
- The compiler is free to apply optimizations within a segment.
- Segments are ordered by synchronization statement and dynamic memory actions (allocate).

Important rule: if a variable is defined in a segment, it must not be referenced, defined, or become undefined in a another segment unless the segments are ordered.

```
real :: p[*]
:                                ! Segment 1
sync all
if (this_image()==1) then        ! Segment 2
read (*,*) p                     ! :
do i = 2, num_images()           ! :
p[i] = p                         ! :
end do                           ! :
end if                           ! Segment 2
sync all
:                                !Segment 3
```

Derived types

```
type mytype
  integer :: a
  real, dimension(3) :: dims
  real, allocatable  :: v(:)
end type mytype

!Some code here

type(mytype) :: my[*]
type(mytype),dimension(2) :: arrmy[*]

allocate(my%v(n))

if(me==2) then
  my%dims(:) = my[1]%dims(:)
  arrmy(:) = arrmy(:)[1]
endif
```

In GNU Fortran there are still some limitations on accessing derived type's components (e.g. allocatable components).

Synchronization

- sync all: barrier on all images
- sync images(list-of-images): partial barrier
- critical: only one image at a time can access a critical region
- locks: similar to critical but with more flexibility

Locks and critical

Example using locks

```
use iso_fortran_env
TYPE(LOCK_TYPE) :: queue_lock[*]
integer :: counter[*]
!Some code
lock(queue_lock[2])
counter[2] = counter[2] + 1
unlock(queue_lock[2])
```

Example using critical

```
real(real64),codimension[*] :: sum
!Some code here
do i=start_int,start_int + l_int-1
  x=dx*(i-0.5d0)
  f=4.d0/(1.d0+x*x)
  critical
  sum[1]=sum[1]+f
  end critical
end do
```

Collectives

- co_sum
- co_max/co_min
- co_reduce (not yet supported by the library)
- co_broadcast

Pi computation: pi = 4* int 1/(1+x*x) x in [0-1]

```
real(real64),codimension[*] :: sum
real(real64) :: dx,x
real(real64) :: f,pi
!Some code here
dx=1.d0/intervals
do i=start_int,start_int + l_int-1
  x=dx*(i-0.5d0)
  f=4.d0/(1.d0+x*x)
  sum=sum+f
end do
call co_sum(sum)
pi = dx*sum
if(me == 1) write(*,*) pi
```

Atomics

An atomic can be integer or logical

- Atomic_define
- Atomic_ref
- Atomic_add
- Atomic_cas
- Atomic_or
- Atomic_and

```
use iso_fortran_env
integer(atomic_int_kind) :: counter[*]
integer ::  res
!Some code
call atomic_add (counter[2], 1)
!Some code
call atomic_ref(res,counter[2])
```

Coarray references

- Original Paper at ftp://ftp.numerical.rl.ac.uk/pub/reports/nrRAL98060.ps.gz
- WG5 N1824.pdf (John Reid's summary) at www.nag.co.uk/sc22wg5
- Rice University - caf.rice.edu
- FDIS J3/10-007r1.pdf (www.j3-fortran.org)
- Modern Fortran Explained by Metcalf, Reid, Cohen.

GNU Fortran and OpenCoarrays (1)

GNU Fortran translates the coarrays syntax into library calls.

```
real(real64), allocatable :: d(:)[:]

! More code here

if(me == 1) d(:)[2] = d(:)

! More code here
```

Becomes:

```
if (me == 1)
{
  _gfortran_caf_send (d.token, 0,
  3 - (integer(kind=4)) d.dim[1].lbound, &d, 0B, &d, 8, 8);
}
```

Library API:

```
void caf_send (caf_token_t token, size_t offset, int image_index,
gfc_descriptor_t *dest, caf_vector_t *dst_vector,
gfc_descriptor_t *src, int dst_kind, int src_kind)
```

GNU Fortran and OpenCoarrays (2)

GFortran uses an external library to support coarrays (libcaf/OpenCoarrays).

Currently there are three libcaf implementations:

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

LIBCAF_MPI uses passive One-Sided communication functions (using MPI_Win_lock/unlock) provided by MPI-2/MPI-3.

GASNet provides higher performance than MPI and PGAS functionalities but is more difficult to use.

MPI version - Current status

Supported Features:

- Coarray scalar and array transfers (efficient strided transfers)
- Synchronization
- Collectives
- Atomics
- Critical
- Locks (compiler support missing)

Unsupported Features:

- Vector subscripts
- Derived type coarrays with non-coarray allocatable components
- Error handling

How to use coarrays in GFortran (1)

Compiler:

The coarray support on GFortran is available since version GCC 5.

Currently, GCC 5.0 is downloadable from the trunk using SVN/GIT or from a snapshot; in both cases the user is supposed to compile GCC from source.

Library:

OpenCoarrays is not part of GCC; it has to be downloaded separately.

opencoarrays.org provides a link to the OpenCoarrays repository.

Once downloaded just type "make" and the LIBCAF_MPI will be compiled by default.

How to use coarrays in GFortran (2)

Assuming MPICH as MPI implementation, the wrapper mpifort has to point to the new gfortran.

In order to turn on the coarray support during the compilation you must use the -fcoarray=lib option.

```
mpifort -fcoarray=lib youcoarray.f90 -L/path/to/libcaf_mpi.a
-lcaf_mpi -o yourcoarray
```

For running your coarray program:

```
mpirun -np n ./yourcoarray
```

Coarray support comparison

Coarray support

In order to evaluate the coarray support provided by GNU Fortran we make a comparison with the two most complete compilers.

