EPiGRAM

Exascale ProGRAmming Models

Towards Exascale Programming Models

HPC 2014

Cetraro

Erwin Laure, Stefano Markidis, KTH

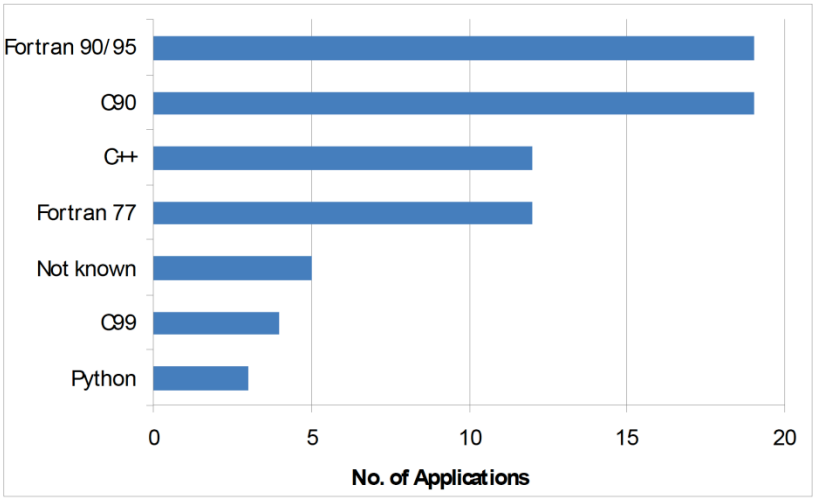
Exascale Programming Models

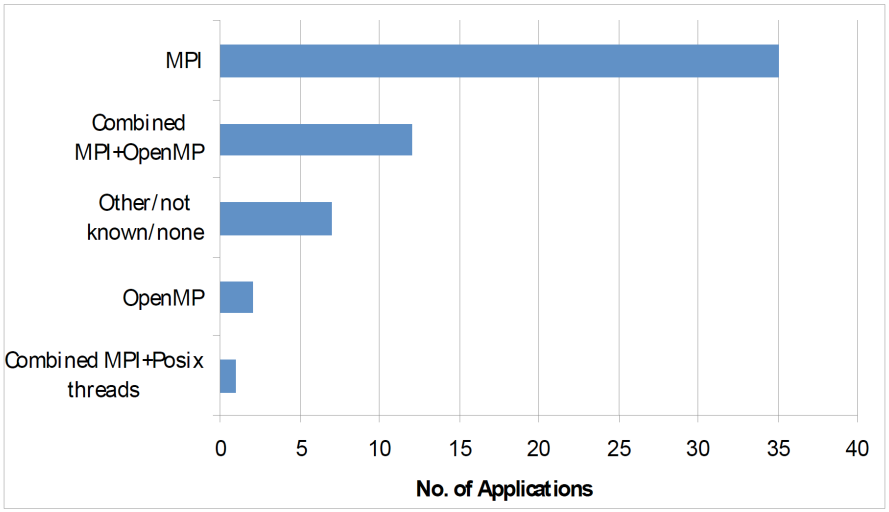
- With the evolution of HPC architecture towards excascale, new approaches for programming these machines need to be found - EPiGRAM focuses on exploring programming models for the exascale era.

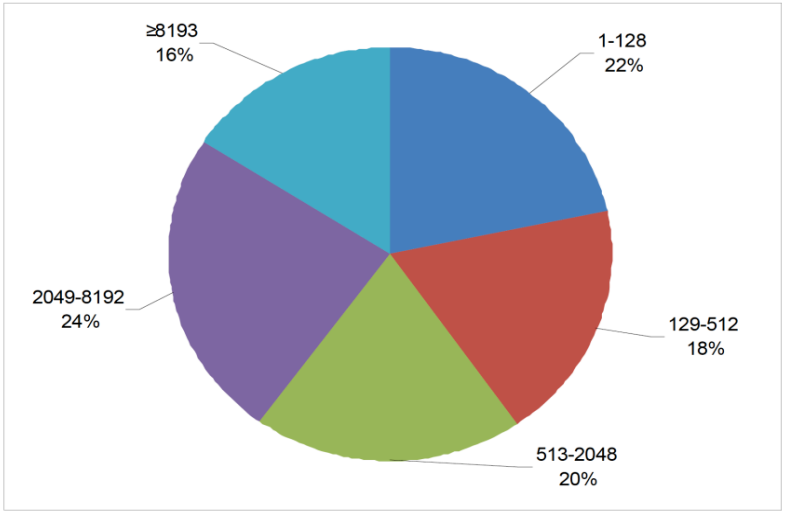
- Intense discussion whether existing models can be improved to exascale or whether disruptive changes are needed.

EPiGRAM

What is used today?





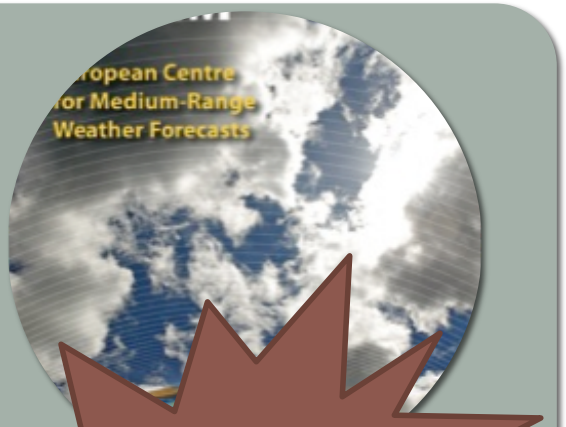


Figures from a study of the 57 leading applications used in Europe by the PRACE Project

