Evaluating New Communication Models in the Nek5000 Code for Exascale

Ilya Ivanov (KTH), Rui Machado (Fraunhofer), Mirko Rahn (Fraunhofer), Dana Akhmetova (KTH), Erwin Laure (KTH), Jing Gong (KTH), Philipp Schlatter (KTH), Dan Henningson (KTH), Paul Fischer (ANL), Stefano Markidis (KTH)

EPiGRAM

Exascale ProGRAmming Models

Outline

- Parallelism, Programming Models and Legacy Applications at Exascale
- Nek5000 code
- Parallel Communication in Nek5000
  - Gather-Scatter Communication Operator
  - New MPI Gather-Scatter Communication Operator
  - New PGAS Gather-Scatter Communication Operator
- Conclusions

Parallelism at Exascale

- Tianhe-2 (#1 in Top500) has 3,120,000 cores in total, much larger number of processes at exascale → billion of processes.
- At exascale:
    - Interconnection networks with higher performance (smaller latency, larger bandwidth)
    - Emerging new network topologies, i.e. Slim Fly, and Cray Dragonfly
    - Support on NIC for communication operations without intervention of CPU

EPiGRAM

Programming Models at Exascale

- Existing: parallel programming models are designed for tera and peta-scale eras.
- Good news: parallel programming models are being equipped with new features to effectively exploit exascale technology:
  - One-sided communication → use communication support from network
  - Non-blocking collectives → more asynchronous model, i.e. in linear solvers
  - Neighborhood collectives → use smart scheduler for communication on small group of processes
- PROBLEM: How many applications at EASC2015 use new features in programming models?

EPiGRAM

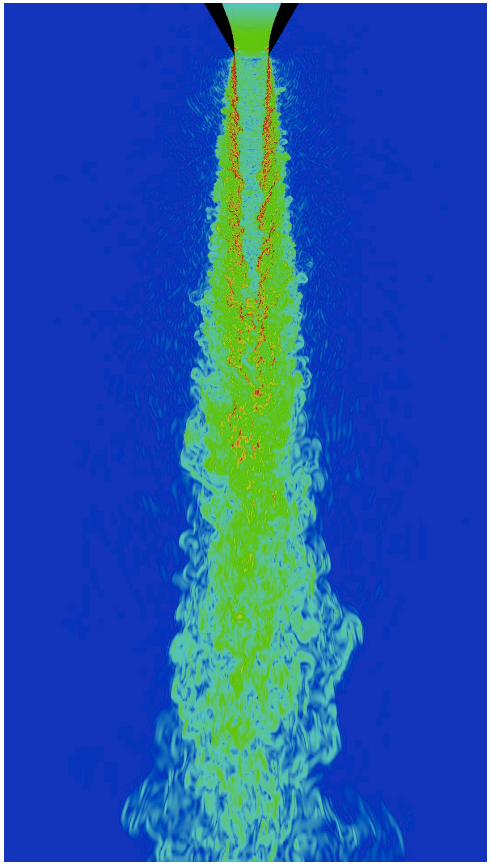
Problem: Legacy Applications

- Communication in legacy codes was designed to allow for features that were not present during the initial development of the code, i.e. thousands of LOC for non-blocking communication.
- Development went so far that there is no turning back point. Communication code is so complex that it is very difficult to add new features.
- We need disruptive changes in codes to use new features in programming models.

EPiGRAM

Nek5000 (our legacy applications)

- Nek5000 is a CFD code for the simulation of incompressible fluids. Nek5000 is used for reactor thermal hydraulics, astrophysics, combustion, oceanography, vascular flow modeling.
- It was developed in 80s and consists of 70,000 lines of code: 90% in Fortran77 (computation) and 10% in C (to handle communication).
- Communication implements halo exchange with 3 different algorithm with non-blocking MPI p2p comm (MPI1)
- Nek5000 communication kernel is very complex and obfuscated → very difficult to use new features in programming models



EPiGRAM

Our Disruptive Change in Nek5000

- We designed a new communication kernel for Cartesian topology and structured grids.
- 7,000 loc → 500 loc → Code readability
- C, MPI C Bindings → Fortran, MPI Fortran Bindings → removed interoperability issue between C and Fortran
- No virtual topology → Cartesian virtual topology → neighborhood collectives possible in Nek5000

"All I'm saying is now is the time to develop the technology to deflect an asteroid."