- Cray:
  - Pros: Most mature, reliable, efficient and complete coarray implementation.
  - Cons: Commercial compiler. Requires a Cray supercomputer.
- Intel:
  - Pros: Pretty complete implentation of coarray functionality.
  - Cons: Commercial compiler. Works only on Linux and Windows.

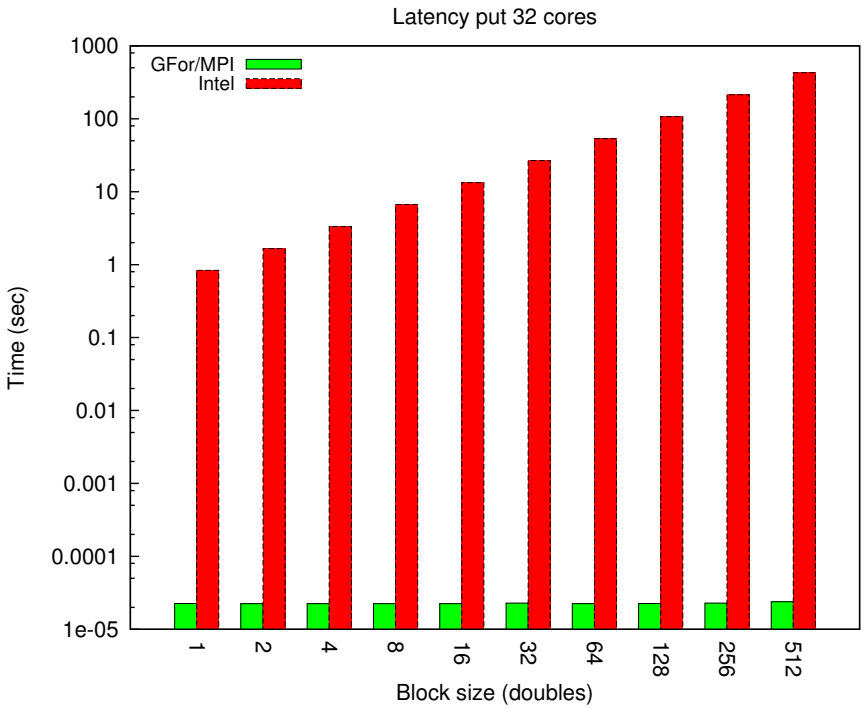
Test Suite Description

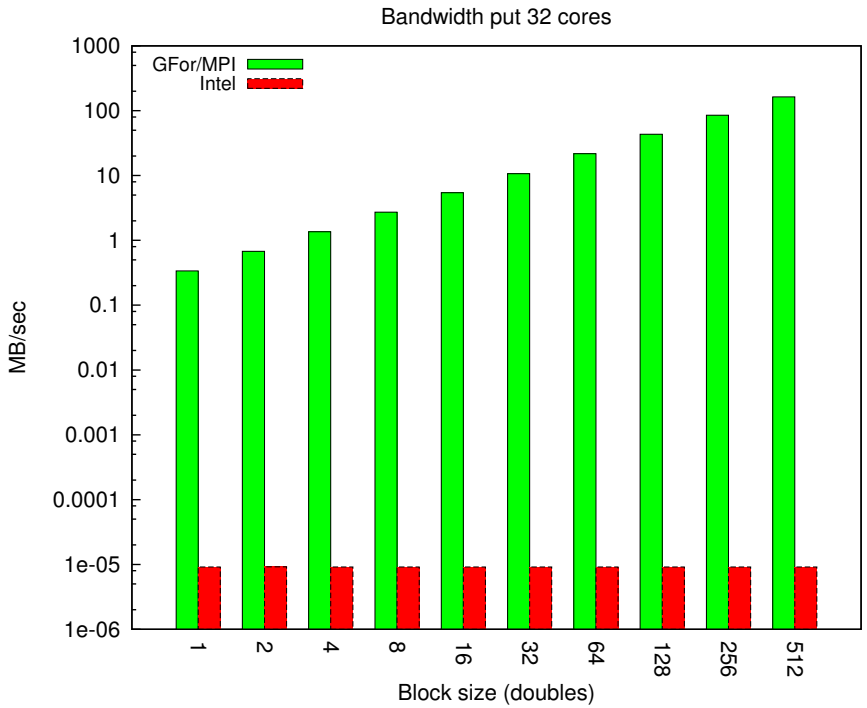
- **EPCC CAF Micro-benchmark suite** (Edinburgh University)
  - It measures a set of basic parallel operations (get, put, strided get, strided put, sync, halo exchange).
- Burgers Solver (Damian Rouson)
  - It has nice scaling properties: it has a 87% weak scaling efficiency on 16384 cores and linear 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 part of a Navier-Stokes solver.

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).