PRACE

EPiGRAM

CRESTA experiences



CAF to overlap communication with computation

IFS

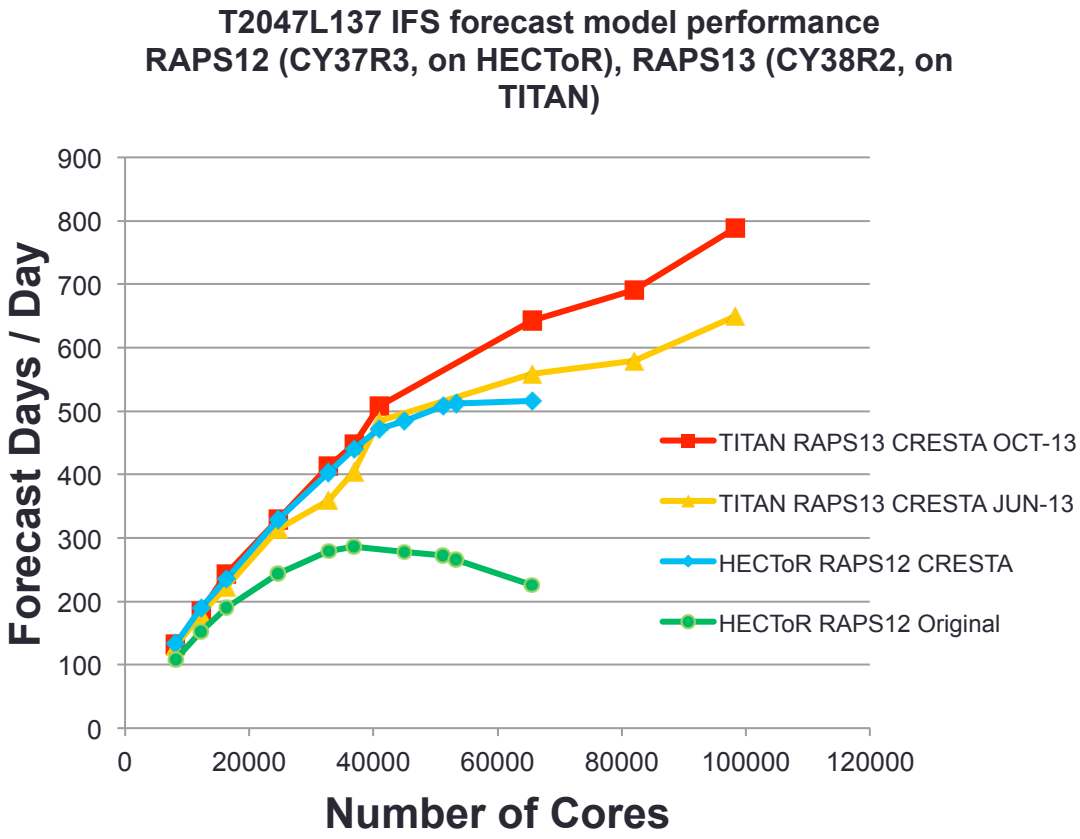
Ti eather

f e lives

Simu g e trajectory of hurric ne Sandy

Acceleration
Task-graph based parallelization
New communication models

CRESTA



EPiGRAM believes in the incremental approach and that the most promising parallel programming environments can be scaled to exascale:

Message Passing and PGAS

A Window of Opportunity

- MPI 3.0 is a major step forward but still not ready for exascale
- By extending and improving GPI to exascale we will consolidate the role of GPI and establish it as the European PGAS approach.
- EPiGRAM can complement the European CRESTA, DEEP, and Mont-Blanc exascale projects.
  - by exploring additional innovative PGAS approaches that go well beyond those considered in the current CRESTA project
  - by investigating efficient MP mechanisms that might useful for hybrid Cluster-Booster architecture in DEEP
  - by studying and analyzing one-sided communication approaches for diverse memory spaces such as the one in hybrid ARM-GPU systems in Mont-Blanc.

EPiGRAM

Key Objectives of the Project

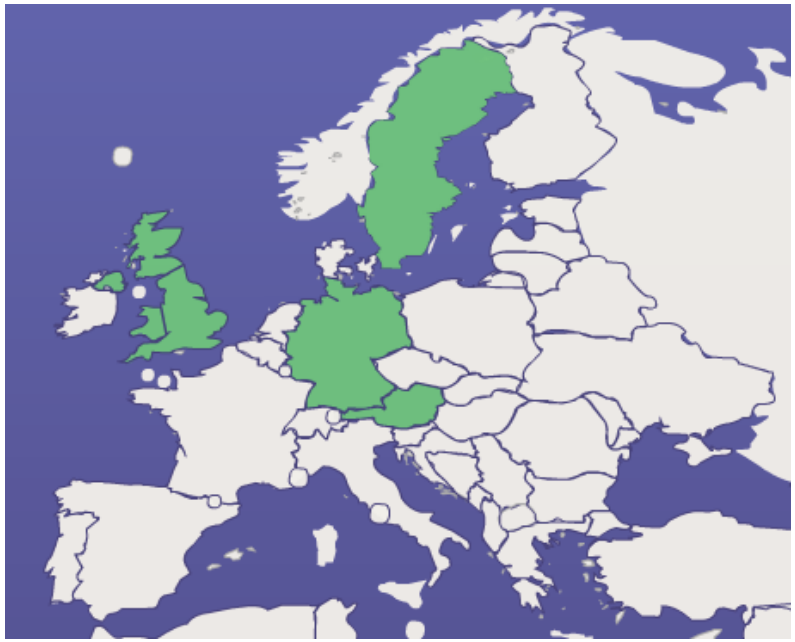
- Address the **scalability** (performance and memory consumption) problem for MP and PGAS models.
- Propose **GPI as the European PGAS** approach to exascale.
- Design an **hybrid MP-PGAS** programming model that combines the best features of the two approaches.
- Contribute to **standardization** efforts
- Prepare two **applications** to exascale by redesigning and implementing their communications kernels.