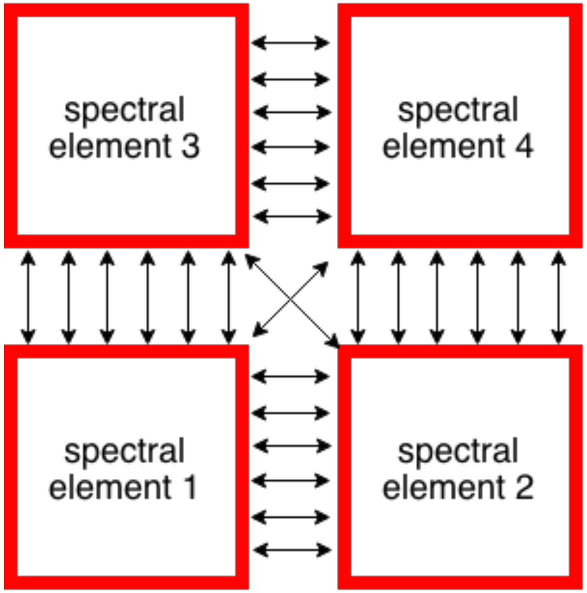
EPiGRAM

Parallel Communication in Nek5000

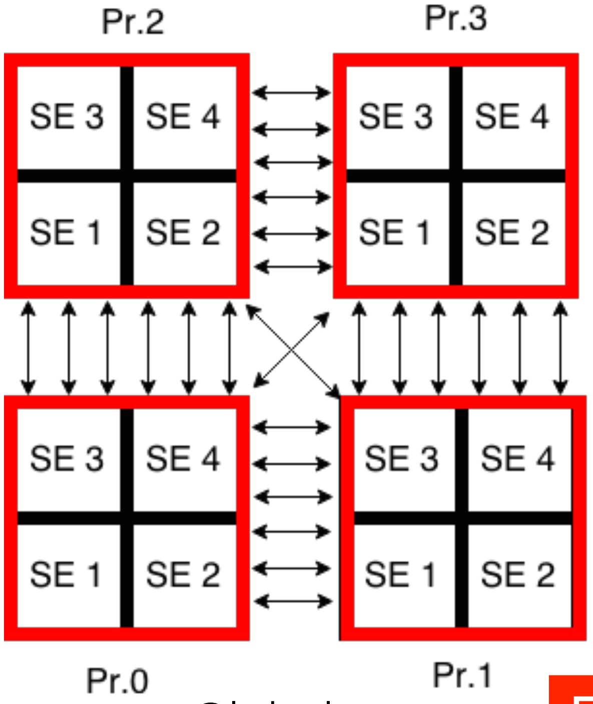
- Global reductions in the CG linear solver, i.e. calculation of inner products of auxiliary vectors.
- Point-to-point communication for a "halo exchange" in the so called gather-scatter operator
- Three old gather-scatter operator algorithms in Nek5000:
  - Pairwise (used for our comparison and fastest one)
  - Chrystal router
  - Allreduce

Gather-Scatter Communication Operator

- It is for communication of spectral element interface values:
  - values on shared (by elements on different processes) nodes have to be consistent