GFortran vs. Intel two Yellowstone nodes

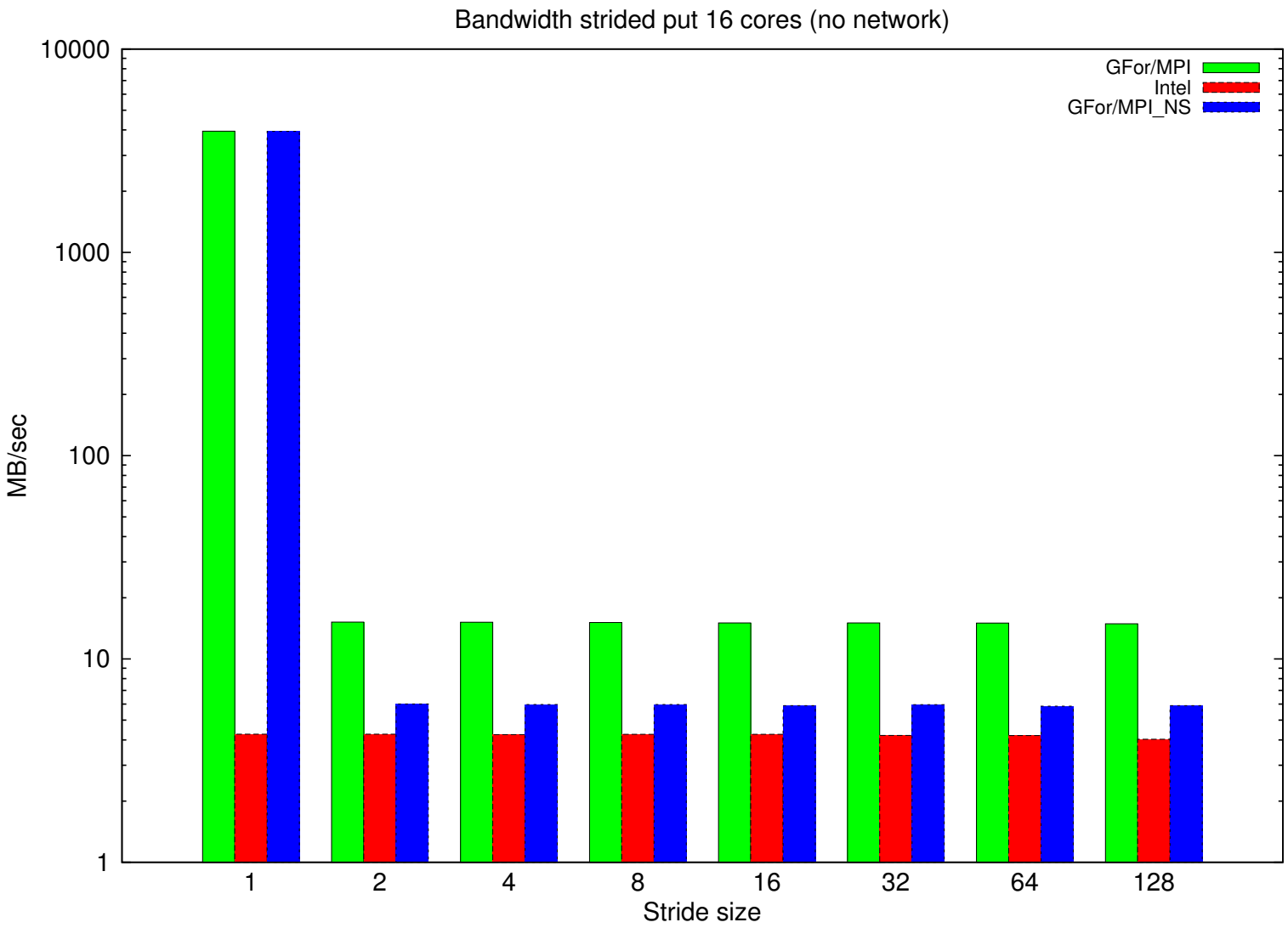




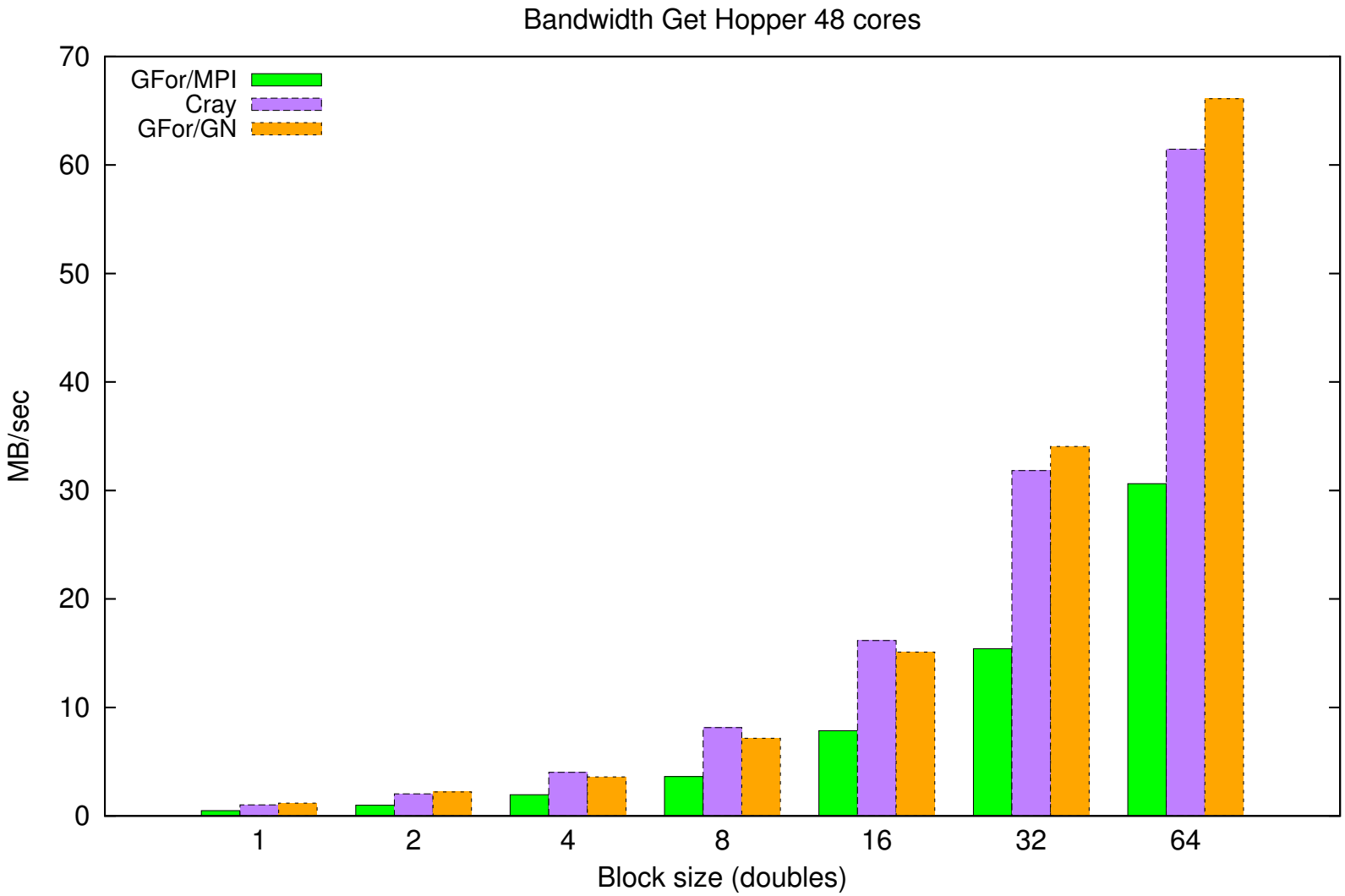
Latency: lower is better - Bandwidth: higher is better

