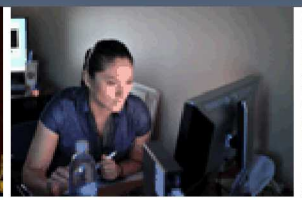
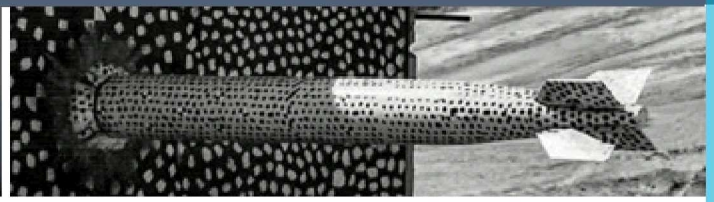
SAND2018-11913PE

Performance Effects of Task Placement on ATS1 and CTS1

Managing MPI Rank and OpenMP Thread Placement with SLURM





PRESENTED BY

Douglas M. Pase, PhD/Sandia National Laboratories

With Anthony M. Agelastos and Joel O. Stevenson

SAND...

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Summary - ATS-1/KNL

- Default placement performs poorly when using MPI and OpenMP together with the Intel programming environment
  - Use --cpu-bind=cores
- Intel gives faster, but less consistent performance vs. CRAY and Gnu
- CRAY and Gnu have similar performance on MPI everywhere
- CRAY is a little faster than Gnu on OpenMP everywhere

Summary - ATS-1/Haswell

- CRAY default placement performs poorly for MPI everywhere codes
  - Use --cpu-bind=cores
- All placements perform poorly for OpenMP everywhere codes
  - Use at least 2 MPI ranks per node
  - Use --cpu-bind=sockets
- Intel outperforms CRAY and Gnu for memory performance

Summary - CTS-1/Broadwell

- Intel outperforms Gnu for memory performance
- Do NOT use OMP_PROC_BIND=“spread”
- Local memory access is about 60% faster than remote

Canary In The Coal Mine

- Throughout this study we use the stream microbenchmark to measure performance because, of the available resources,
- Memory performance is most sensitive to task placement
- Core, MPI and I/O performance are generally next, often in that order
- Depending on how tasks are placed, stream shows performance impact due to both memory affinity and core loading (i.e., load imbalance), but not MPI or I/O

ATS1 (Mutrino) KNL

Three Programming Environments

- Cray PE - PrgEnv-cray/6.0.4
- Gnu PE - PrgEnv-gnu/6.0.4
- Intel PE - PrgEnv-intel/6.0.4

Three Memory Environments

- Main Memory (DDR)

$ sbatch --constraint=knl,quad,flat

$ srun numactl --membind=0 *$exe*

- Cache

$ sbatch --constraint=knl,quad,cache

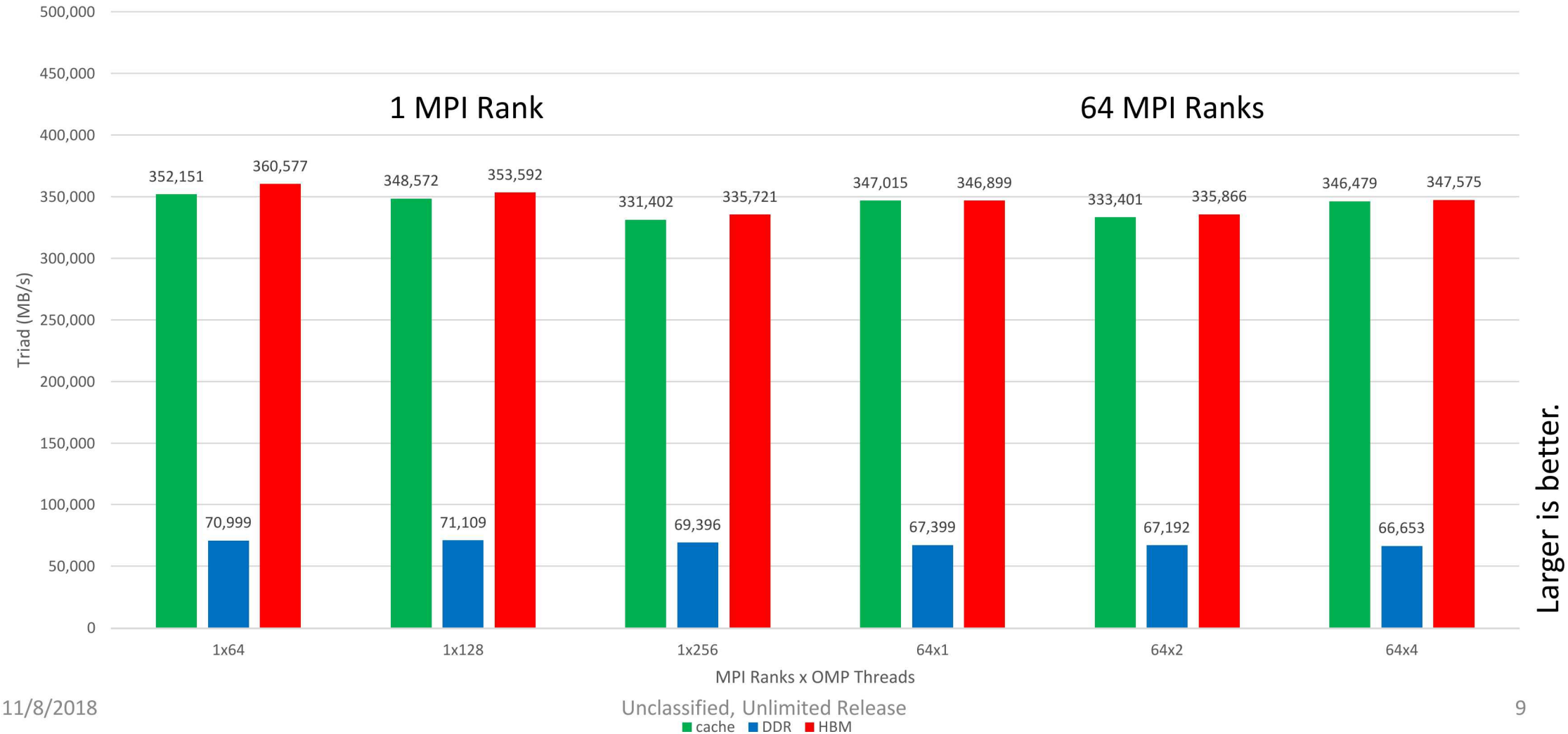
$ srun *$exe*

- High-Bandwidth Memory (HBM)

