GASPI AND THE EXA2CT PROJECT

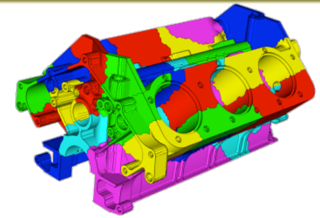
JUNE 2015, CRIHAN-CORIA

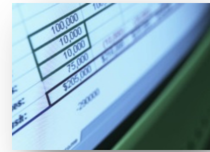
ERIC PETIT

UVSQ

Outline

Outline



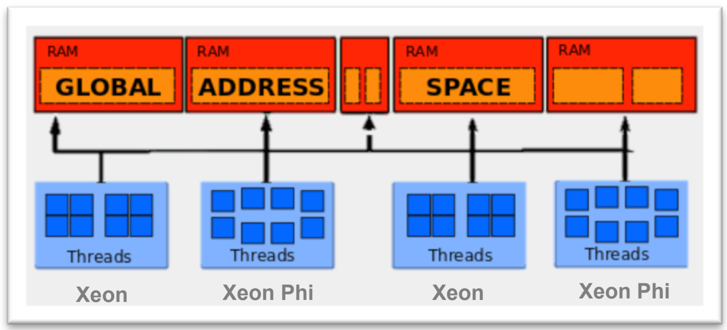


- A brief introduction to Exa2ct.

- About proto-applications

- Distributed/shared, harware/software, address space…

- An introduction to one of the main building blocks of Exa2ct - GASPI



Exa2ct

Exa2ct: EXascale Algorithms and Advanced Computational Techniques

Strategy

Proto-Applications

- Extracted from real-life HPC applications of Scientific & Industrial Board (SIB) members.

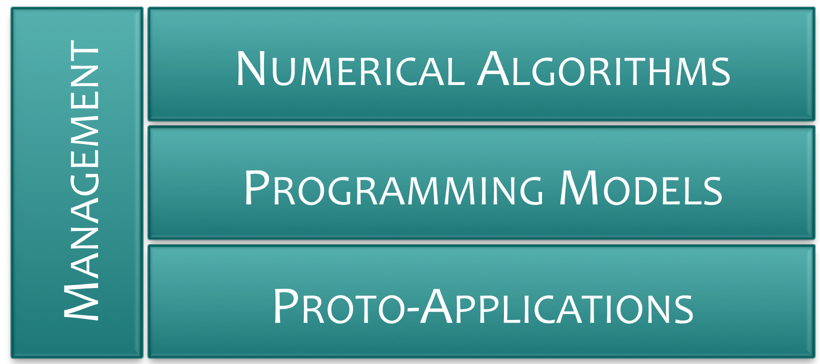
Enhanced Numerical Algorithms

Scalable, Pipelined, Robust Numerical Solvers

- That scale up to exascale performance
- That offer increased arithmetic intensity
- That survive hardware failures

Enhanced Programming Models

- Scalable
- Suitable for heterogeneous Architectures
- Resilient



The proto-application concept

Why not experimenting in the original application?

- Full applications are complex and costly to execute at scale
  - Difficulty to experiment ground breaking solutions
  - Cost of the experiments (time, PY, CPUs)
  - Need proof of concept demonstrating ROI to decide
- Codes and use-cases might not be easily shared with the community
- Need a strong and daily support of the application developer
- Portability of the solution
  - Over specialization
  - Learning curve, even in the same company/context

The Proto-App concept

- Aka mini-app, proxy-app (NERSC trinity, Argonne CESAR, the Montevo project…)
- Objectives: Reproduce at scale the behavior of a set of HPC applications and support the development of optimizations that can be translated into the original applications
  - Easier to execute, modify and re-implement
- If you cannot make the application open-source, you can at least open-source the problems.
  - Support community engagement
  - Reproducible and comparable results
  - Interface with application developers

Building a proto-application

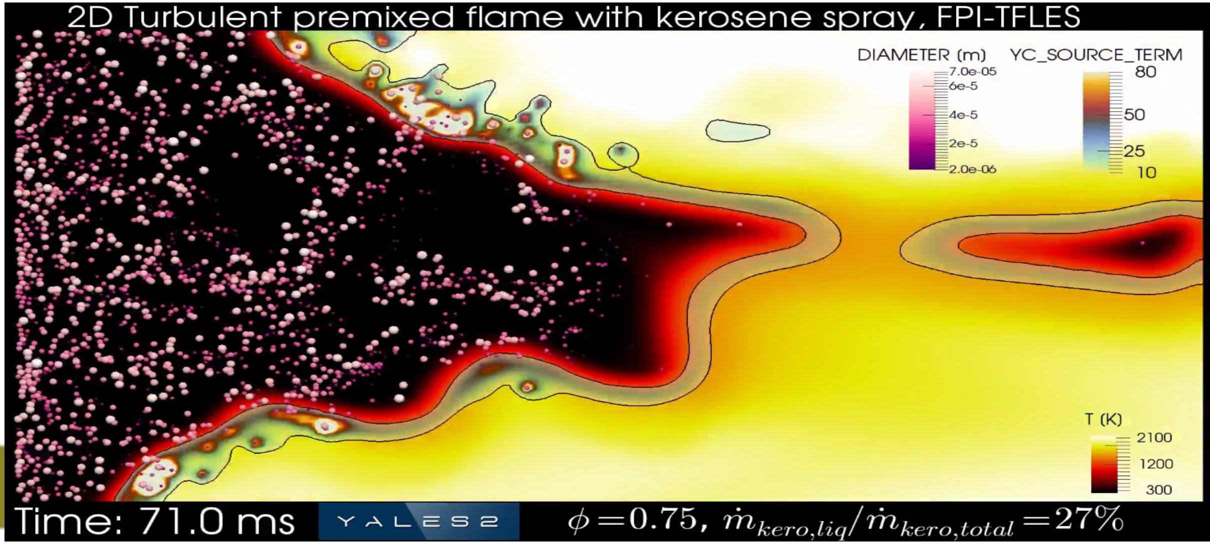
- Two alternatives with pros and cons
  - Build-up (CFD-proxy, DLB_bench, upcoming mini-FMM)
    - 'Mini-app' that mimic a full application with simpler physic
    - All aspects are explored
    - No/Less IP issue(s)
    - No specific problem targeted
    - Behavior at scale?
    - Representativeness?
    - Feedback to the real code?
    - Use cases?
  - Strip down (mini-FEM, DefCG (Yales2))
    - 'Proxy-app' which extracts and refines a particular kernel from an application
    - Target a specific issue
    - Must be representative at scale
    - Easy feedback to the user
    - Only a part of the application is addressed
    - Problem coupling?
    - Use cases generation?
    - IP (code and use case)
- IMHO I prefer the second one, building multiple proto-apps from an application to expose the different problems => however it requires the application developer and end-user experience