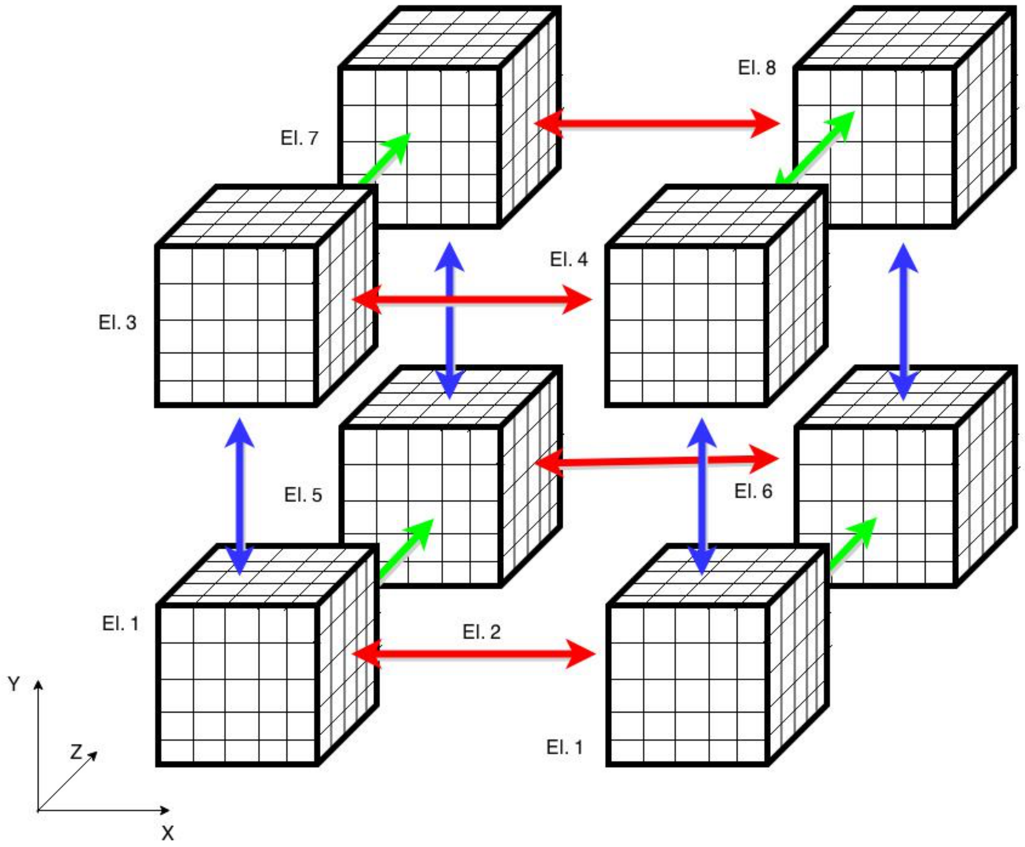
Local gs_op



Global gs_op

New Gather-Scatter Communication Operator

- Designed for Cartesian topology and uniform grids

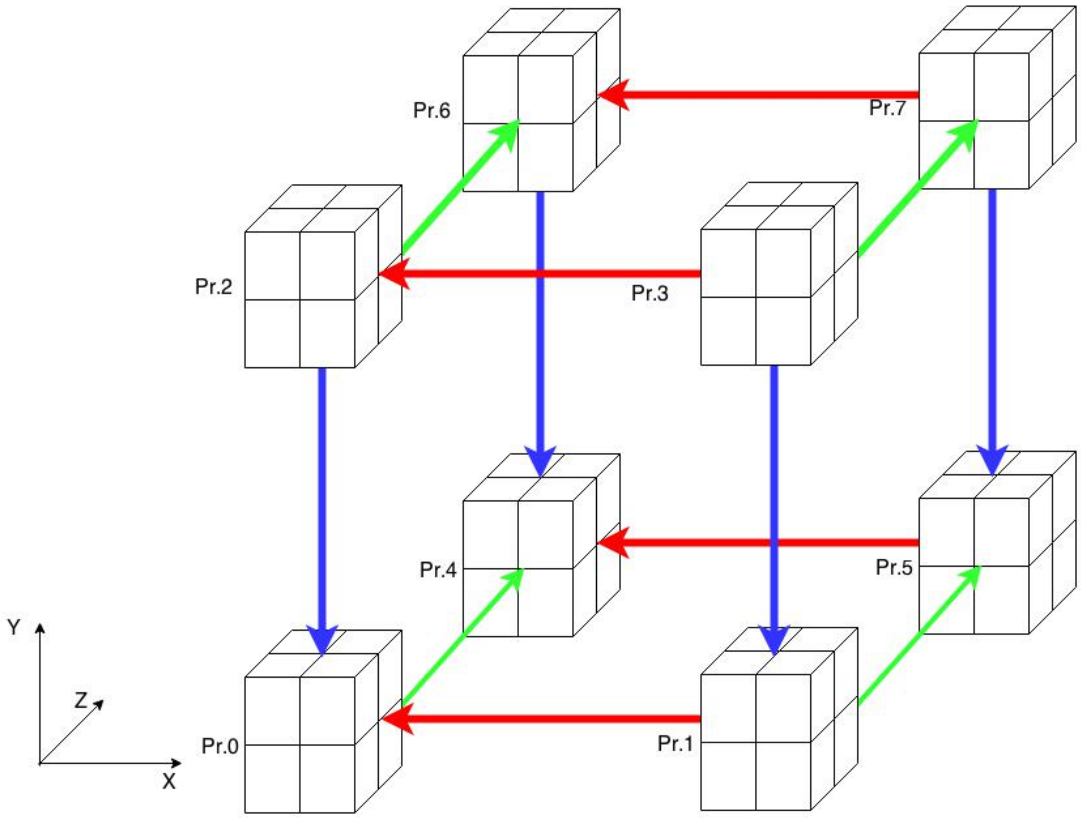


- Local gs_op
  - synchronization between elements on one process
  - synchronization of all boundary points except those that are neighbors of elements on other processes
  - gs_op along X, Y, Z directions

EPiGRAM

New Gather-Scatter Communication Operator

- Designed for Cartesian topology and uniform grids



- Global gs_op
  - synchronization of shared points between elements located on different processes
  - gs_op along X, Y, Z directions

EPiGRAM

MPI Implementation

- New implementation uses Cartesian virtual topology
- Uses MPI blocking point-to-point communication (next step: non-blocking and neighborhood collectives)
- Old one uses non-blocking point to point collectives.