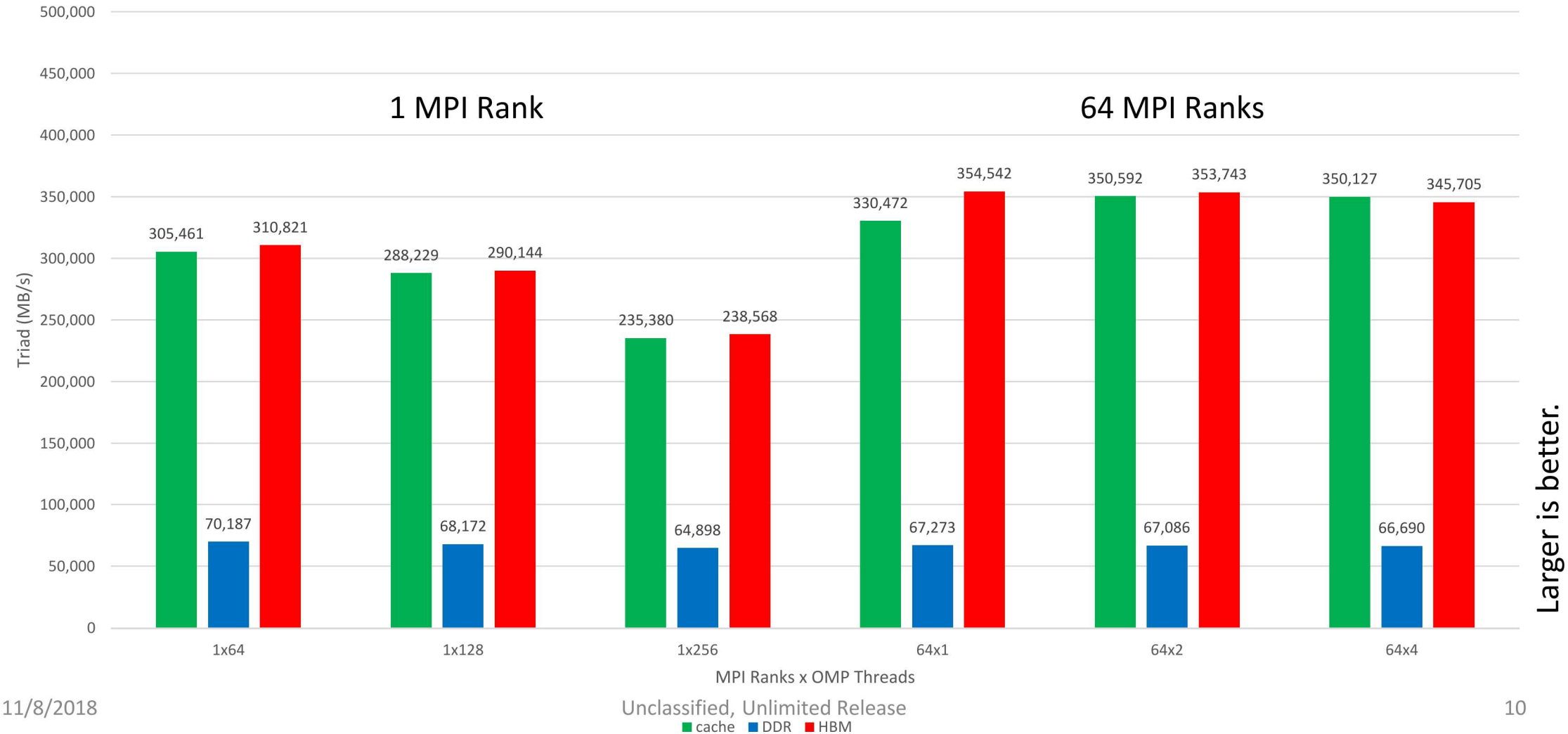
$ sbatch --constraint=knl,quad,flat

$ srun numactl --membind=1 *$exe*

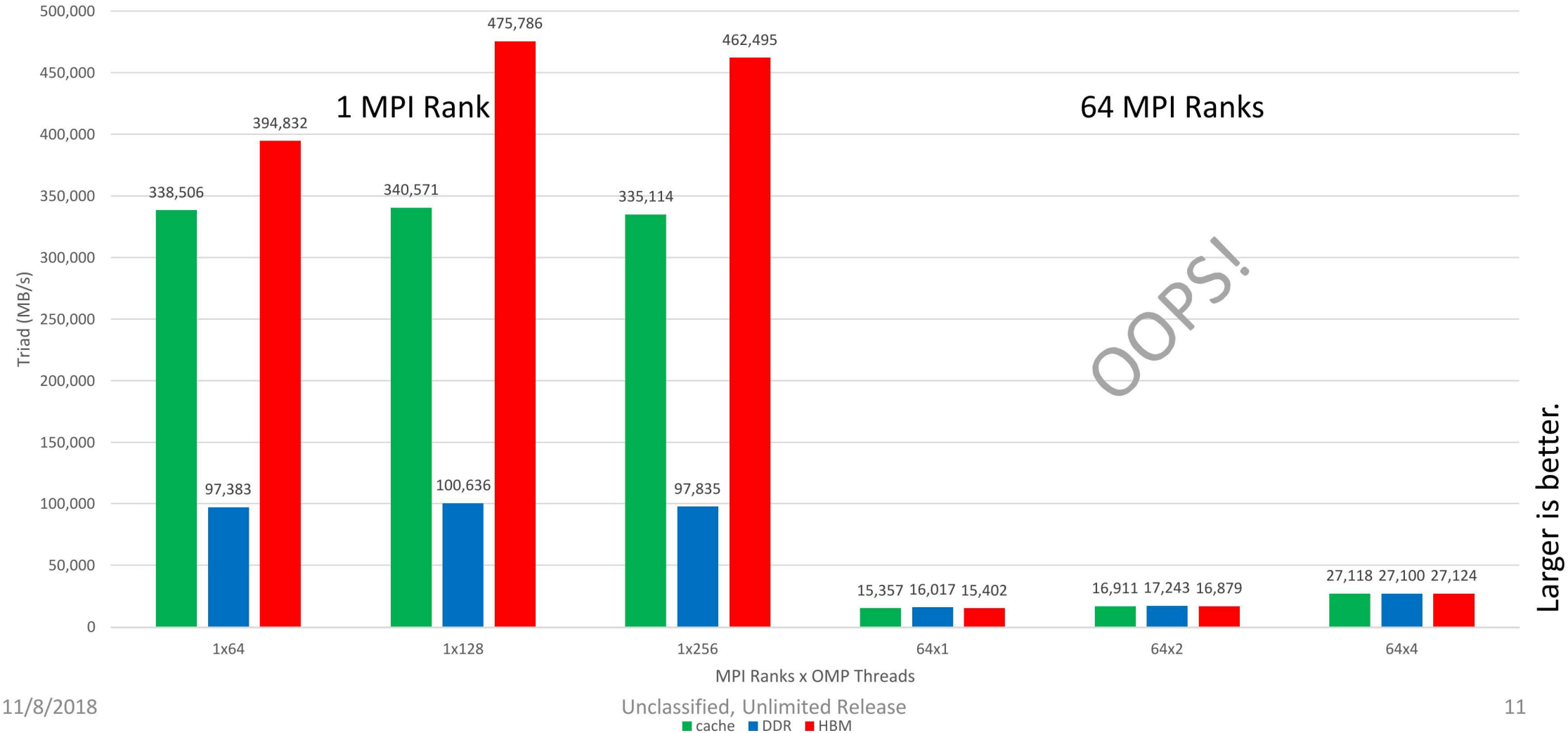
Stream on CRAY PE (Default Rank Placement)



Stream on Gnu PE (Default Rank Placement)



Stream on Intel PE (Default Rank Placement)



What is going wrong?

- Srun is not constraining the binding of ranks to hardware threads
  - The CPU affinity mask allows all hardware threads to be selected
  - This is as it should be, allowing the Linux kernel to manage load balancing across threads and cores
- Intel OpenMP run-time is binding OMP threads to hardware threads in an attempt to gain more performance
  - Each MPI rank has its own collection of OMP threads
  - OMP thread binding is based on the values in the rank CPU affinity mask
  - Each OMP run-time is making the *same* decisions based on the *same* info
  - The result is that each MPI rank is bound to the **same** hardware threads, causing a massive load imbalance

It is a combination of factors…

- The problem is caused by using **all** of the following at the same time:
  - Default (unconstrained) rank binding
  - Multiple MPI ranks per KNL node
  - Intel OpenMP threads
- The fix is to change any **one** (or more) of the above conditions
  - Explicitly bind MPI ranks to disjoint sets of cores/threads, or…
  - Use only a single MPI rank per KNL node, or…
  - Compile applications without Intel OpenMP
    - Setting OMP_NUM_THREADS=1 is **not** enough!
- In other words, either avoid hybrid models of parallel execution, or explicitly bind MPI ranks to disjoint sets of cores

Hierarchy of Controls

- SLURM resource manager (via sbatch) allocates resources
- srun/mpirun/mpiexec
  - Launches programs on reserved resources
  - Maps MPI ranks to nodes, NUMA domains and cores
  - Binds MPI ranks to nodes, NUMA domains and cores
- OMP run-time system may re-map and re-bind ranks and threads
- Linux/TOSS3 kernel executes the requested core/domain bindings
  - srun and OMP run-times request mapping and binding
  - The kernel is responsible for honoring the binding
  - Bindings are communicated via CPU affinity masks
  - CPU affinity masks list the allowed hardware threads for a process or thread

Sbatch Options

- --constraint={knl,quad,flat *or* knl,quad,cache}
- --nodes=*$nodes_per_job*

Srun Options

- --nodes=*$nodes_per_job*
- --ntasks=*$ranks_per_job*
- --ntasks-per-node=*$ranks_per_node*
- --distribution={block;cyclic}
- --cpu-bind=...
  - map_cpu:r_0,r_1,r_2,…,r_{n-1}
  - cpu_mask:0xF...F,0xF...F,...
  - cores
- numactl --membind={0;1} --cpunodebind={1,0} *$exe*

Mpirun and Mpiexec Options

- Default
  - MPI ranks are distributed BLOCK
  - All slots are filled
- --map-by ppr:$ranks_per_node:node
  - Limited to no more than $ranks_per_node ranks on each node
- --rank-by {node,socket,core,hwthread}
  - --rank-by node - MPI ranks are distributed CYCLIC (i.e., round robin)
  - --rank-by socket - MPI ranks are distributed BLOCK
- --bind-to {none,socket,core,hwthread}
  - MPI ranks are allowed to float anywhere (none), within the socket (socket), within a core (core), or not allowed to float (hwthread)