What are our objectives?

- Code modernization by mean of proto-application
  - Extract proto-application from real use case
  - Devellop numerical algorithm and runtimes and demonstrate on the proto-application
  - Port back the improvement in the original application
  - or devellop genuine HPC application if the modernization is not possible
- At UVSQ: Focus on task based programming and runtime
  - Dassault Aviation FEM CFD (published and released) (colab. with the ITEA2 coloc project)
    - Mini-FEM proto-application
    - DC_lib efficient and scalable library for hybrid parallelisation of unstructured FEM code
    - Ported back in the original application AND another DA application => validation of the concept
  - CORIA Yales2 Combustion (in progress) using GASPI
    - DLBbench proto-application
    - DLB_lib a library for dynamic load balancing

Use case: Yales2 Combustion

- Chimie and Lagrangian particules (for now)
- Unbalanced load!!! => Demontrate « how to » and propose a library to efficiently balanced the work on large scale distributed (and heterogeneous) systems
- Use GASPI and taskify the work to do efficient dynamic load balancing



Distributed/shared, hardware/software, address space…

Runtime and programing model taxonomy in a nutshell (1/2)

- Physically shared memory
  - Cache, ram, NUMA, local disk (not NFS)
  - Thread based: OpenMP, Cilk+, TBB, Posix
  - In between: e.g. MPC
- Physically distributed memory
  - Network (infiniband, ethernet)
  - Process based: Message passing (MPI), PVM, IPC (posix)

Runtime and programing model taxonomy in a nutshell (2/2)

- Virtually shared: NFS, PGAS
  - Patitioned Global Address Space: Processus shared a virtually common adress space:
    - OpenSHMEM, Co-array, symmetric PGAS, all processor have a version of the 'shared' space, mostly SPMD like parallelism
    - GASPI, asymmetric PGAS, any process can independantly expose a part of his memory (aka segment) to the outside world.
      - Other can address any piece of data on any remote exposed memory (rank, segment, offset)

      => allow more general graph paralleism, dataflow
    - ≠(orth.) APGAS, asynchronous PGAS (X10, Chapel) local and remote task creation

=> No message passing, but reading and writing to remote memory

Why PGAS should be more scalable?

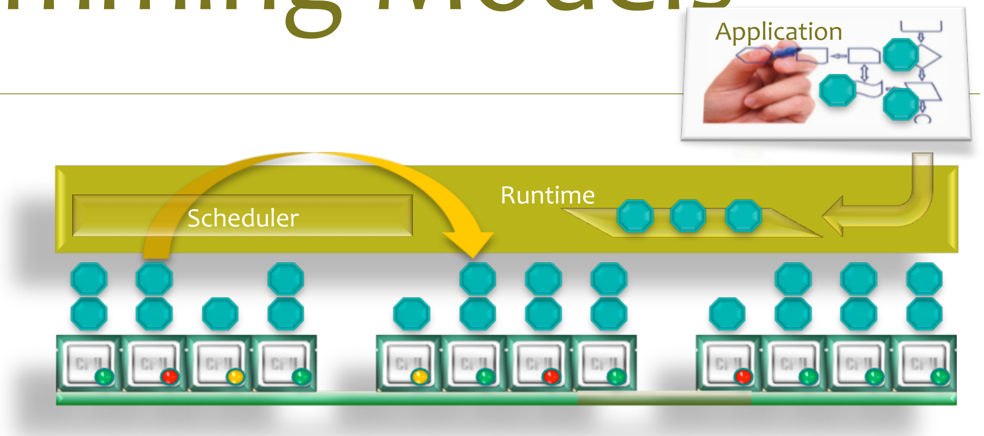
- No buffer, passive communication, asynchronous, thread-friendly
- Data flow oriented programming: producer-consummer
  - As soon as a rank produces some data he can write it to the consumer
  - As soon as a consumer is ready, he can consumes this data

  ⇒ asynchronous, fine grain

  ⇒ Sync on data dependancy, no over-synchronization (e.g. bulk sync. phases)