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

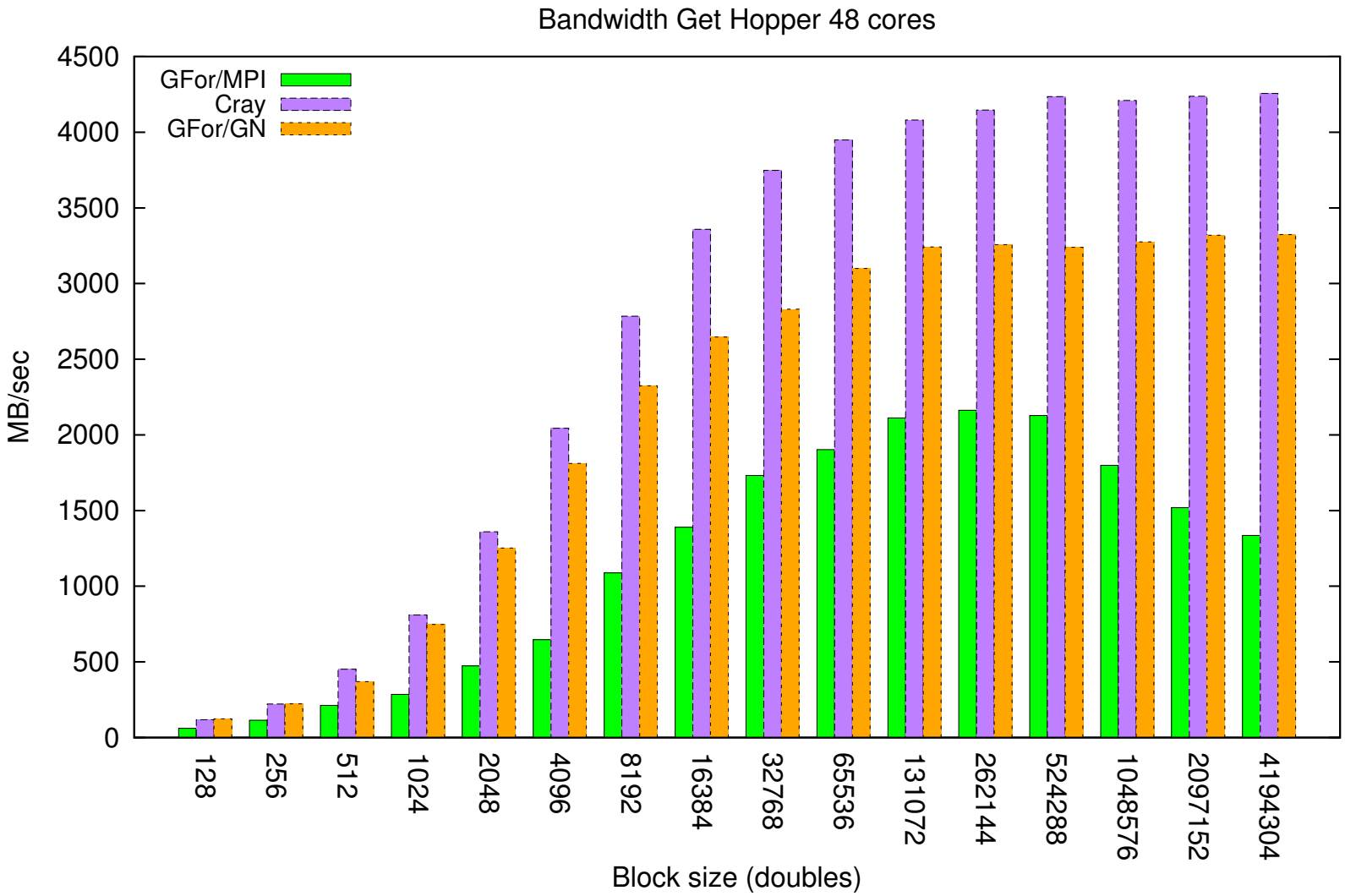
Bw Single Strided Put 16 cores Yellowstone