OpenMP Controls

- export OMP_PROC_BIND=*true*,*false*,*spread*,*close*,*master*
  - *true*: threads should not be moved
  - *false*: threads may be moved
  - *spread*: threads are distributed across partitions (i.e., NUMA domains)
  - *close*: threads are kept close to the master thread, similar to *master*
  - *master*: threads are kept in the same partition as the master thread
- export OMP_PLACES=*threads*,*cores*,*sockets*
  - *threads*: OpenMP threads are placed on successive hardware threads
  - *cores*: OpenMP threads are placed on successive processor cores
  - *sockets*: OpenMP threads are placed on successive sockets

What I used to get the best performance…

- srun
  --nodes=*$nodes_per_job*
    - *Specifies the number of nodes to use, must be less than or equal to the nodes allocated*

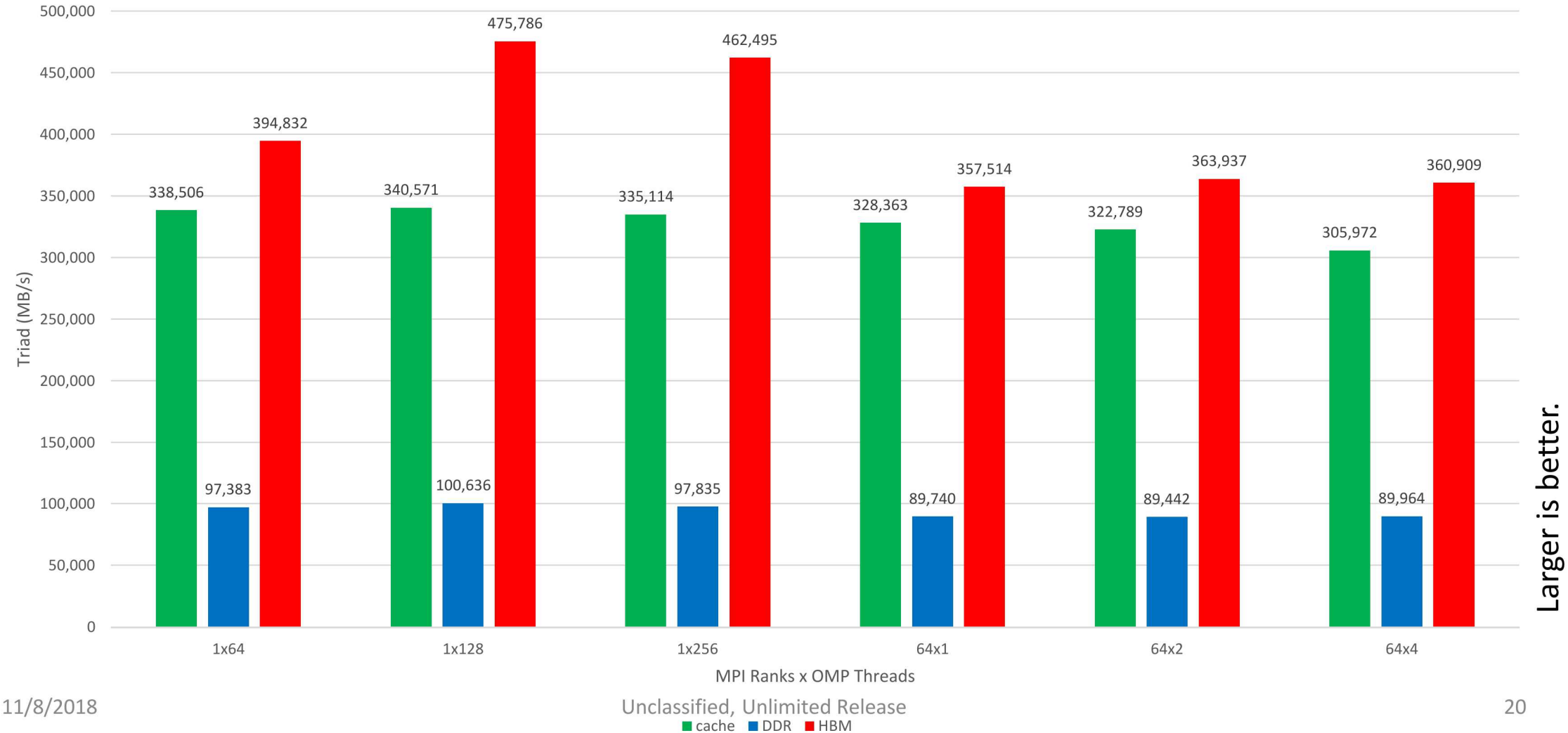
  --ntasks=*$ranks_per_job*
    - *Specifies the total number of MPI ranks to use*

  --ntasks-per-node=*$ranks_per_node*
    - *Specifies the number of ranks to use on each node*

  --cpu-bind=cores

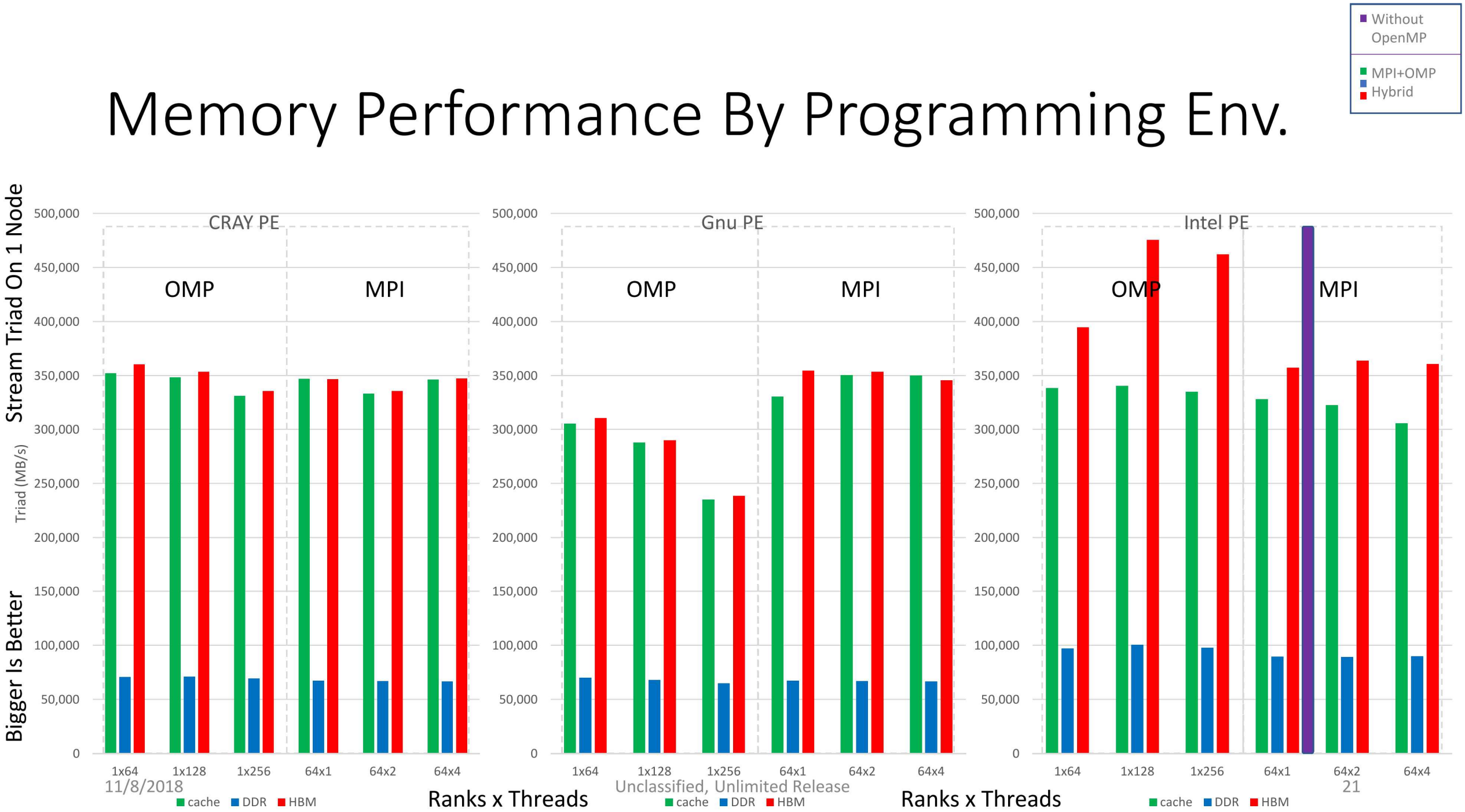
  numactl --membind=1 *$exe*

Stream on Intel PE (--cpu-bind=cores)



Larger is better.