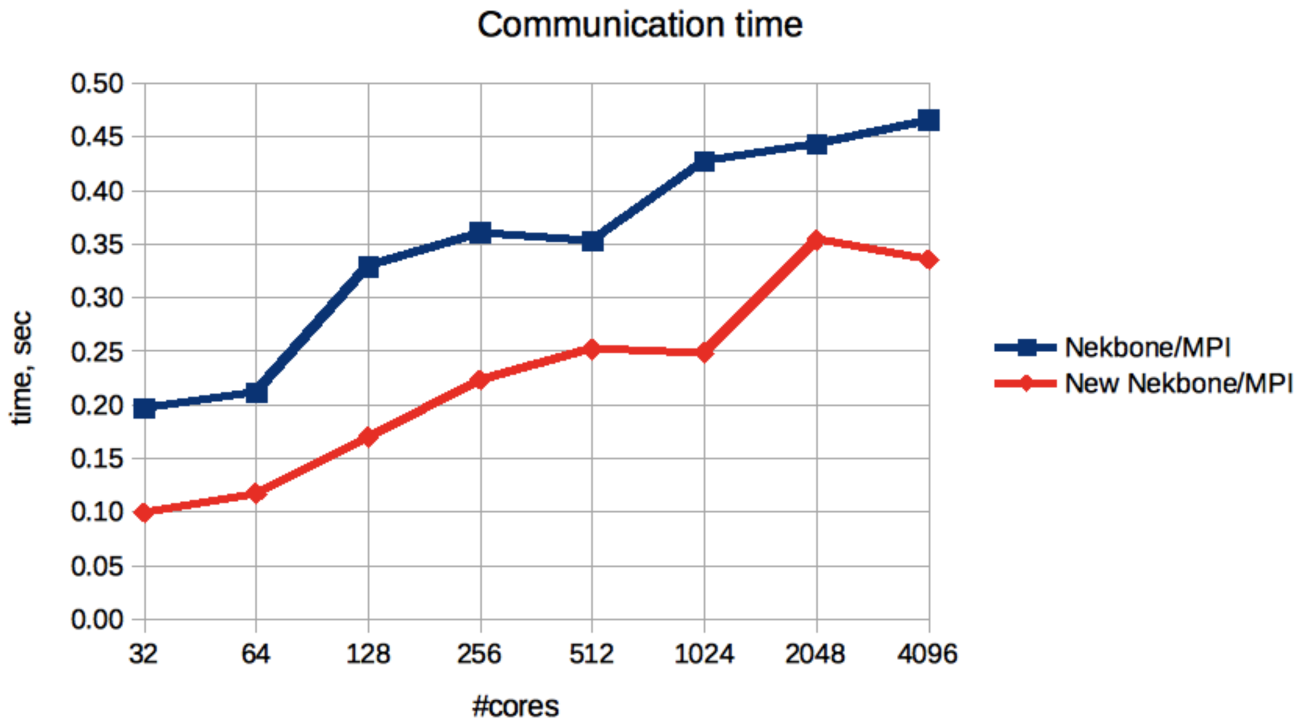
Test Environment

Beskow supercomputer at PDC, KTH:

- Cray XC40 system, Cray Aries interconnect;
- Cray Fortran77 and C/C++ compilers of version 5.2.40;
- Cray MPICH2 of version 7.0.4;
- Nekbone – skeleton version of Nek5000, with the same communication kernel

EPiGRAM

Comparison between Old and New MPI Communication in Nek5000/Nekbone (weak scaling)