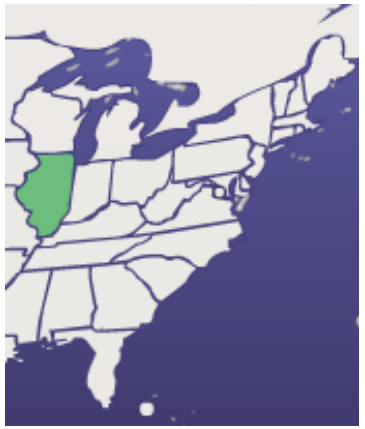
EPiGRAM

Key Players and Their Main Focus

- **KTH**: management (WP1), applications (WP6)
- **TUW**: exascale MP (WP2)
- **FRAUNHOFER**: exascale PGAS (WP3)
- **CRAY UK**: programming models for diverse memory spaces (WP3)
- **EPCC**: PGAS-based MPI (WP4)

- External Contributor: **UNIVERSITY OF ILLINOIS**: exascale MP (WP2)





EPiGRAM

Exascale Message Passing

1. Dealing with limited and slower memory:
   - in-depth analysis of MPI **derived datatype** mechanism for saving copy-operations;
   - analysis of MPI **collective interface** specification with suggestions for improvement
2. Collective communication at scale:
   - proposal for specification of homogeneous stencils, towards improved (homogeneous, regular) sparse collectives
3. Other issues to be addressed:
   - collective communication in sparse networks
   - Multi-threaded MPI
   - MPI with other models (threads, PGAS, extended message-passing models)

EPiGRAM

MPI derived datatype mechanism

MPI derived datatype mechanism (functionality for defining application-specific, structured, units of communication) enables

Zero-copy implementation:

- No explicit pack/unpack and other process local data reorganization.
- All necessary (process local) data movement implicit by communication operations.

Higher-level, descriptive advantages, can lead to genuine performance improvements

EPiGRAM

Implementing a Classic: Zero-copy all-to-all communication with MPI derived datatypes

Jesper Larsson Träff, Antoine Rougier, Sascha Hunold
traff@par.tuwien.ac.at
Vienna University of Technology (TU Wien)
Faculty of Informatics
Institute for Information Systems
Research Group Parallel Computing

Case study: Zero-copy implementation of a classic, log-round alltoall algorithm

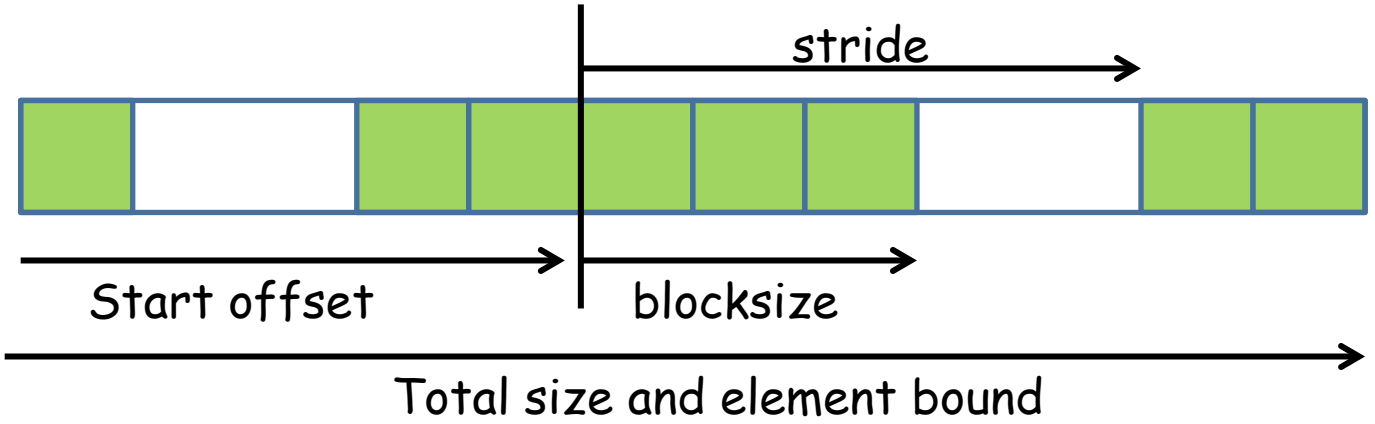
J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, D. Weathersby: Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems. IEEE Trans. Parallel Distrib. Syst. 8(11): 1143-1156 (1997)

Standard formulation:
Step 1: process-local rotate of data elements
Step 2: log rounds of communication, roughly p/2 (non-contiguous) elements grouped together and send/received per round
Step 3: process-local reverse and rotate of elements

We propose:
- Algorithmic change to get rid of Step 3
- Using per-element double-buffering and datatypes to achieve zero-copy implementation