Memory Performance By Programming Env.



Combinations of Parallelism

- OpenMP Everywhere - 1 MPI rank; 68, 136, 272 OMP threads / rank
- Hemisphere - 2 MPI ranks; 34, 68, 136 OpenMP threads per rank
- Quadrant - 4 MPI ranks; 17, 34, 68 OpenMP threads per rank
- Octant - 8 MPI ranks; 8, 17, 34 OpenMPI threads per rank
- x17 - 17 MPI ranks; 4, 8, 17 OpenMPI threads per rank
- x34 - 34 MPI ranks; 2, 4, 8 OpenMPI threads per rank
- MPI Everywhere - 68 MPI ranks; 1, 2, 4 OpenMP threads per rank

Using Combinations of Parallelism

- OpenMP Everywhere (default)
- MPI Everywhere (--cpu-mask=cores)
- Hemisphere (x2) --cpu-mask=

  0x00000003ffffffff00000003ffffffff00000003ffffffff00000003ffffffff,
  0xffffffffc00000000ffffffffc00000000ffffffffc00000000ffffffffc00000000

- Quadrant (x4) --cpu-mask=

  0x000000000001ffff000000000001ffff000000000001ffff000000000001ffff,
  0x000000003fffe000000000003fffe000000000003fffe000000000003fffe0000,
  0x00007fffc000000000007fffc000000000007fffc000000000007fffc00000000,
  0xffff8000000000000ffff8000000000000ffff8000000000000ffff800000000000

Other Combinations of Parallelism

- Octant (x8) --cpu-mask=

```
0x00000000000000ff00000000000000ff00000000000000ff00000000000000ff,
0x0000000000000ff00000000000000ff00000000000000ff00000000000000ff00,
...
0x000ff00000000000000ff00000000000000ff00000000000000ff00000000000,
0x0ff00000000000000ff00000000000000ff00000000000000ff00000000000000
```

- x17 --cpu-mask=

```
0x0000000000000000f0000000000000000f0000000000000000f0000000000000000f,
0x0000000000000000f0000000000000000f0000000000000000f0000000000000000f0,
...
0x0f0000000000000000f0000000000000000f0000000000000000f0000000000000000,
0xf0000000000000000f0000000000000000f0000000000000000f0000000000000000
```

- x34 --cpu-mask=

```
0x0000000000000000300000000000000003000000000000000300000000000000003,
0x0000000000000000c00000000000000000c00000000000000c00000000000000000c,
...
0x3000000000000000300000000000000003000000000000000300000000000000000,
0xc00000000000000000c00000000000000c00000000000000c00000000000000000
```

8

17

34

ATS1 (Mutrino) Haswell

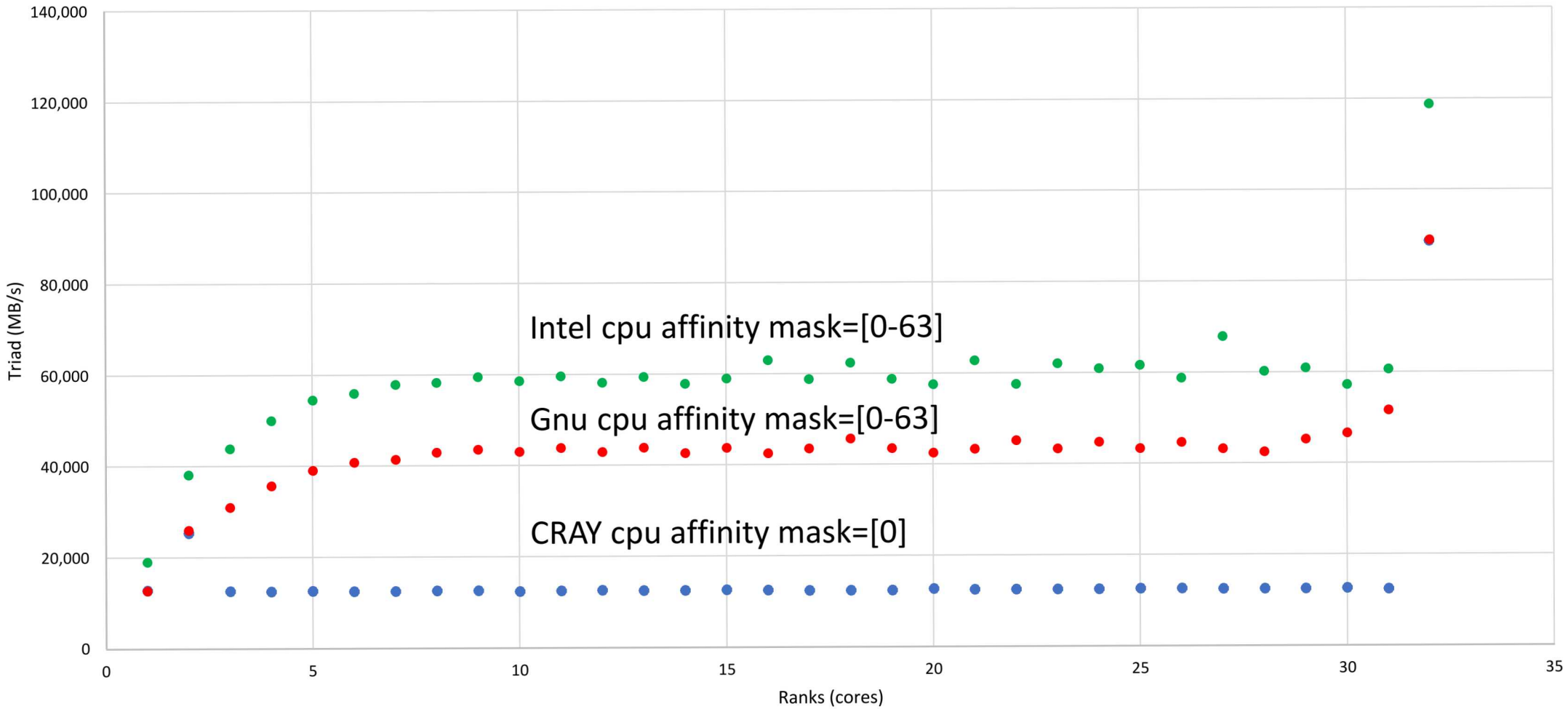
Three Programming Environments

- Cray PE - PrgEnv-cray/6.0.4
- Gnu PE - PrgEnv-gnu/6.0.4
- Intel PE - PrgEnv-intel/6.0.4