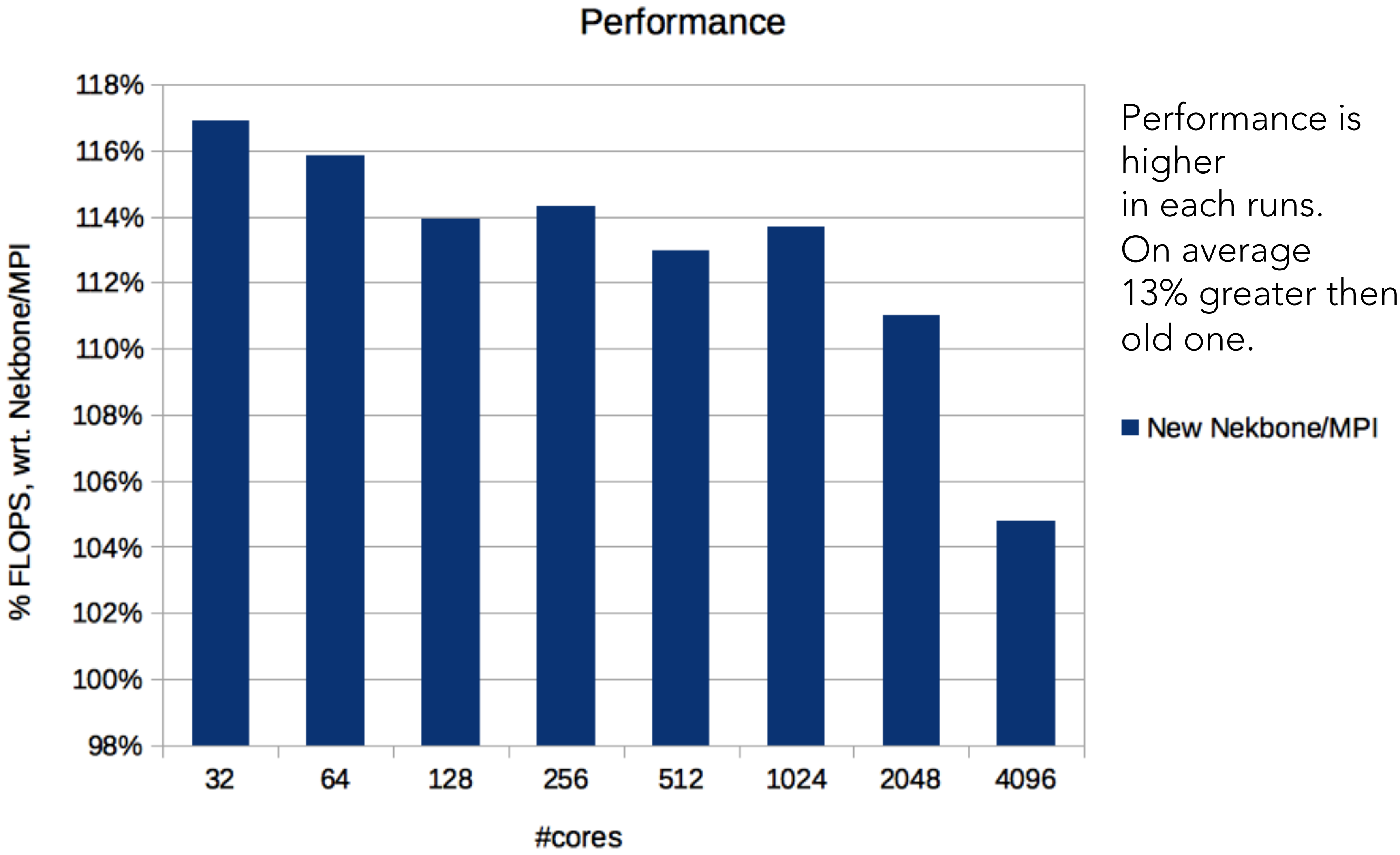


The new Nekbone implementation is always faster than old. On average 37% faster.
This is expected as old Nekbone is designed for complex geometry.

1 MPI process per core
polynomial order – 10
# spectral elements per core - 256

EPiGRAM

Comparison between Old and New MPI Communication in Nek5000/Nekbone (weak scaling)



Performance is
higher
in each runs.
On average
13% greater then
old one.

PGAS, GASPI and GPI-2

- Partitioned Global Address Space
  - Global memory space that is accessible for all the processes
  - One-sided communication (very fast when supported by network)
- GPI-2
  - Implementation of the GASPI standard of a PGAS API
  - Developed by Fraunhofer Institute for Industrial Mathematics ITWM

EPiGRAM

GPI-2 Implementation

- Segment is a contiguous block of virtual memory. Segments may be globally accessible from every thread of every GPI-2 process.
- One-sided asynchronous communication: GPI-2 process specifies all communication parameters, both for the local and the remote side.
- GPI-2 offers the possibility to use different queues to handle communication requests.