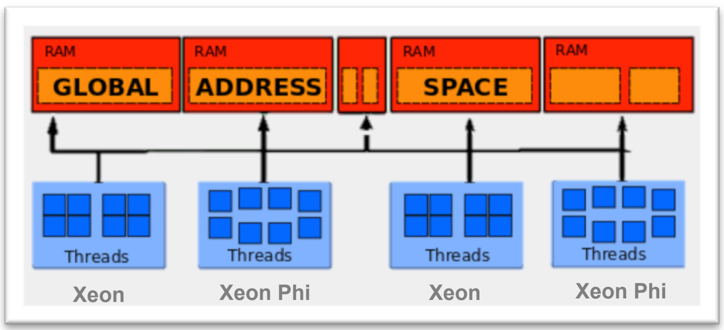
  ⇒ Very natural programming
- Task based programming:
  - more flexibility to communicate
  - authorize finner grain work decomposition => more concurency (comp/comp, comm/comp)

Exa2ct Programming Models



Tasks

Formulate your program in terms of logical tasks, instead of threads.



GASPI – a *PGAS API*

- ***not*** a new language or a language extension, but complements existing languages *(library approach ~ MPI)*
- Support for resilience *e.g. time-out mechanisms for all non-local procedures*

GASPI + Tasks ➔ extreme scalability

Opportunities : Heterogeneous execution platforms for tasks, task/data migration, task/data resilience, ...

An introduction to one of the main building blocks of Exa2ct - GASPI

GASPI History

- **GPI/GPI2**
  - Originally called Fraunhofer Virtual Machine (**FVM**)
  - Developed since 2005
  - Used in industry projects at CC-HPC of Fraunhofer ITWM
- **GPI2 – implements GASPI (GPLv3)**

GPI: Winner of the „Joseph von Fraunhofer Preis 2013“

www.gpi-site.com

GASPI

Key Objectives of GASPI

- **Scalability**
  - From bulk–synchronous two sided communication patterns to asynchronous one-sided communication
  - Remote completion via notifications and bundled communication.
- **Flexibility and Versatility**
  - Multiple configurable segments
  - Support for multiple memory models
  - Configurable hardware ressources
- **Failure Tolerance**
  - Timeouts in non-local operations
  - Dynamic node sets.

GASPI

Scalability

Performance

- One-sided read and writes
- **Remote completion in PGAS** with notifications.
- Asynchronous execution model
  - **RDMA queues** for one-sided read and write operations, including support for arbitrarily distributed data.
- Threadsafety
  - Multithreaded communication is the default rather than the exception.
- Write, Notify, Write_Notify
  - **relaxed synchronization**
  - traditional (asynchronous) handshake mechanisms remain possible.
- No Buffered Communication - Zero Copy.

GASPI

Scalability

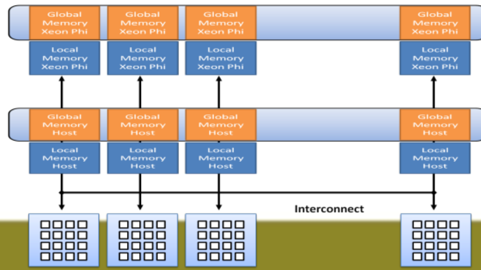
Performance

- **No polling** for outstanding receives/acknowledges for send
  - **no communication overhead,** true asynchronous RDMA read/write.
- Fast synchronous collectives with time-based blocking and timeouts
  - Support for asynchronous collectives in core API.
- Global Atomics for all data in segments
  - FetchAdd, cmpSwap.
- Extensive profiling support. (inc. Scalasca/scoreP)

GASPI

Versatility

- **Segments**
  - Support for **heterogeneous Memory Architectures** (NVRAM, GPGPU, Xeon Phi, Flash devices).
  - Tight coupling of Multi-Physics Solvers
  - Runtime evaluation of applications (e.g Ensembles)
- **Multiple memory models**
  - Symmetric Data Parallel (OpenShmem)
  - Assymetric
  - Symmetric Stack Based Memory Management
  - Master/Slave



Flexibility

Interoperability and Compatibility

- Compatibility with most Programming Languages. (library approach)
- Interoperability with MPI.

⇒ Allow iterative porting, and reuse the original code as most as possible
⇒ Possibility to port back to MPI (Why ? Real life constraints…)

- Compatibility with the Memory Model of OpenShmem.
- Support for all Threading Models (OpenMP/Pthreads/..)
  - GASPI is orthogonal to Threads.

- GASPI is a nice match for **tile architecture** with **DMA** engines (e.g. kalray, tilera…) Not implemented yet…

GASPI

EXA2CT

Flexibility

Flexibility

- Allows for **shrinking and growing** node set.
- User defined collective with **time based blocking**.
- Offset lists for RDMA read/write (write_list, write_list_notify)
- **Groups** (Communicators)
- Advanced Ressource Handling, configurable setup at startup.
  - Explicit connection management.
  - Explicit segment registration

GASPI

Failure Tolerance

GASPI

Failure Tolerance.

- Timeouts in all non-local operations
- Timeouts for Read, Write, Wait, Segment Creation, Passive Communication.
- Dynamic growth and shrinking of node set.
- Fast Checkpoint/Restarts to NVRAM.
- State vectors for GASPI processes.