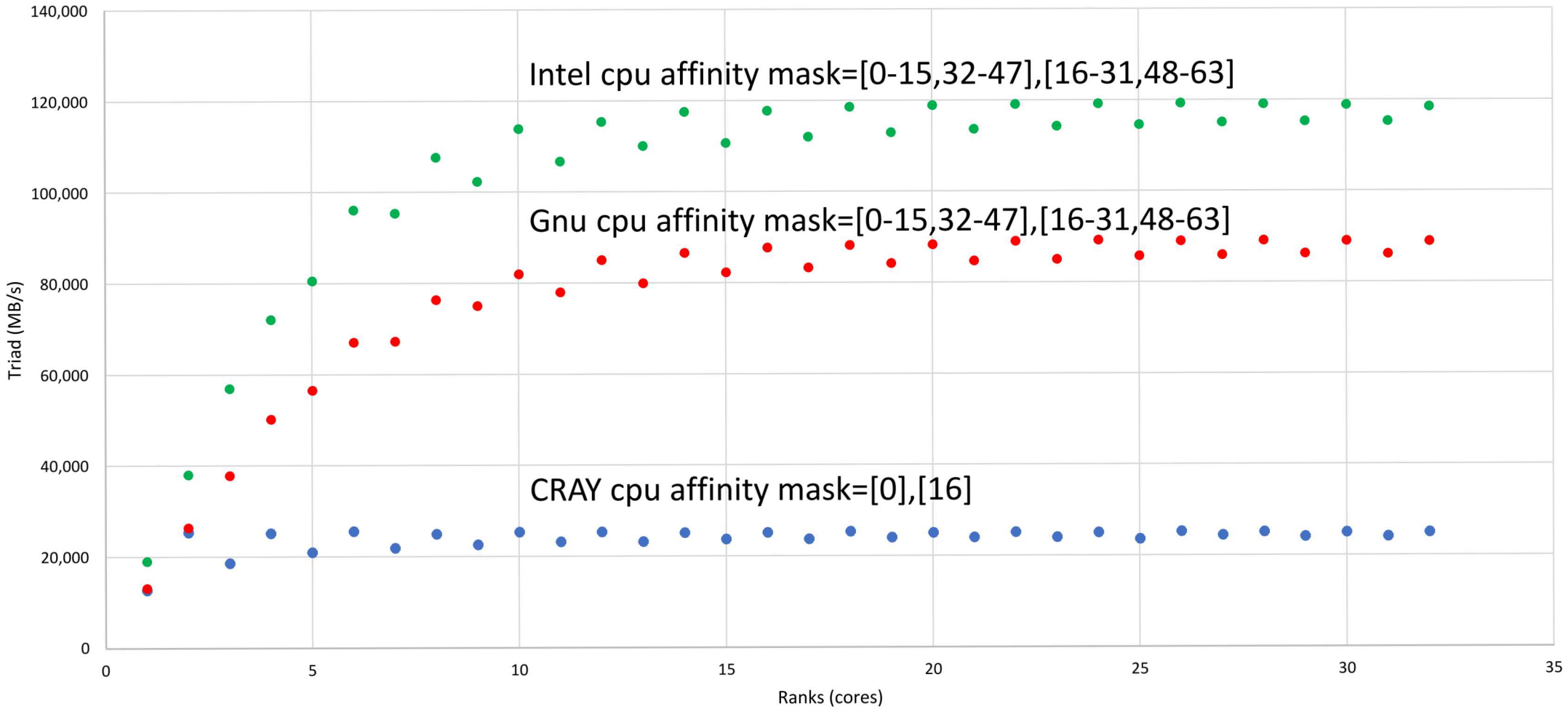
...

- And two memory domains

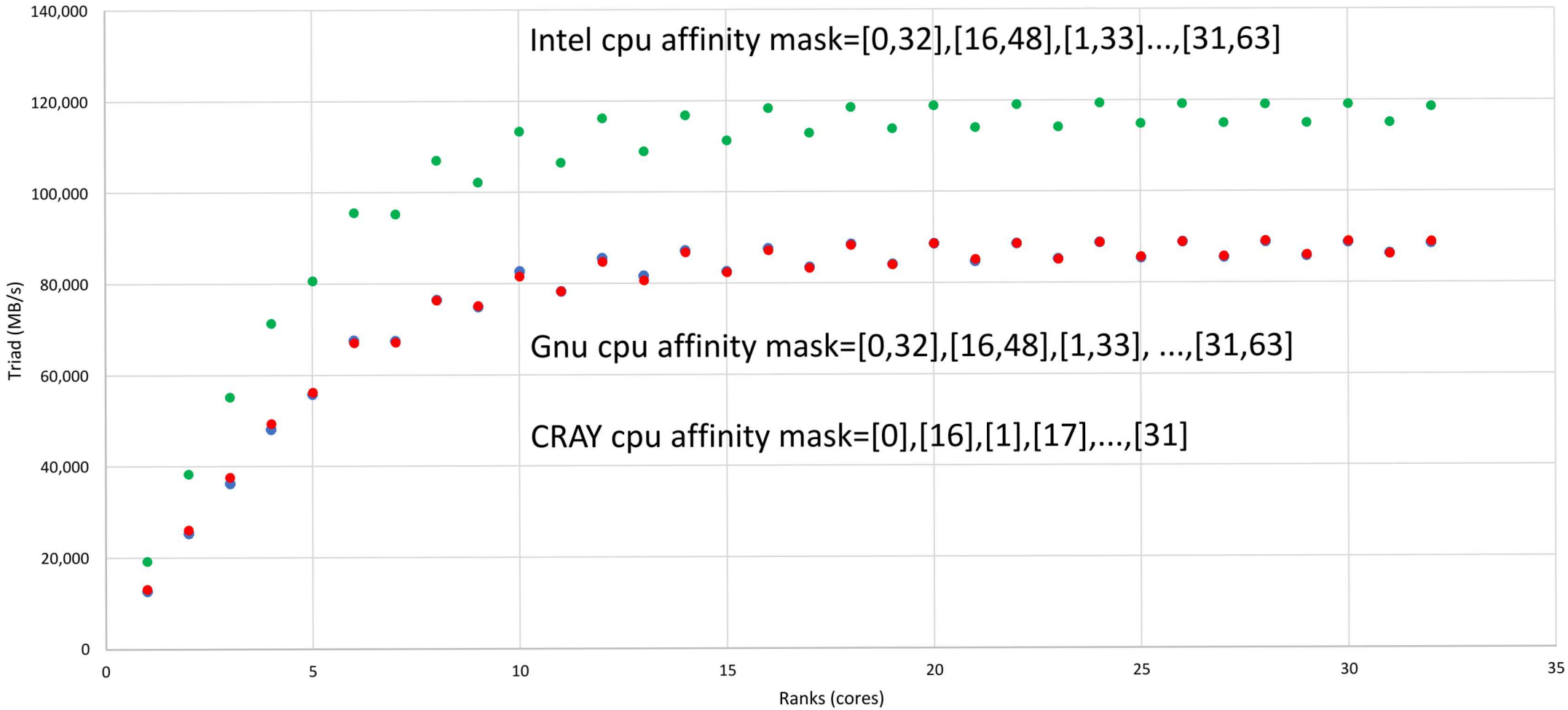
Default Distribution and Bindings (MPI)



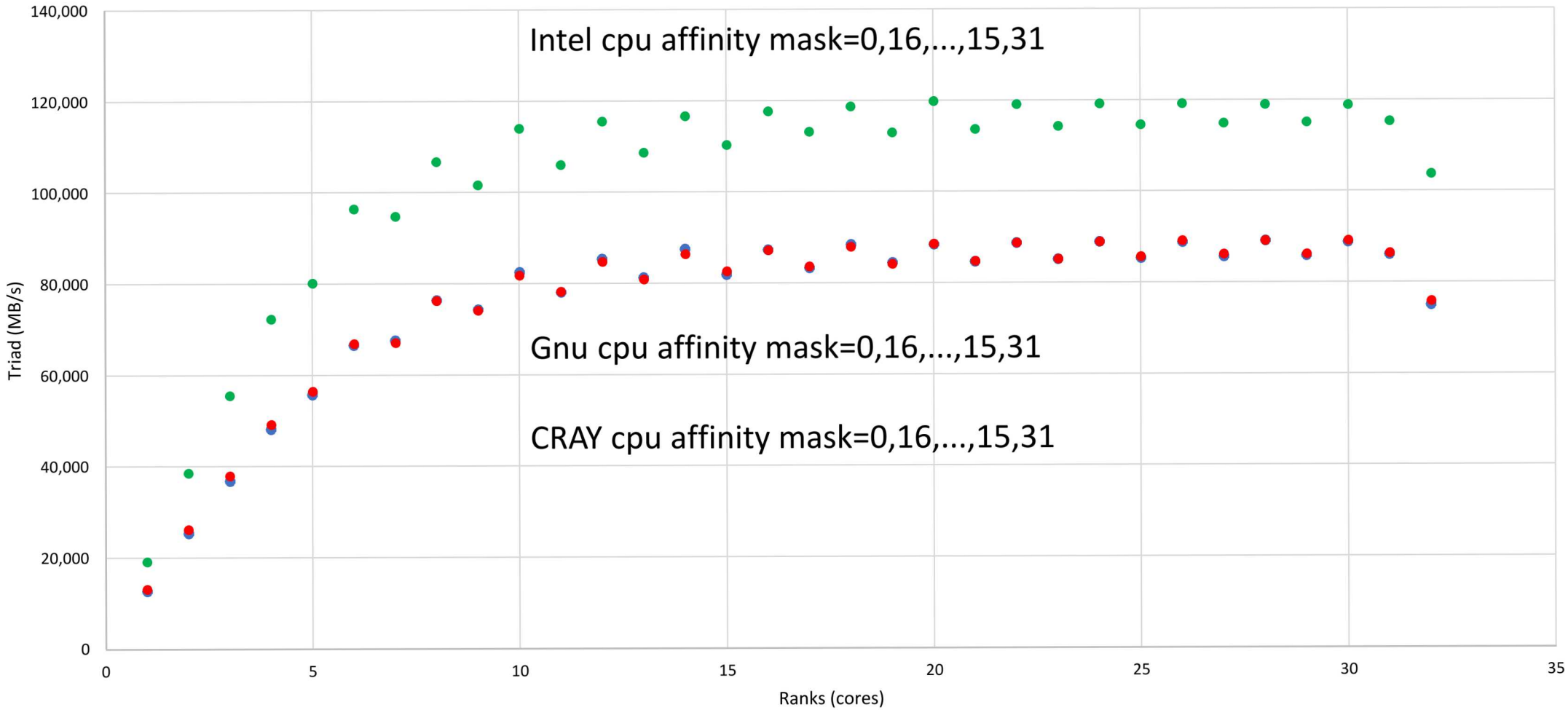
--cpu-bind=sockets (MPI)