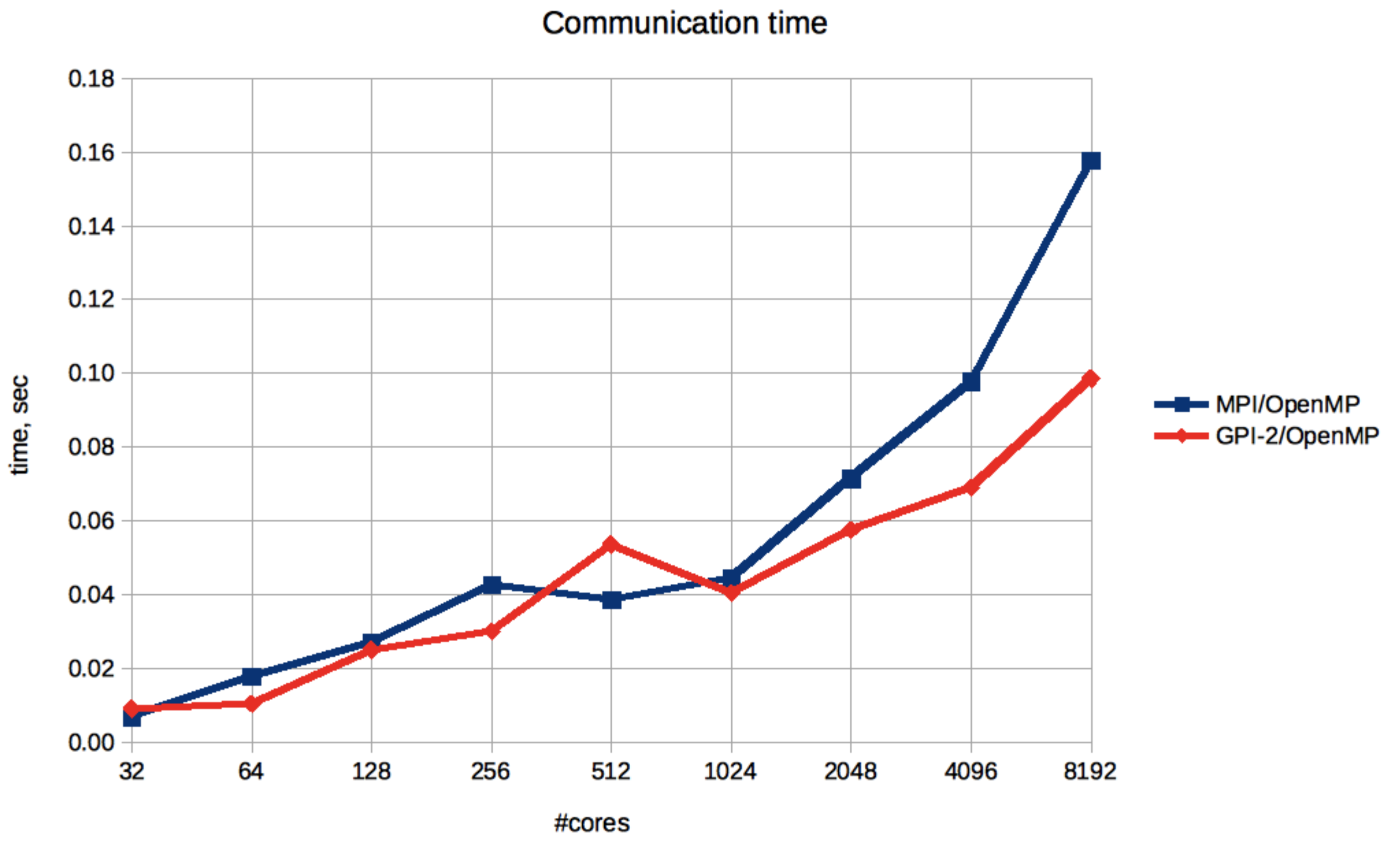
GPI-2 Implementation

- OpenMP directives are used in loops when evaluating A*x for the spectral elements, also in packing and averaging interface values, e.g.:

```
!$OMP PARALLEL DO PRIVATE(e,ur,us,ut,wk) SHARED(nelt,w,u,gxyz)
!$OMP& SCHEDULE(STATIC)
      do e=1,nelt
         call ax_e( w(1,e),u(1,e),gxyz(1,1,e),ur,us,ut,wk)
      enddo
!$OMP END PARALLEL DO
```

- Asynchronous communication allowed to overlap between computation + packing data and communication.

Comparison between New MPI and GPI-2 Communication in Nek5000/Nekbone (weak scaling)