Circular vector: derived, derived datatype

Useful derived datatype, not in MPI

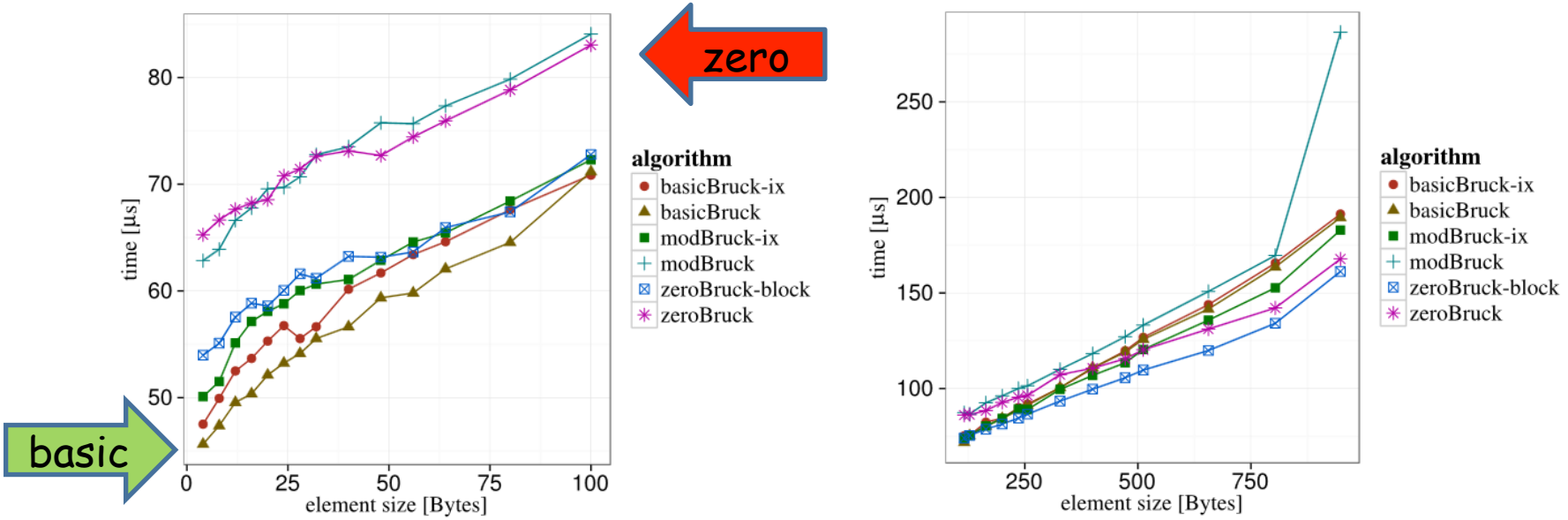


Can also be implemented using existing MPI constructors (struct of vector and contig); but quite tedious, with high overhead, and possibly more efficiently handled as native type

The Zero-copy implementation

```
for (k=1; k<p; k<<1) { // communication round
  for (j=k; j<p; j++) {
    // analyze bits of j
    ..
    b++; // number of blocks to send/receive
  }
  MPI_Type_create_struct(b,…,&sendblocktype);
  MPI_Type_create_struct(b,…,&recvblocktype);
  MPI_Type_commit(&sendblocktype);
  MPI_Type_commit(&recvblocktype);

  MPI_Sendrecv(MPI_BOTTOM,1,sendblocktype,
               MPI_BOTTOM,1,recvblocktype,…);
  MPI_Type_free(&sendblocktype);
  MPI_Type_free(&recvblocktype);
}
```

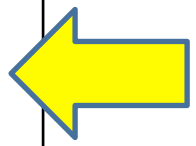
Jupiter, 36x1 MPI processes (small problems)



- BasicBruck: Bounded vector better than indexed-block
- ModBruck: Circular vector expensive

The Zero-copy implementation

```
for (k=1; k<p; k<<1) { // communication round
  for (j=k; j<p; j++) {
    // analyze bits of j
    ..
    b++; // number of blocks to send/receive
  }
  MPI_Type_create_struct(b,…,&sendblocktype);
  MPI_Type_create_struct(b,…,&recvblocktype);
  MPI_Type_commit(&sendblocktype);
  MPI_Type_commit(&recvblocktype);

  MPI_Sendrecv(MPI_BOTTOM,1,sendblocktype,
               MPI_BOTTOM,1,recvblocktype,…);
  MPI_Type_free(&sendblocktype);
  MPI_Type_free(&recvblocktype);
}
```



Per round overhead

Proposal, not in MPI: persistent collective operation

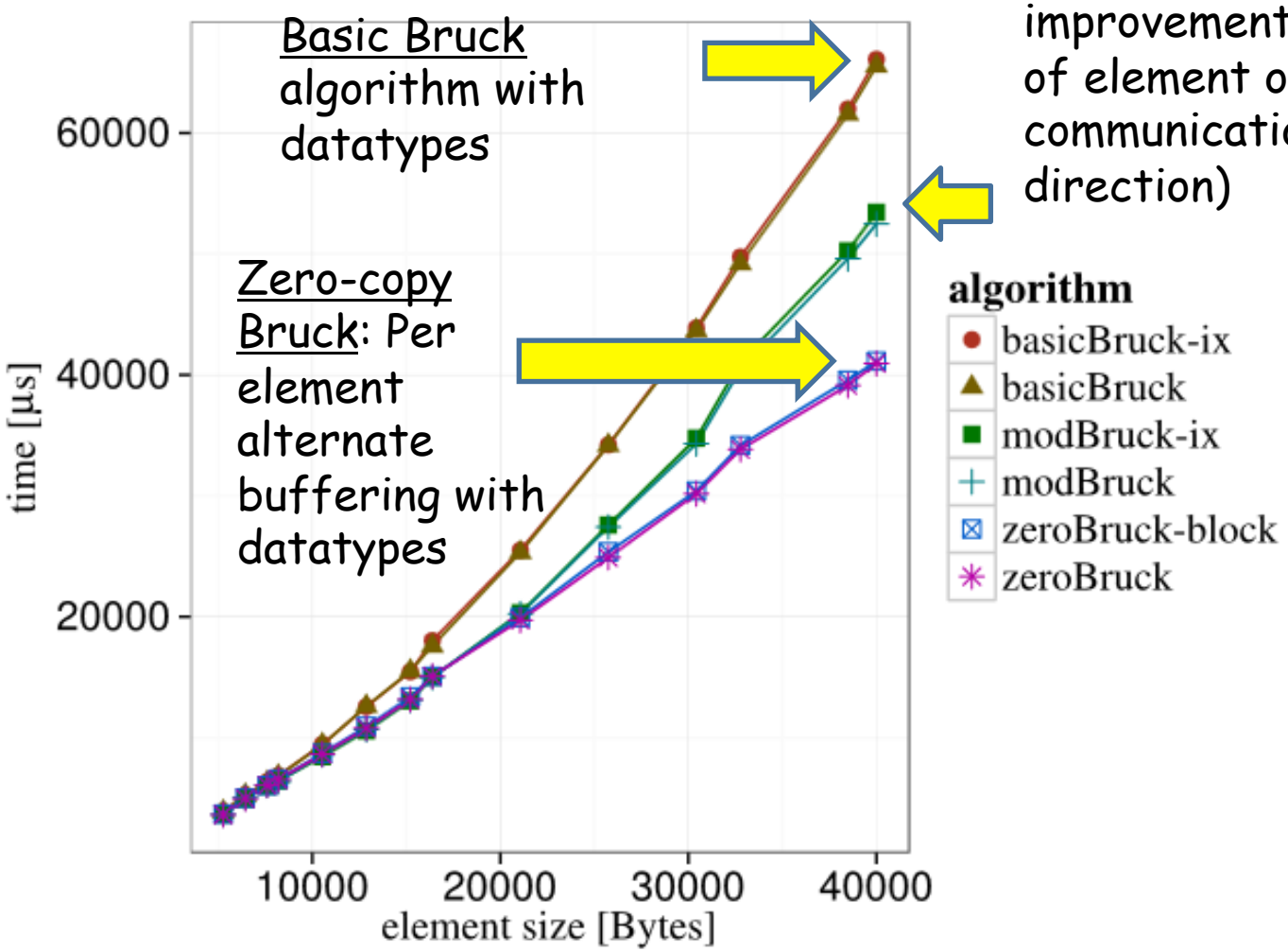
```
MPI_Alltoall_init(sendbuf,sendcount,sendtype,

recvbuf,recvcount,recvtype,comm,  &req);
```