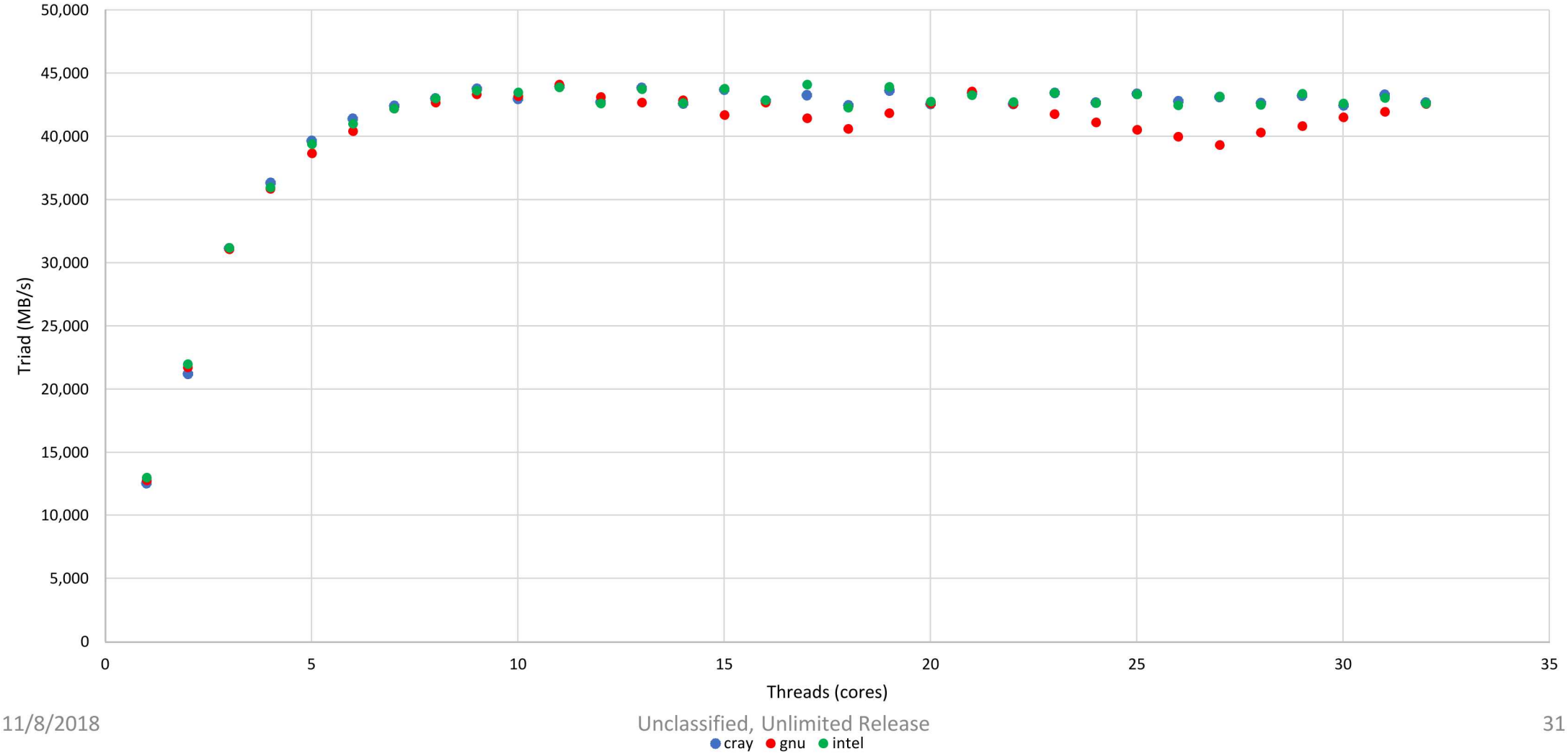
--cpu-bind=cores (MPI)



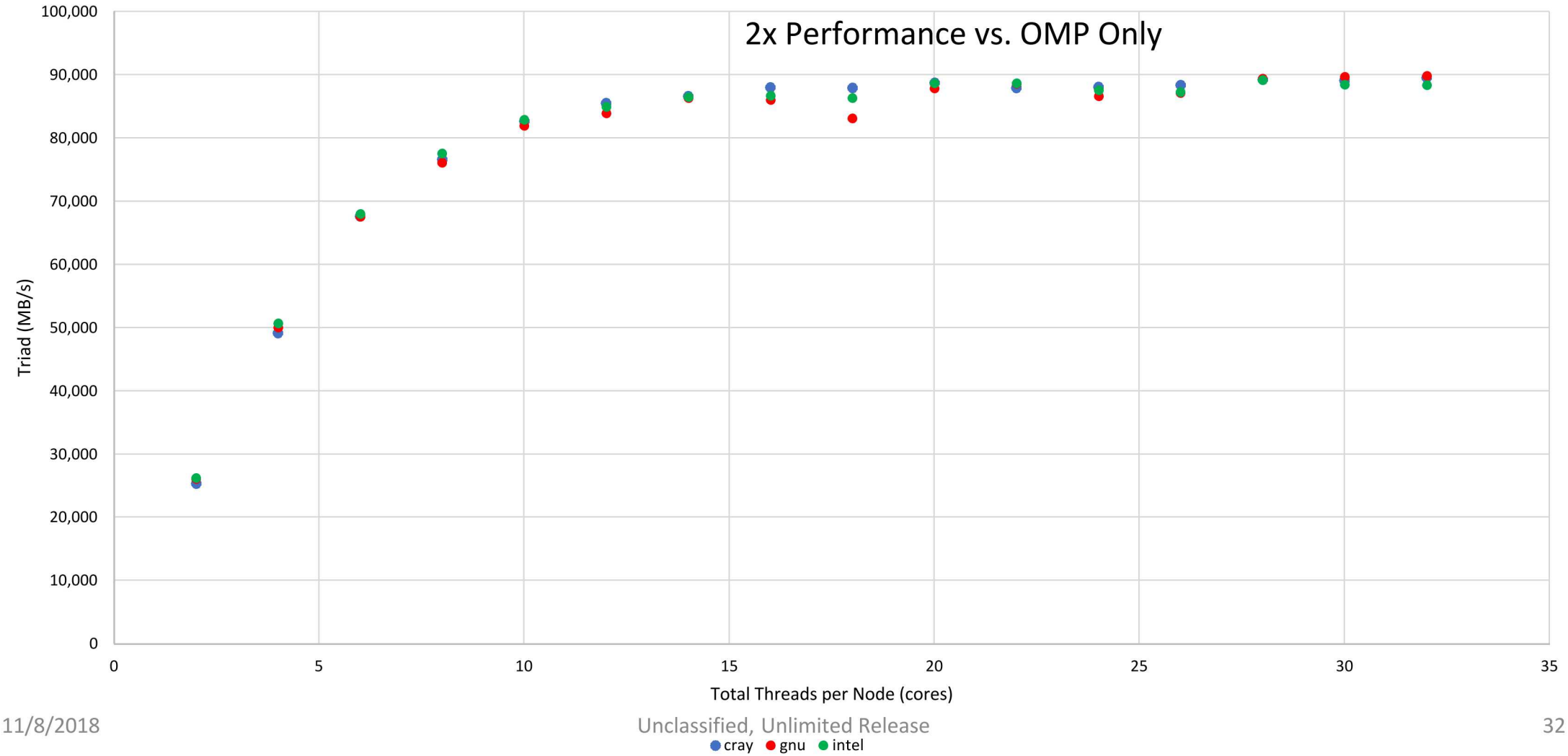
--cpu-bind=map_cpu:0,16,1,...,30,15,31 (MPI)