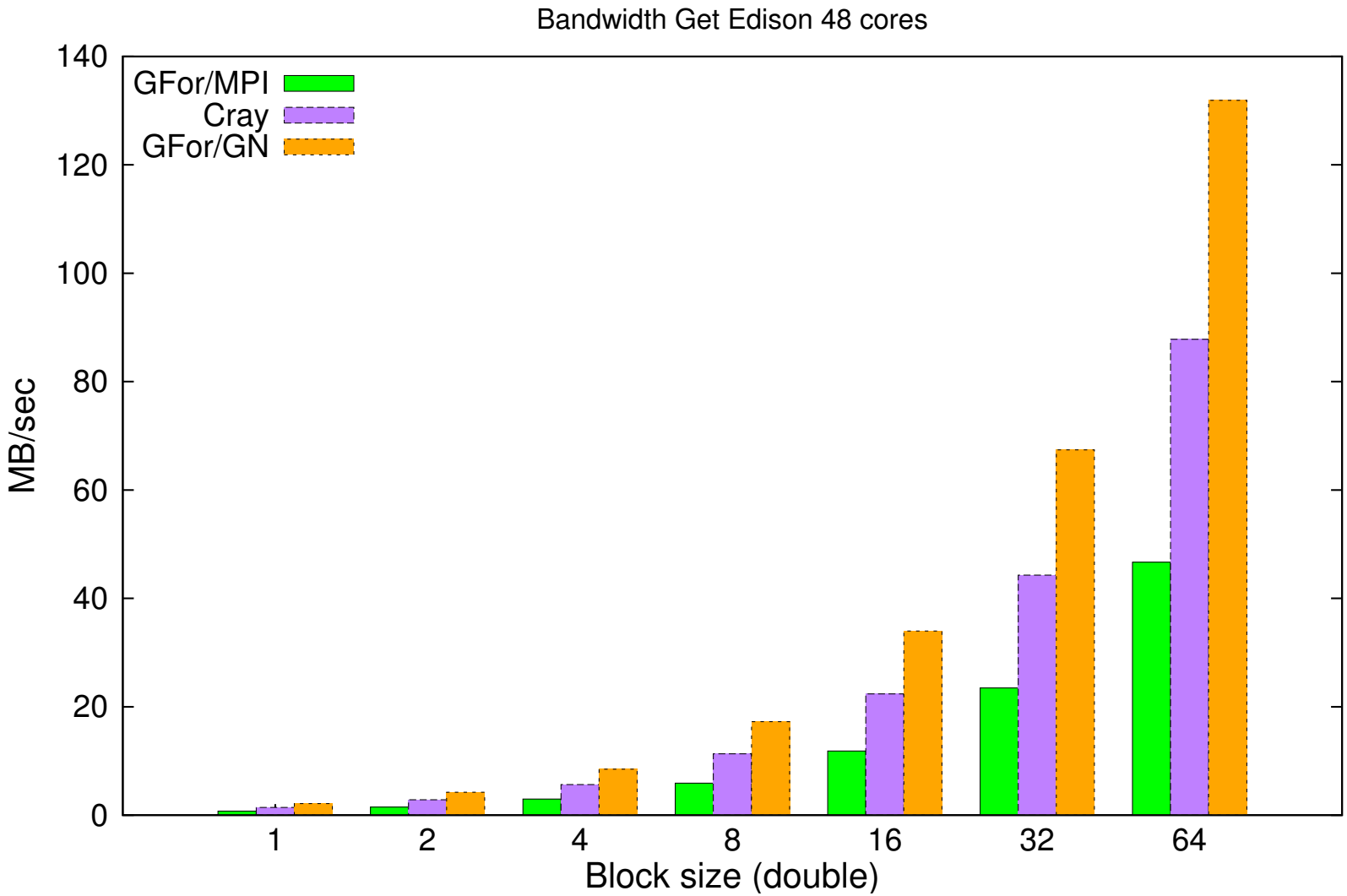
Bw two Hopper nodes - Small sizes



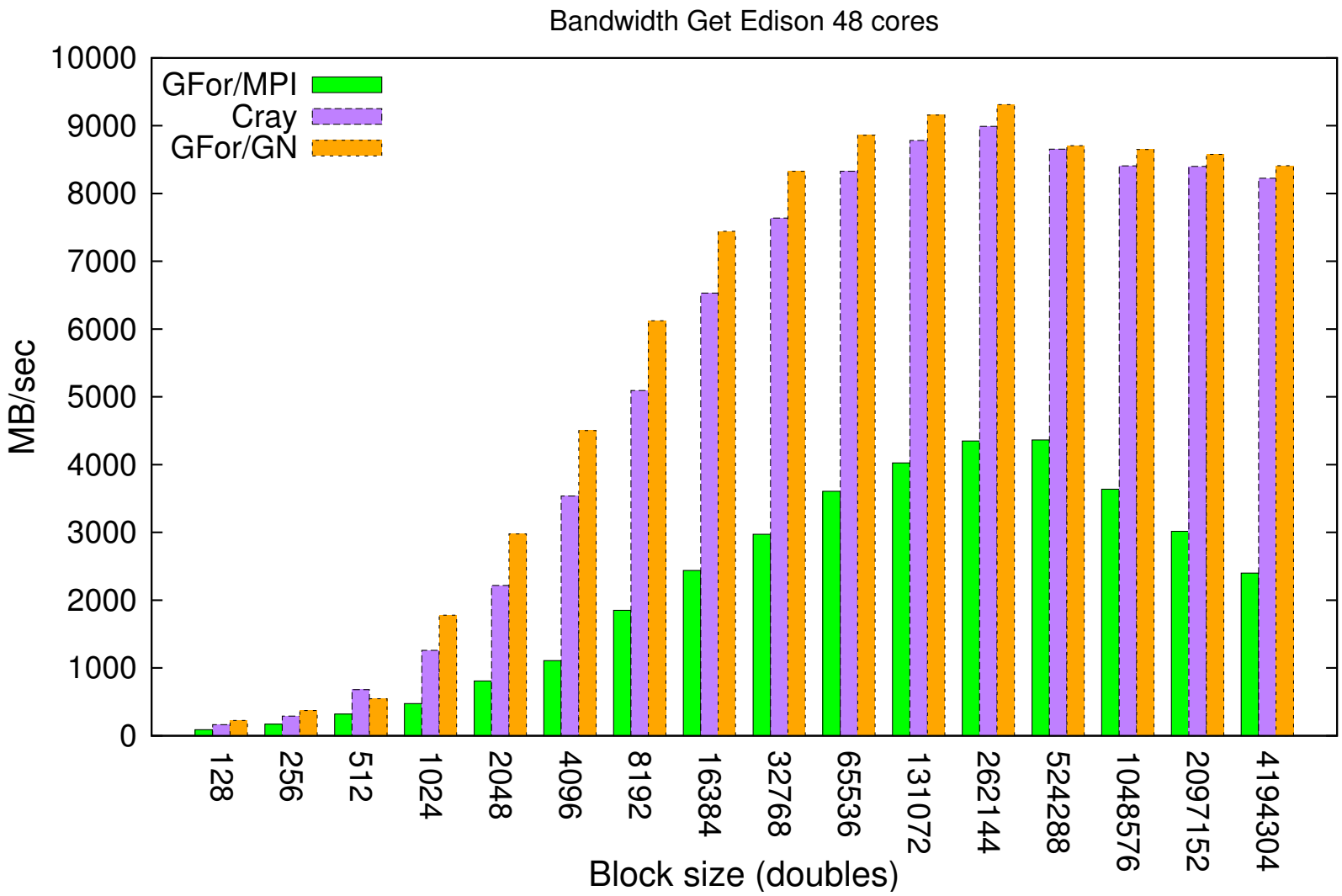