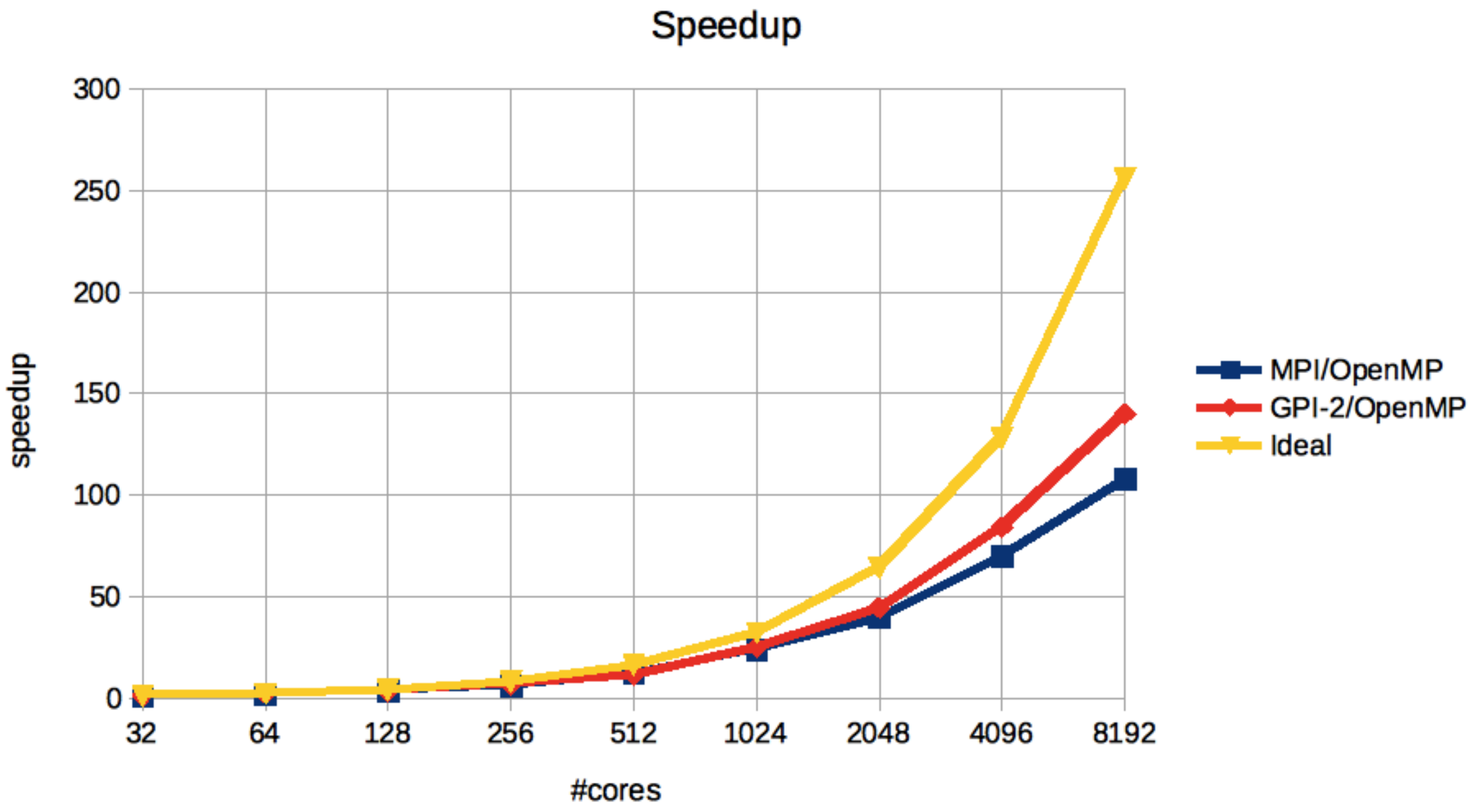


GPI-2 version is always faster, except for 512 cores.
At 8192 cores GPI-2 is 60% faster than MPI.
Fair comparison would be with non-blocking MPI (in future).

1 GPI-2/MPI process per socket (16 cores)
polynomial order – 10
# spectral elements per process - 128

EPiGRAM

Comparison between New MPI and GPI-2 Communication in Nek5000/Nekbone (weak scaling)



At 8192 cores GPI-2 is 39% faster than MPI and 45% lower than ideal scaling, MPI – 58% lower than ideal scaling. Could be improved by adding more OpenMP directives to computations.

EPiGRAM

Conclusions

- Going to exascale requires disruptive changes.
- New gather-scatter communication operator has approx. 500 lines of code, while the old one - 7000.
- New communication kernel can be used for co-design work as it is much easier than the old one.
- Communication time was decreased.

EPiGRAM

Thank you!

EPiGRAM