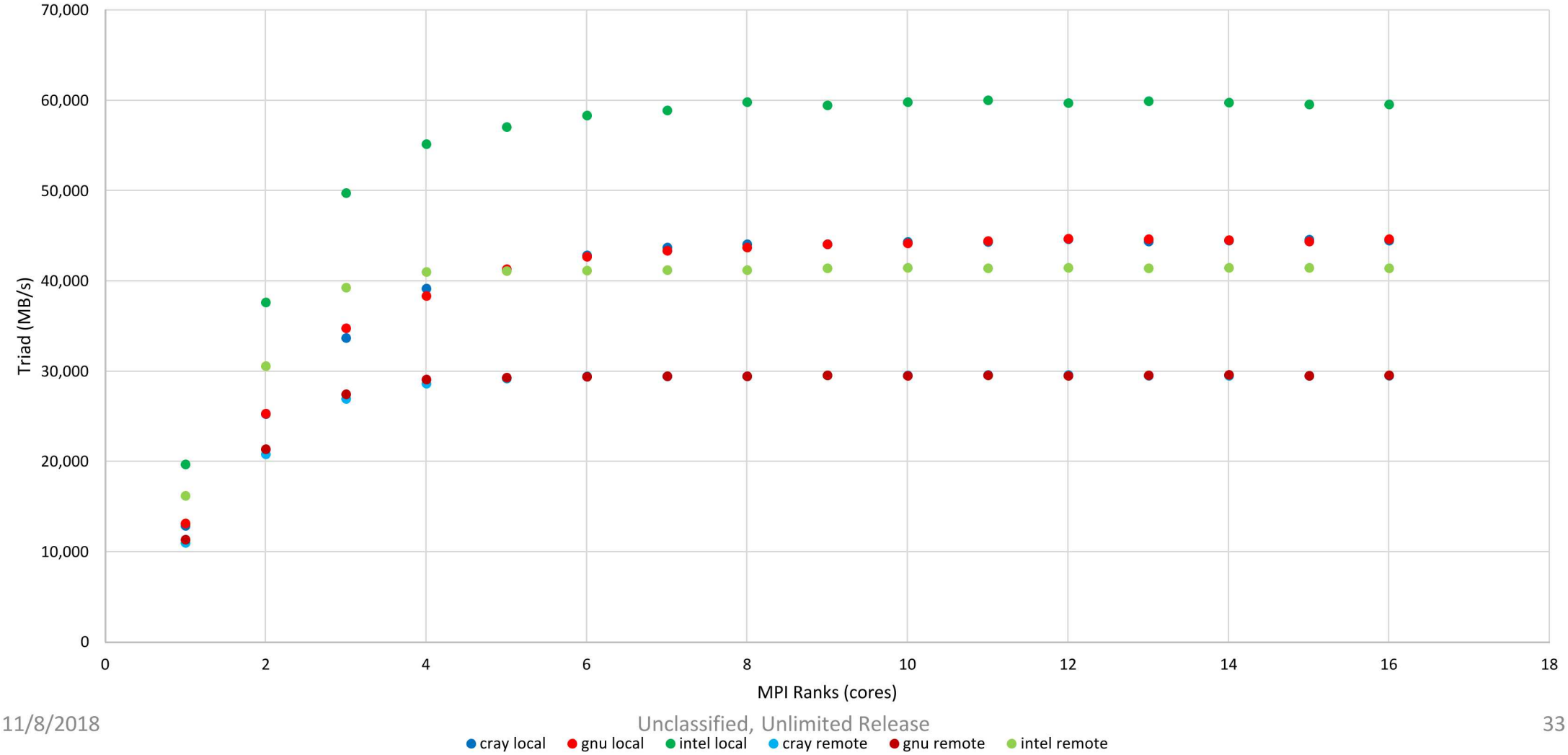
Threads Only - (1 Rank, *k* Threads per Node)



2 Ranks, *k* Threads per Node



Single Socket Local and Remote (MPI)

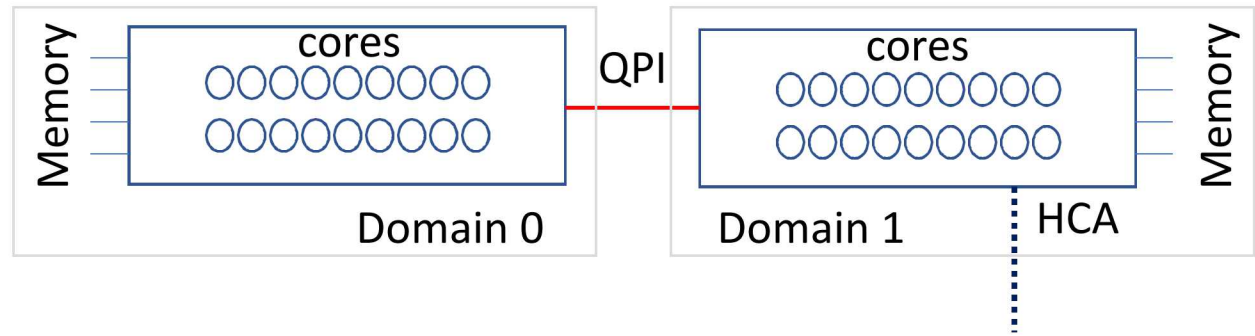


CTS1 (Eclipse) Broadwell

Two Programming Environments

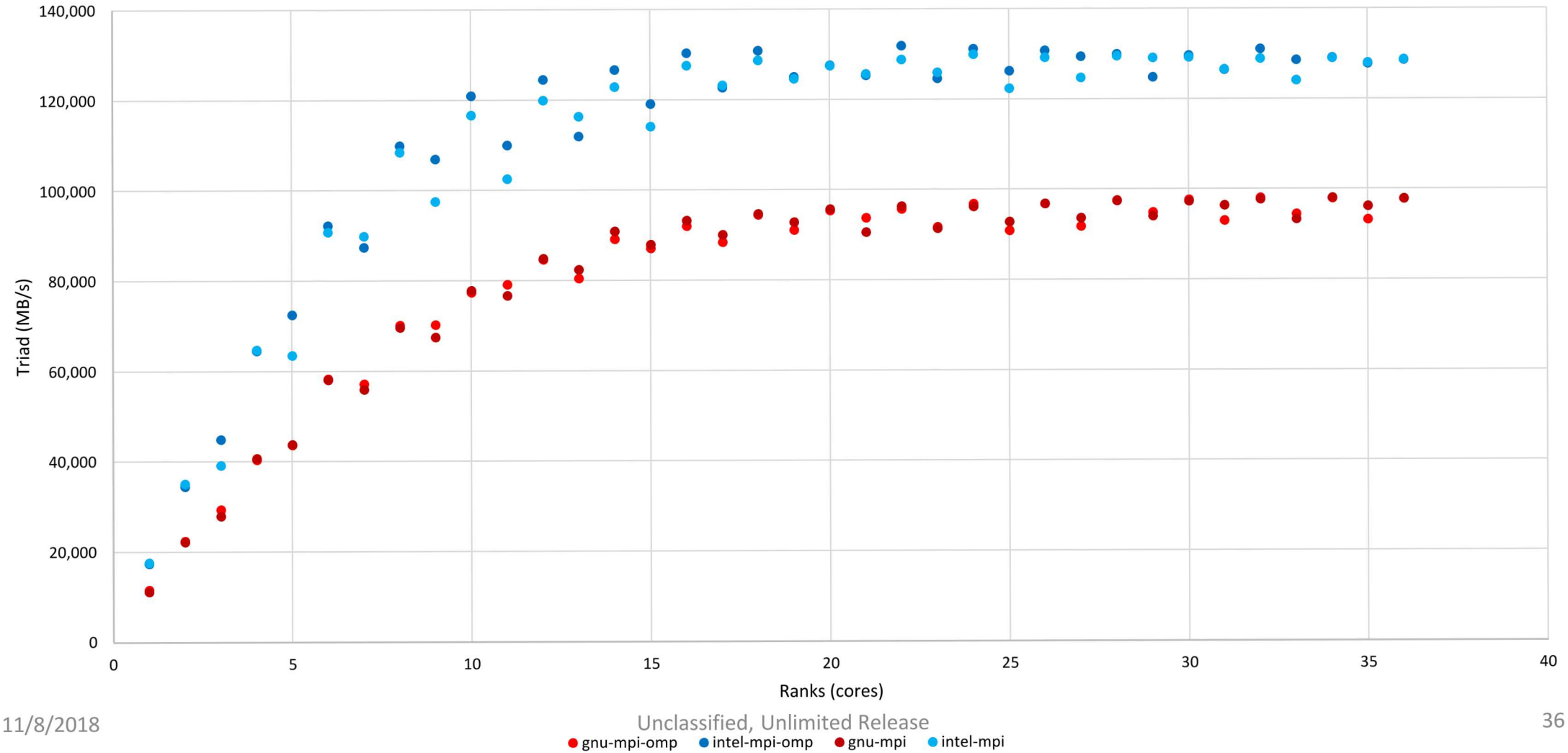
- Gnu - we used gnu/7.3.1
- Intel - we used intel/18.0

...