Bw two Hopper nodes - Big sizes



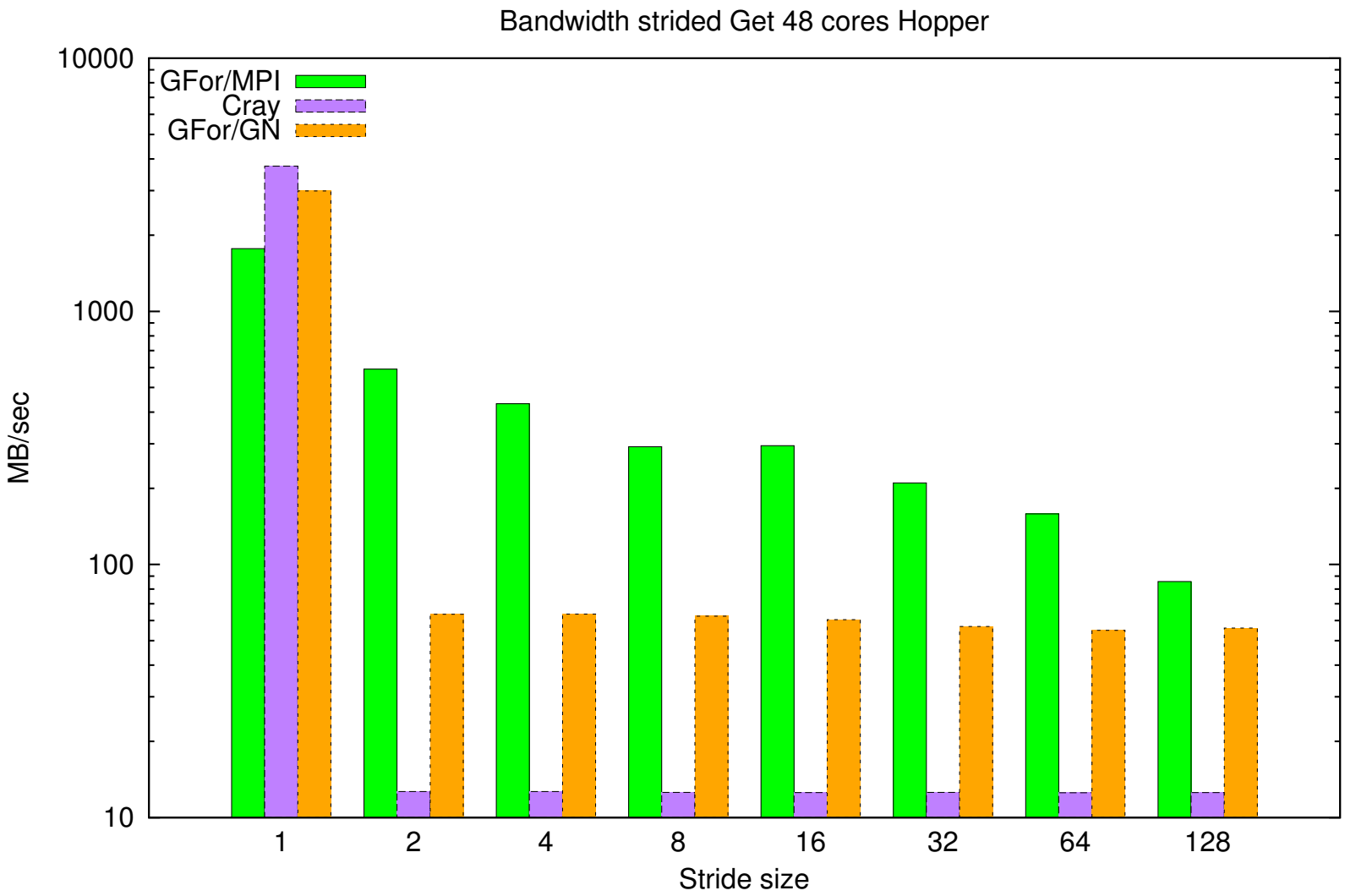
Bw Single Get 48 cores Edison (small)