The GASPI API

- 52 communication functions
- 24 getter/setter functions
- 108 pages

… but in reality:

- Init/Term
- Segments
- Read/Write
- Passive Communication
- Global Atomic Operations
- Groups and collectives

```
GASPI_WRITE_NOTIFY ( segment_id_local
                   , offset_local
                   , rank
                   , segment_id_remote
                   , offset_remote
                   , size
                   , notification_id
                   , notification_value
                   , queue
                   , timeout )
```

Parameter:

(in) segment_id_local: the local segment ID to read from

(in) offset_local: the local offset in bytes to read from

(in) rank: the remote rank to write to

(in) segment_id_remote: the remote segment to write to

(in) offset_remote: the remote offset to write to

(in) size: the size of the data to write

(in) notification_id: the remote notification ID

(in) notification_value: the value of the notification to write

(in) queue: the queue to use

(in) timeout: the timeout

GASPI

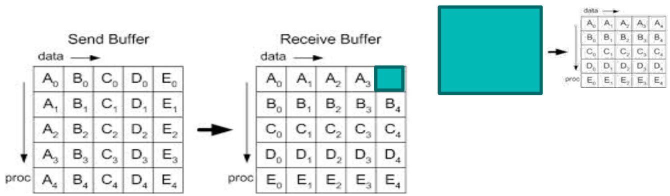
www.gaspi.de

Matrix Transpose

GASPI Matrix Transpose pseudo-code
(From the tutorial, simple but very efficient)

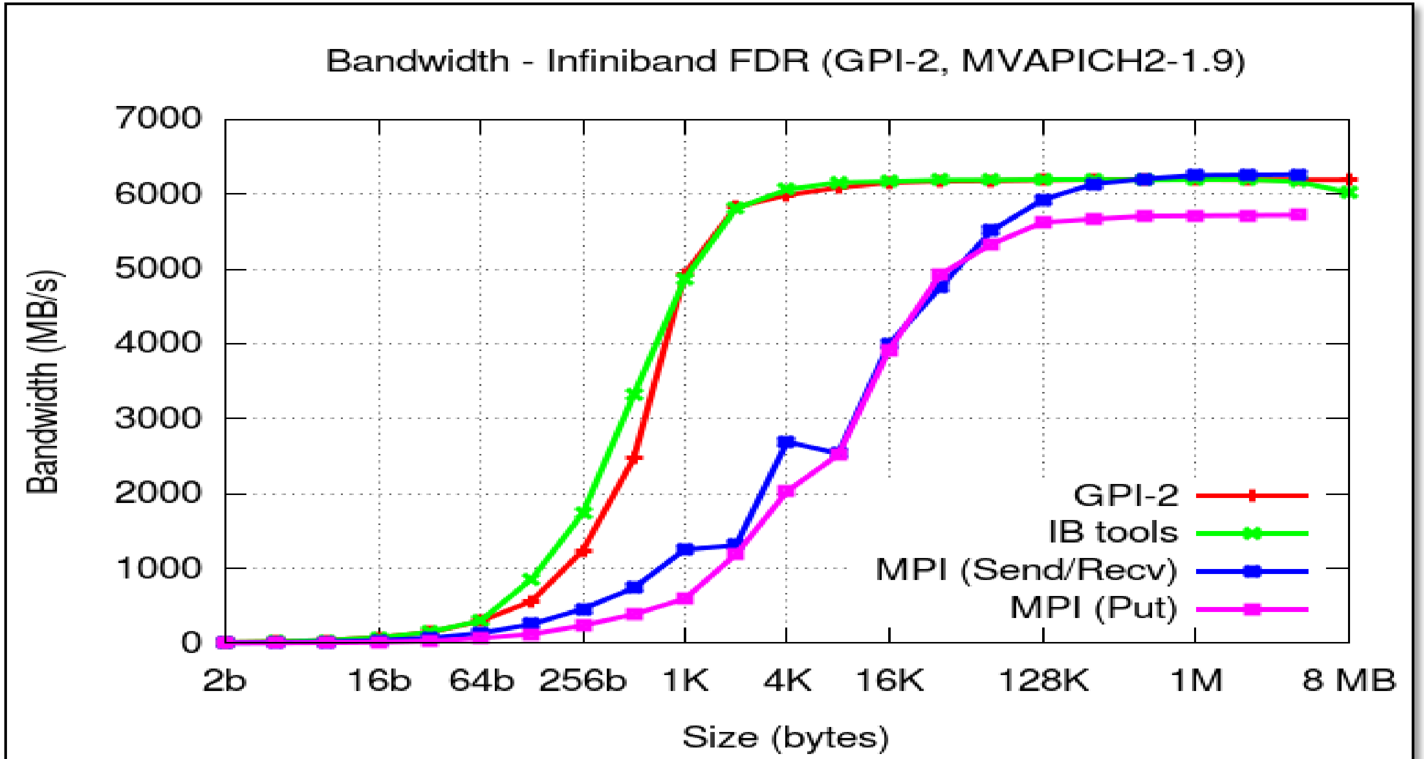
```
#pragma omp parallel
{
  #pragma omp master
  for all neighbours
      write_notify(tile)

  while (notcomplete)
  {
      wait_for_notify(@tileId)
      atomic_reset_notification(@tileId)
      If (tileId)
                notcomplete=notcomplete-1
                do_local_transpose(tileId)
  }
}
```