precomputes all datatypes, creates communication schedule (log p round loop), and

```
MPI_Start(&req);
```

initiates and completes next alltoall operation (with given parameters)

Analogous to persistent point-to-point operations (in MPI)



Note: Also for small problems, where the algorithm is competitive, we have a statistically significant improvement

Other zero-copy investigations

1. Hierarchical alltoall algorithms – use new MPI 3.0 functionality, competitive performance, complex reorderings by datatypes and MPI_Alltoallw. We identify a lack of functionality in MPI_Gather/Scatter interfaces

2. Hierarchical gather/scatter/allgather algorithms: we conclusively show that zero-copy implementations are not possible. A change of interface specification is elaborated on (perhaps for discussion with MPI Forum?), a collection of new datatype constructors discussed

3. Datatype normalization: Complexity issues have never been formally studied. We show that the problem can be solved optimally in polynomial time when only vector and index-block constructors are allowed

EPiGRAM

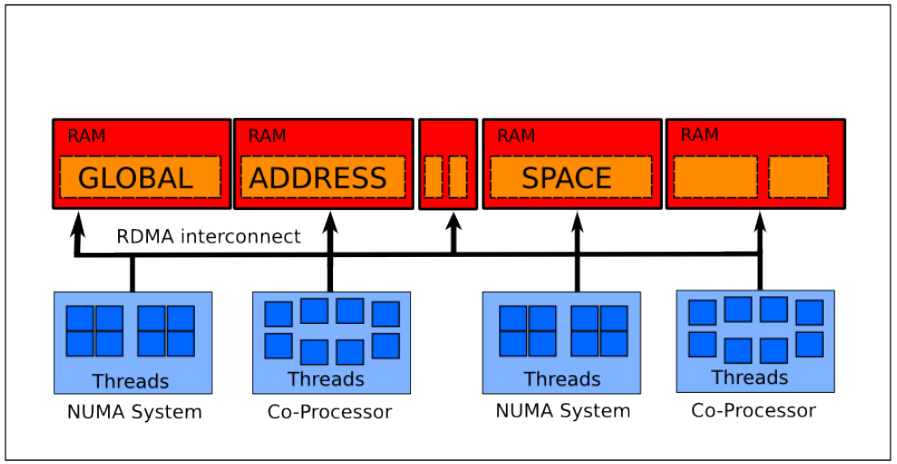
Exascale PGAS

- Objectives:
  - To investigate current limitations of traditional PGAS.
  - To propose concrete solutions to current PGAS limitations.
  - To increase scalability of collective operations and synchronization in GPI.
  - To support fault-tolerance in GPI.
  - Exploitation of diverse and hierarchical memory spaces in PGAS.

EPiGRAM

Global Address Space API: GPI
Intranode communication API

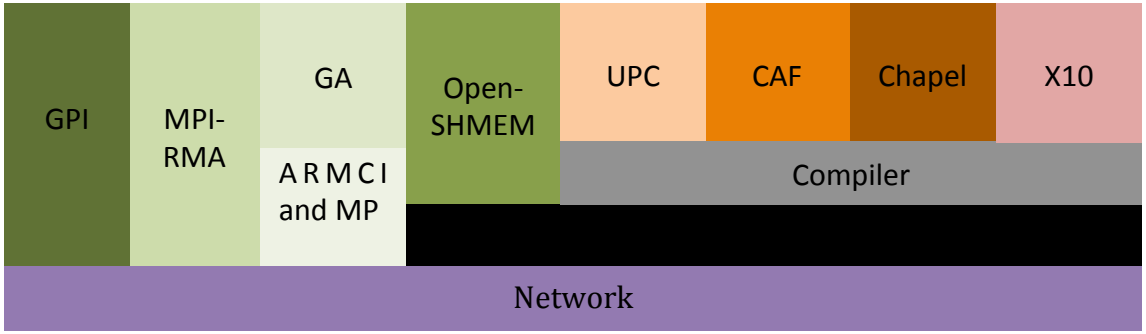
- **GPI: G**lobal Address Space A**PI**
- RDMA over IB, RoCE, Cray
- Low Latency & Wire-speed
- C/C++, Fortan
- Thread safe
    - Intel Xeon Phi, GPUs
- MPI Interoperability