Bw Single Get 48 cores Edison (Big)

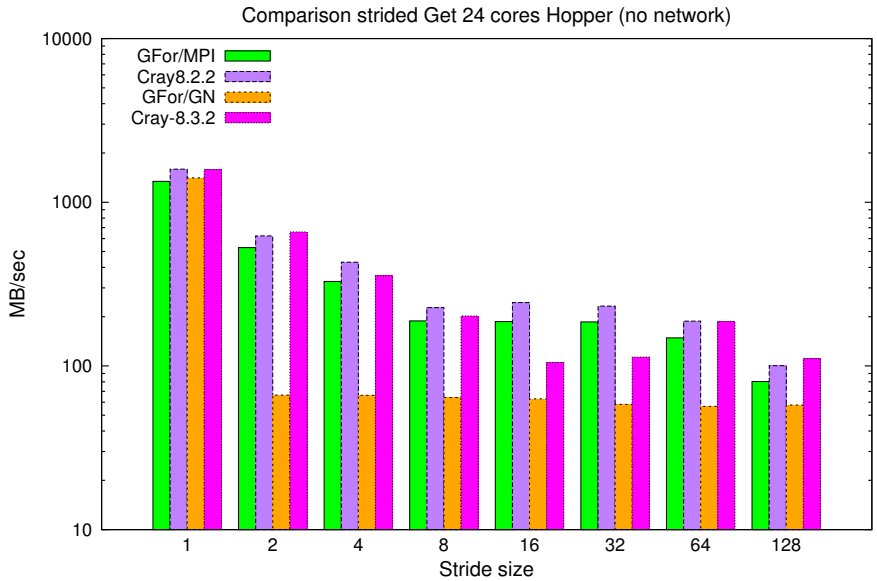


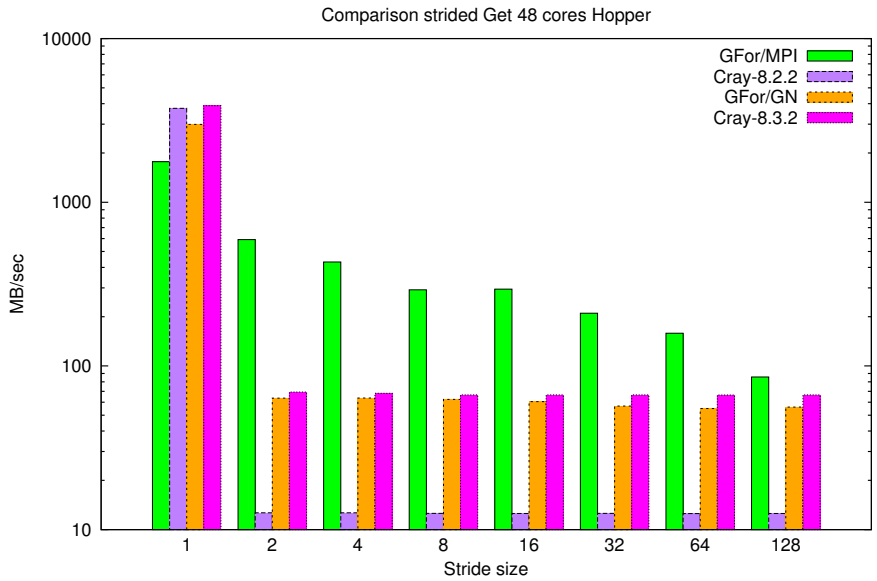
Bw Strided Get on Hopper - Two nodes



Bw Strided Get on Hopper (2)

The Cray strided transfer support has been strongly improved in the new CCE 8.3.2.