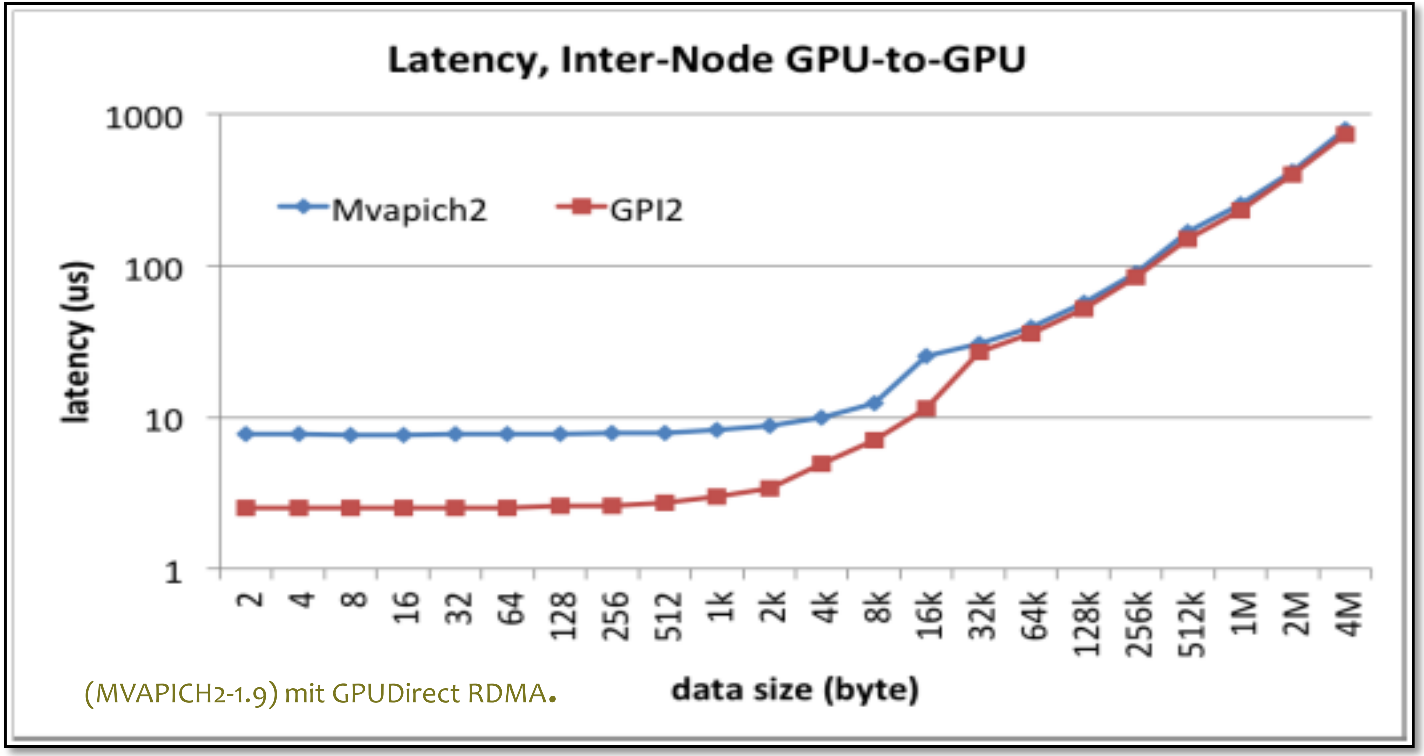
Implementation (GPL v3)

GASPI



Implementation (GPL v3)

GASPI



(MVAPICH2-1.9) mit GPUDirect RDMA.

http://www.gpi-site.com

Conclusion

- Sure Exascale will require some weak-scaling…
- but it will be also be strong scaling (manycores)
- Flat MPI and bulk synchronous models will not work at scale
- OpenMP? Pragma? Communication and scheduling explicit management? Vectorization? Accelerator DSL?

  ⇒ Can it remain the end-user concern? => oblivious programing concept

Todays concerns:

Code/algorithm modernization

Taking the right direction

Secure the investment

⇒ Propose a new alternative to address parallel programming, PGAS is one step in that direction.

⇒ Not a big jump! (like Chapel for example)