- **Asynchronous communication between segments, one-sided, non-blocking**
- **Zero-copy, no CPU cycles for communication**
- **Support for fault tolerance by a timeout mechanism**
- **(Sparse) collectives with time based blocking**
- **fast remote notification**

EPiGRAM

Overview over the relevant PGAS software stack



PGAS approaches can be roughly divided into:

- Languages:
  - UPC which extends the C language, CAF which extends Fortran, Chapel and X10 as standalone languages)
  - Easy to switch to, however a lot of pressure on the compiler

- APIs (GPI, GA, ARMCI, OpenShmem):
  - Explicitly implement the communication between the nodes
  - To be used by application developers (like GPI) or by tool developers (like ARMCI)

EPiGRAM

Gap Analysis for Exascale

Heterogeneous hardware	fine
Data Types	fine
Fault Tolerance	Will become more important at Exascale, minimal requirements need to be checked
Migration Paths from MPI to PGAS	**MPI end-points** proposal will help future interoperability to map PGAS with MPI processing elements
Isolation of Libraries	Resources shared between main application and a library make isolation difficult
Locality	Topology information needs to be incorporated in PGAS approaches to quantify a distance to data

EPiGRAM

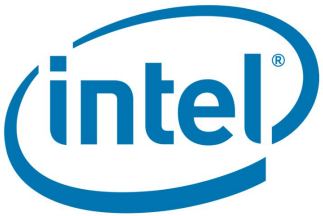
PGAS-based MPI

- Objectives:
  - Implement and evaluate efficient message passing libraries on top of RDMA operations
  - Implement and evaluate collective operations on top of RDMA operations
  - Prototype implementation of MPI endpoints proposal
  - Develop recommendations for MPI to allow efficient implementation on top of RDMA
  - Develop recommendations for RDMA hardware

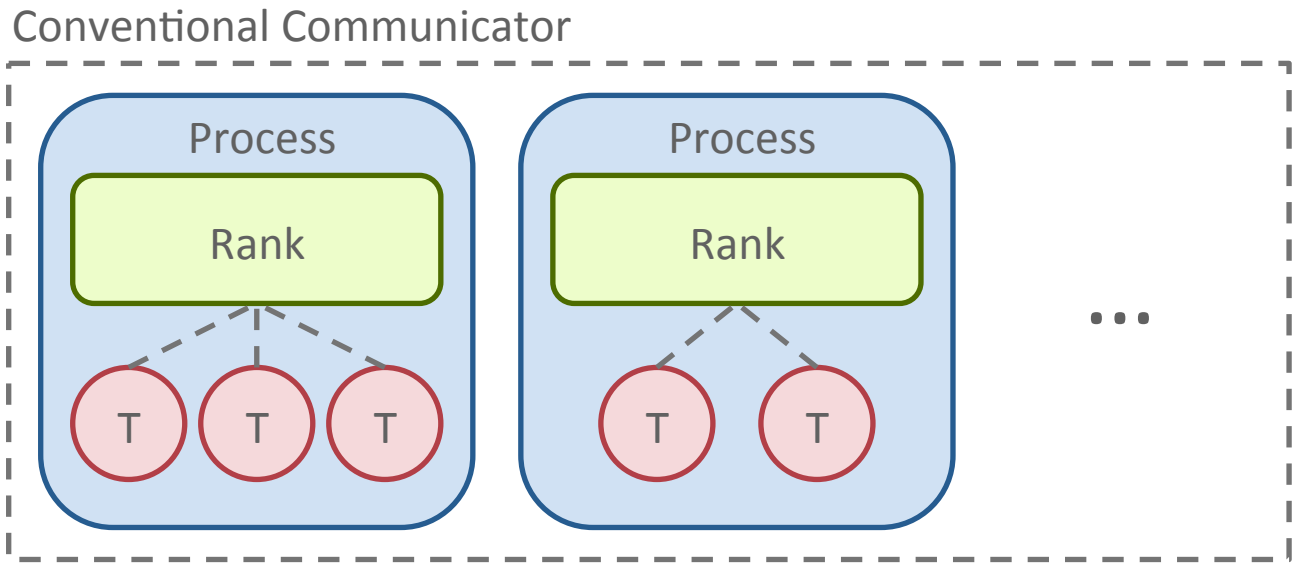
EPiGRAM

Enabling MPI Interoperability Through Flexible Communication Endpoints

James Dinan, Pavan Balaji, David Goodell,
Douglas Miller, Marc Snir, and Rajeev Thakur



Mapping of Ranks to Processes in MPI



- MPI provides a 1-to-1 mapping of ranks to processes
- This was good in the past, but usage models have evolved
  - Programmers use many-to-one mapping of threads to processes
    - E.g. Hybrid parallel programming with OpenMP/threads
  - Other programming models also use many-to-one mapping
    - Interoperability is a key objective, e.g. with Charm++, etc…