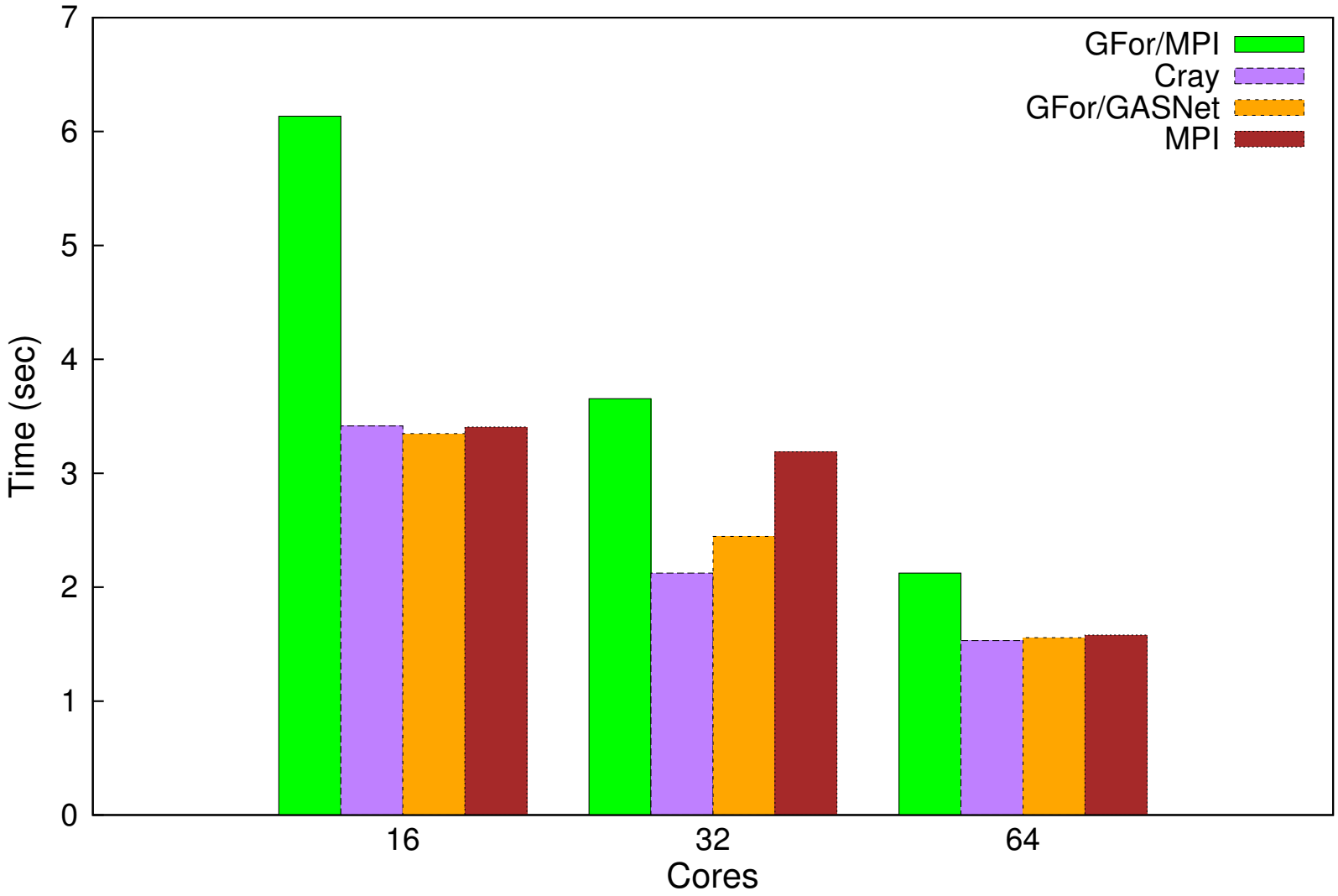




Distributed Transpose

- 3-D Distributed Transpose.
- The problem size has been fixed at 1024,1024,512.
- Part of a parallel Navier-Stokes solver.

Distributed Transpose - Hopper



Conclusions - GFortran vs. Intel

Intel Pros:

- Intel provides good coverage in terms of coarray functionality.
- Intel shows better performance than GFortran only during scalar transfers within the same node.

Intel Cons:

- Intel pays a huge penalty when it uses the network (multi node).
- Coarray support provided only on Linux and Windows.

GFortran Pros:

- GFortran shows better performance than Intel in every configuration which involves the network.
- GFortran works on any architecture able to compile GCC and a standard MPI implementation.
- GFortran is free.

GFortran Cons:

- GFortran does not yet cover all the coarray features.

Conclusions - GFortran vs. Cray

Cray pros:

- Cray has better performance than GFortran within the same node and on multi node for contiguous array transfers.
- Cray provides the best coverage in terms of coarray functionality.

Cray cons:

- Cray requires some tuning in order to work properly (env vars and modules).
- The Cray compiler requires a Cray supercomputer.

GFortran pros:

- GFortran has better performance than Cray for strided transfers on multi node.
- GFortran works on any architecture able to compile GCC and a standard MPI implementation.
- GFortran is free.

GFortran cons:

- GFortran does not yet cover all the coarray features.

Conclusions

- GFortran provides a stable, easy to use and efficient implementation of coarrays.
- GFortran provides a valid and free alternative to commercial compilers.
- GFortran + LIBCAF_MPI has significant performance improvement/decrease according to the machine and the MPI one sided implementation.

Acknowledgment

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

Thanks

Questions?