GASPI

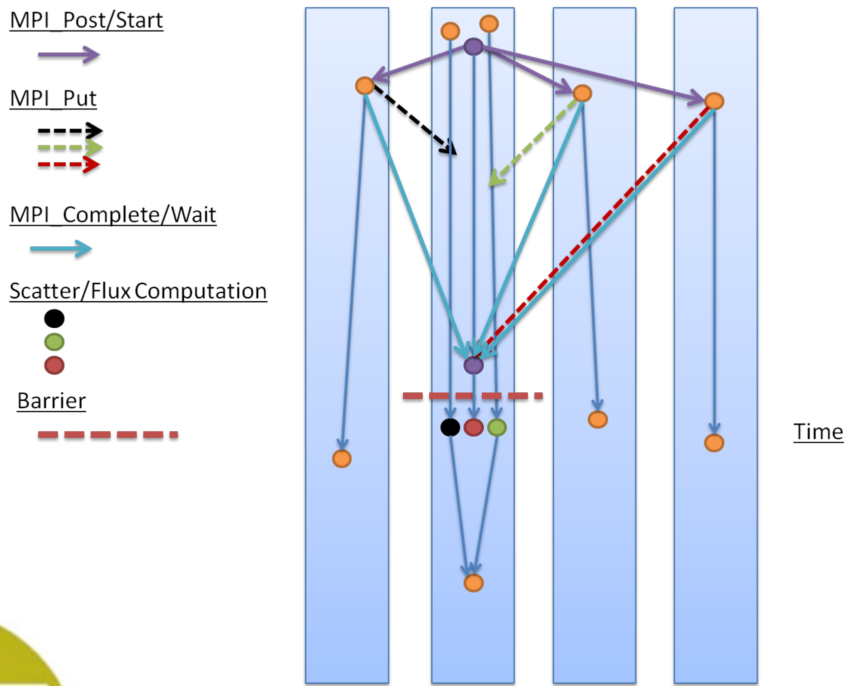
EXA2CT

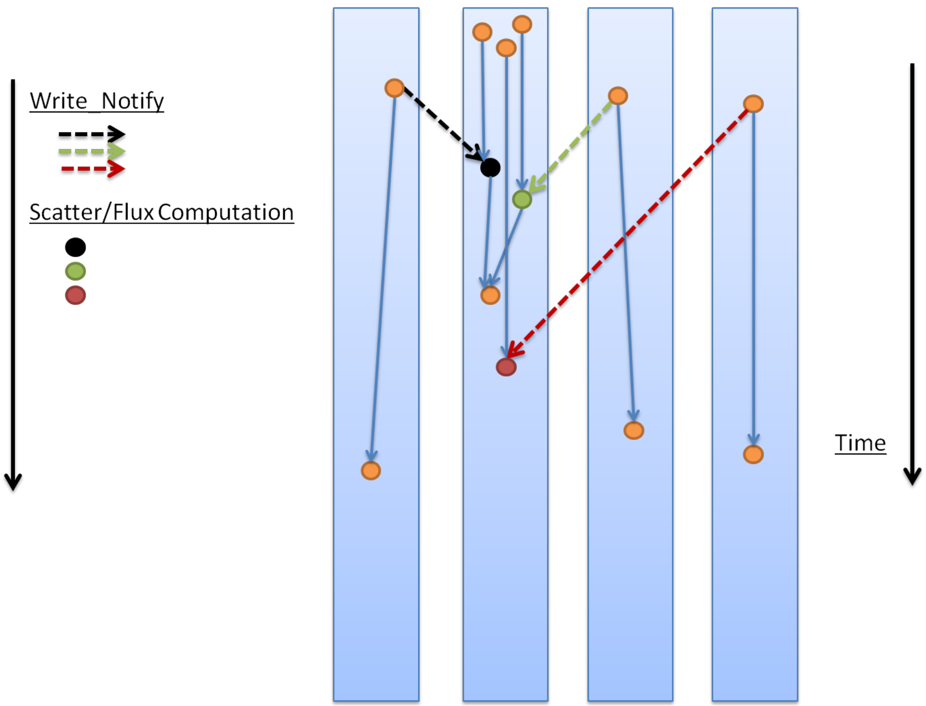
Thanks, questions?

- Exa2ct member will be at ISC (Booth, HPCSET and
- Next GASPI Forum Meeting: Parallel to ISC in Frankfurt.
- Next GASPI Tutorials 27.4. HLRS, 21.5 Bristol (Archer)
- GASPI Forum mailing list: gaspi-forum@kth.se