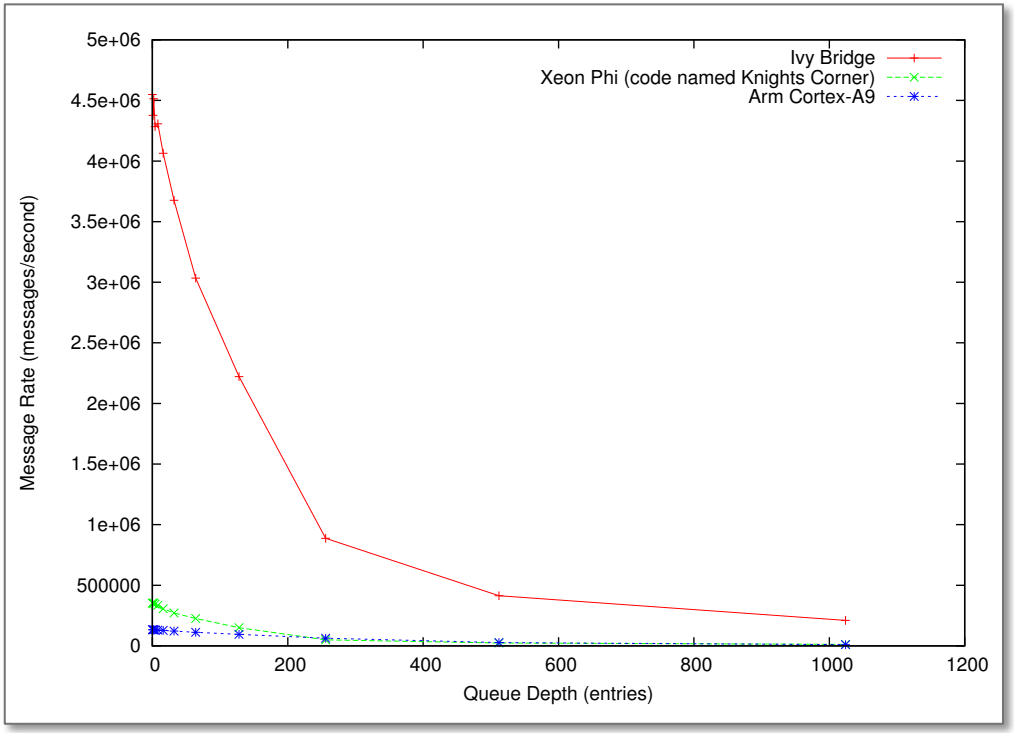
Current Approaches to Hybrid MPI+Threads

- MPI message matching space: `<communicator, sender, tag>`
- Two approaches to using THREAD_MULTIPLE

1. Match specific thread using the tag:
   - Partition the tag space to address individual threads
   - Limitations:
     - Collectives – Multiple threads at a process can't participate concurrently
     - Wildcards – Multiple threads concurrently requires care

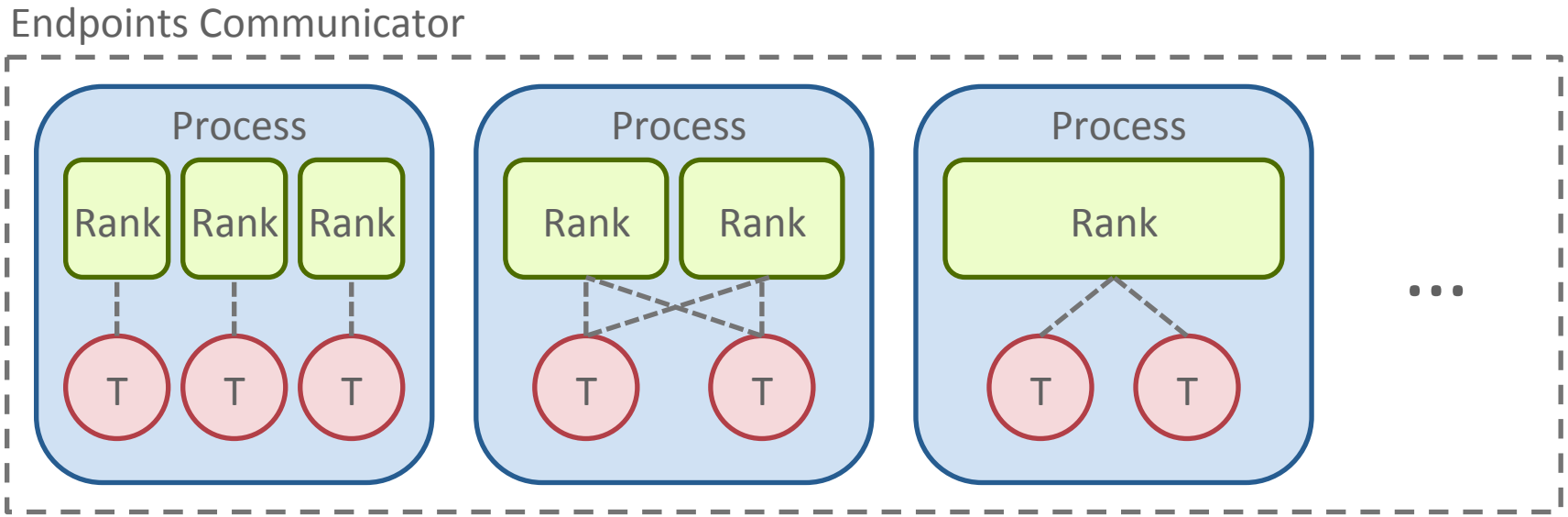
2. Match specific thread using the communicator:
   - Split threads across different communicators (e.g. Dup and assign)
   - Can use wildcards and collectives
   - However, limits connectivity of threads with each other

Impact of Light Cores and Threads on Message Rate



- Shamelessly stolen from Brian Barrett, et al. [EuroMPI '13]
- Threads sharing a rank increase posted receive queue depth (x-axis)
- Solution: More ranks!
  - Adding more MPI processes fragments the node
  - Can't do shared memory programming across the whole node

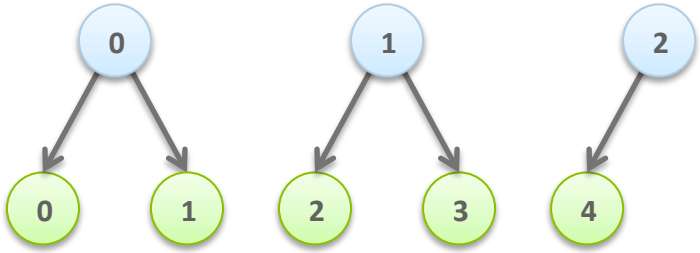
Endpoints: Flexible Mapping of Ranks to Processes