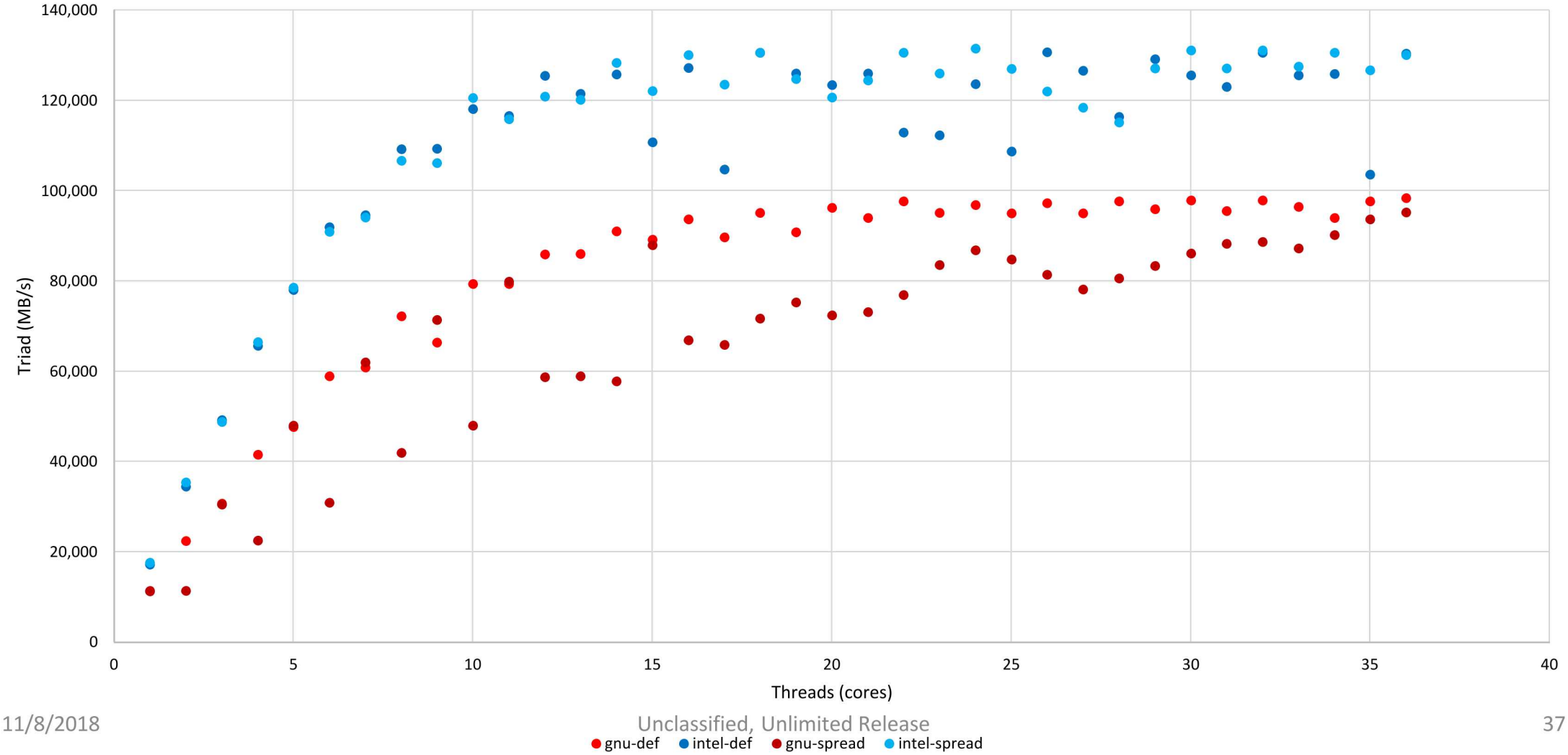


- And two memory domains (local and remote)
  - Memory in domain 0 is local to cores in domain 0
  - Memory in domain 1 is local to cores in domain 1
  - Memory in domain 0 is remote to cores in domain 1, and vice versa

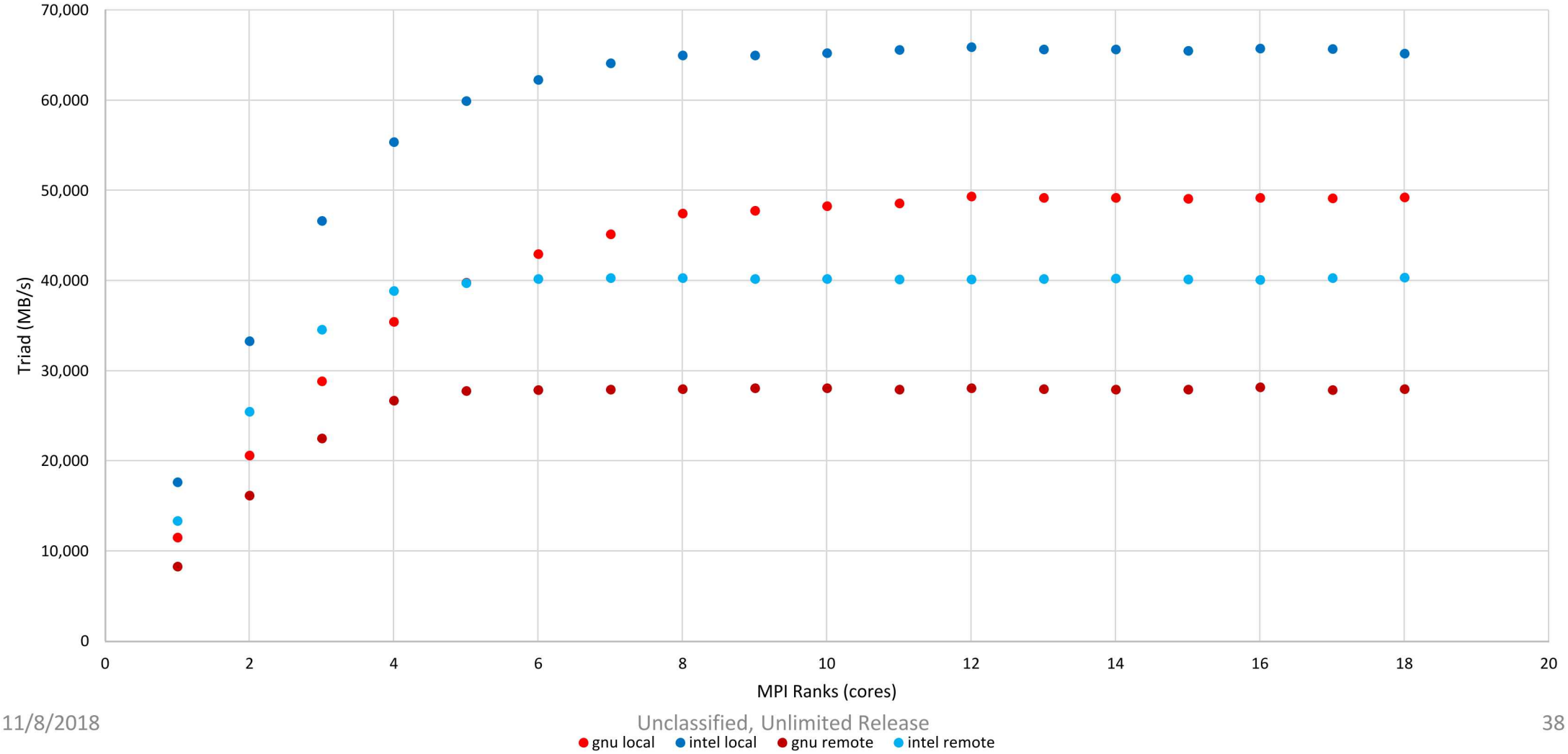
Default Placement (MPI)



Threads With Default and Spread (OMP)



Single Socket Local and Remote (MPI)



Questions?