Backup slides

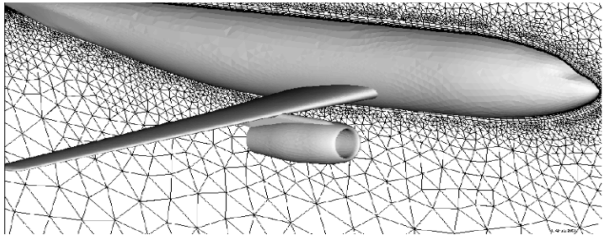
Task (Graph) Models + X





Halo Exchange

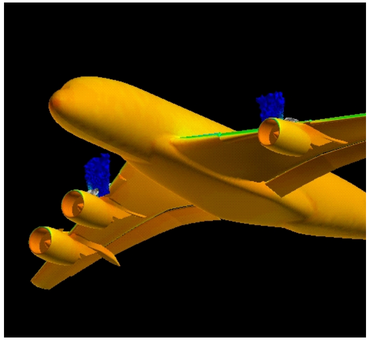
CFD Proxy



Exa2ct Proto Application – Ghost Cell Exchange at Exascale

DLR

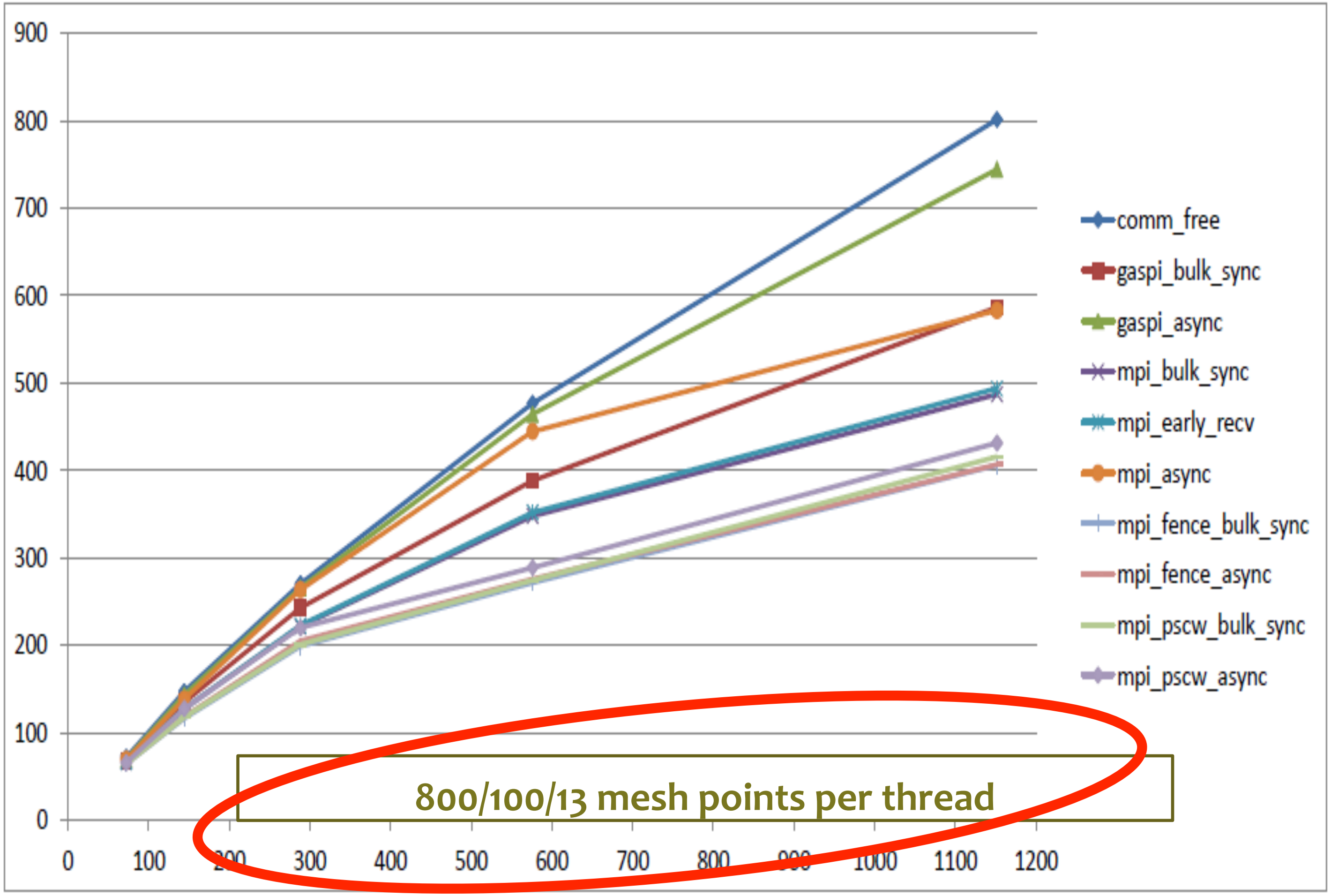
- Multithreaded OpenMP/MPI/GASPI reconstruction of gradients of a pre-partitioned and pre-coloured aircraft (DLR F6) 2 million point mesh.
- Subsequent halo (ghost cell) exchange for the gradients.
- Reordering of mesh faces, halo faces first.
- Trigger send / put / write of the halo parts as early as possible.
- First thread which completes the halo for one of the commpartners issues send / put / write.
- Linear weak scaling.
- Strong scaling scenario ~ 50 mesh points per core.