- Provide a many-to-one mapping of ranks to processes
  - Allows threads to act as first-class participants in MPI operations
  - Improve programmability of MPI + node-level and MPI + system-level models
  - Potential for improving performance of hybrid MPI + X
- A rank represents a communication “endpoint”
  - Set of resources that supports the independent execution of MPI communications
- Note: Figure demonstrates many usages, some may impact performance

Putting It All Together: Proposed Interface

```
int MPI_Comm_create_endpoints(
        MPI_Comm parent_comm,
        int my_num_ep,
        MPI_Info info,
        MPI_Comm *out_comm_hdls[])
```



- Each rank in *parent_comm* gets *my_num_ep* ranks in *out_comm*
  - *My_num_ep* can be different at each process
  - Rank order: process 0's ranks, process 1's ranks, etc.
- Output is an array of communicator handles
  - i^{th} handle corresponds to i^{th} endpoint create by parent process
  - To use that endpoint, use the corresponding handle

Usage Models are Many…

- Intranode parallel programming with MPI
  - Spawn endpoints off MPI_COMM_SELF
- Allow true thread multiple, with each thread addressable
  - Spawn endpoints off MPI_COMM_WORLD
- Obtain better performance
  - Partition threads into groups and assign a rank to each group
  - Performance benefits without partitioning shared memory programming model
- Interoperability
  - Examples: OpenMP and UPC

EPiGRAM Contributions

- Context id allocation algorithm
  - Multiple instantiations of the algorithm with identical answer
  - influence the implementations in both MPICH and OpenMPI
- Alpha prototype implementation in McMPI
- Beta prototype implementation in T3DMPI
- Participation in the Hybrid working group

EPiGRAM

Applications

- Objectives:
  - Use of the exascale MP, PGAS and PGAS-based MPI software in two real-world applications: Nek5000, iPIC3D
  - Analyze the performance of newly developed communication kernels in Nek5000 and iPIC3D
  - Provide feed-back and guidance to the development of exascale programming models

EPiGRAM

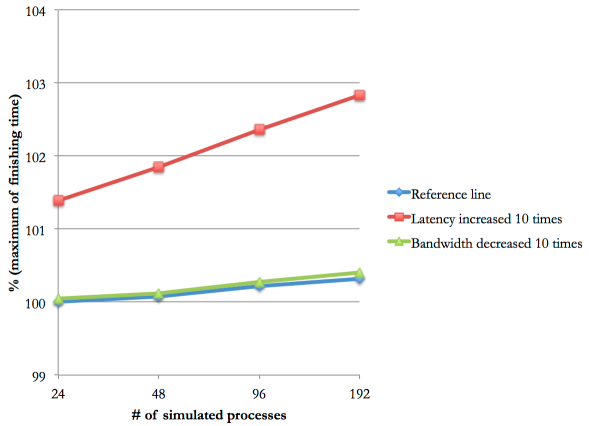
EPiGRAM Applications

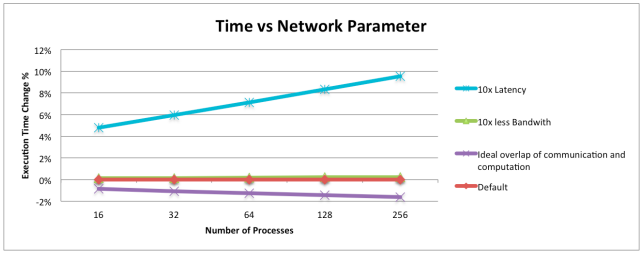
- Two real-world application with exascale needs:
  - Nek5000 (https://nek5000.mcs.anl.gov/):
    - CFD code for simulation of nuclear reactors and fluid turbulence
    - F77 (95%) C (5%) and MPI
    - Regular communication pattern involving neighbor processes
  - iPIC3D (https://github.com/CmPA/iPic3D/):
    - Particle code for simulation of interaction of solar wind with Earth's magnetosphere (space weather)
    - C++ and MPI
    - Regular communication pattern involving neighbor processes
- Initial work:
  - Identifying possible improvement in MPI communication kernels
    - Use of non-blocking CG and GMRes linear solvers. This requires redesign of solvers.
    - Use of sparse-collective to replace neighbor communication (might have improvement of message scheduling).
  - Porting applications to GPI
    - Identify interoperability issues between MPI and GPI.
    - Performance improvement with GPI One-sided.

EPiGRAM

Thinking Exascale

- We use the LogGOPSim simulator (1), based on the LOGGops model, to:
  - Simulate the EPiGRAM applications on million nodes.
  - Assess the dependency of communication kernel performance on network bandwidth and latency.
  - Assess the importance of noise on applications communication kernel
- We consider additional data-driven applications with irregular application patterns and : Graph500 (Graph construction + Breadth-first search), mpiBLAST (nucleotide or proteing alignment search)





(1) Hoefler, Torsten, Timo Schneider, and Andrew Lumsdaine. "LogGOPSim: simulating large-scale applications in the LogGOPS model." In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 597-604. ACM, 2010.

Summary

- MPI and PGAS will likely be important programming models on exascale systems
- Scalability and performance issues need to be resolved
- EPiGRAM tackles some of the important issues (collectives, data types, interoperability)

EPiGRAM

ExaMPI Workshop @ SC14

- Explore potential and issues of MPI for the exascale era
- Keynotes and invited talks from world-leading experts
- Explore hybrid systems
  - Booster, GPU, MIC, PGAS

- http://www.epigram-project.eu/exampi14/

EPiGRAM