EXA2CT

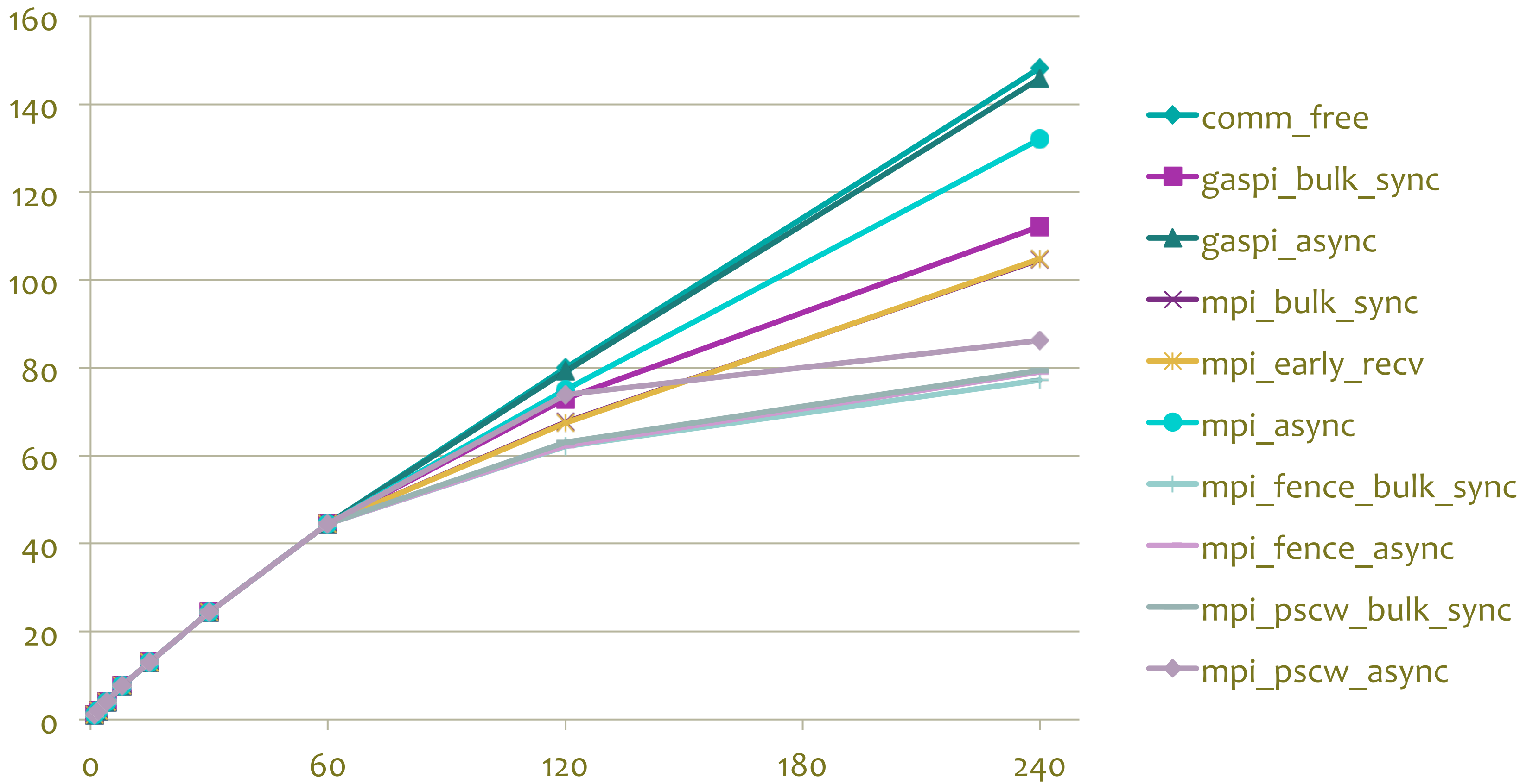
CFD Proxy on Xeon Ivy Bridge

Halo Exchange



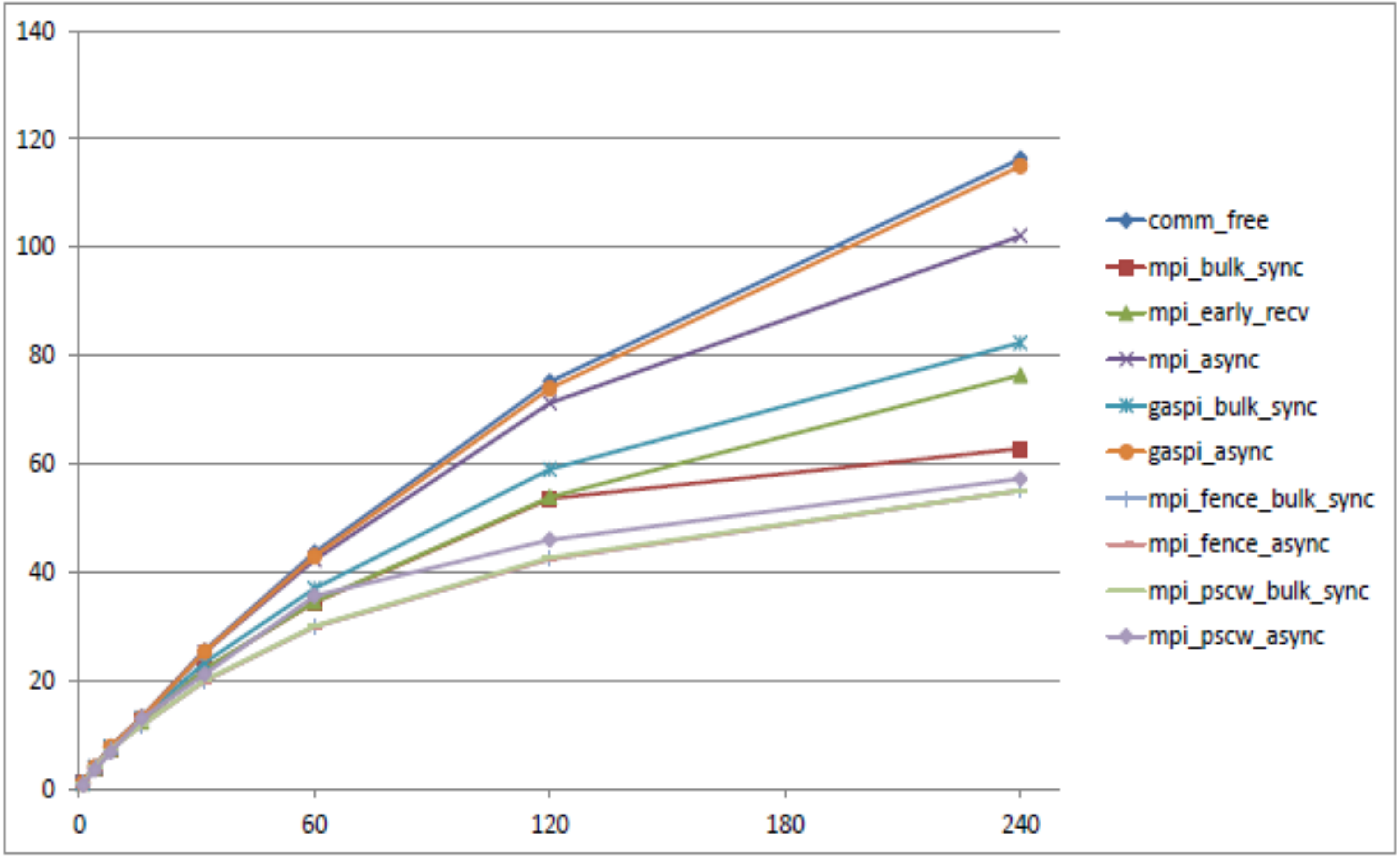
- MPI Notification Emulation on Xeon Phi

Halo Exchange

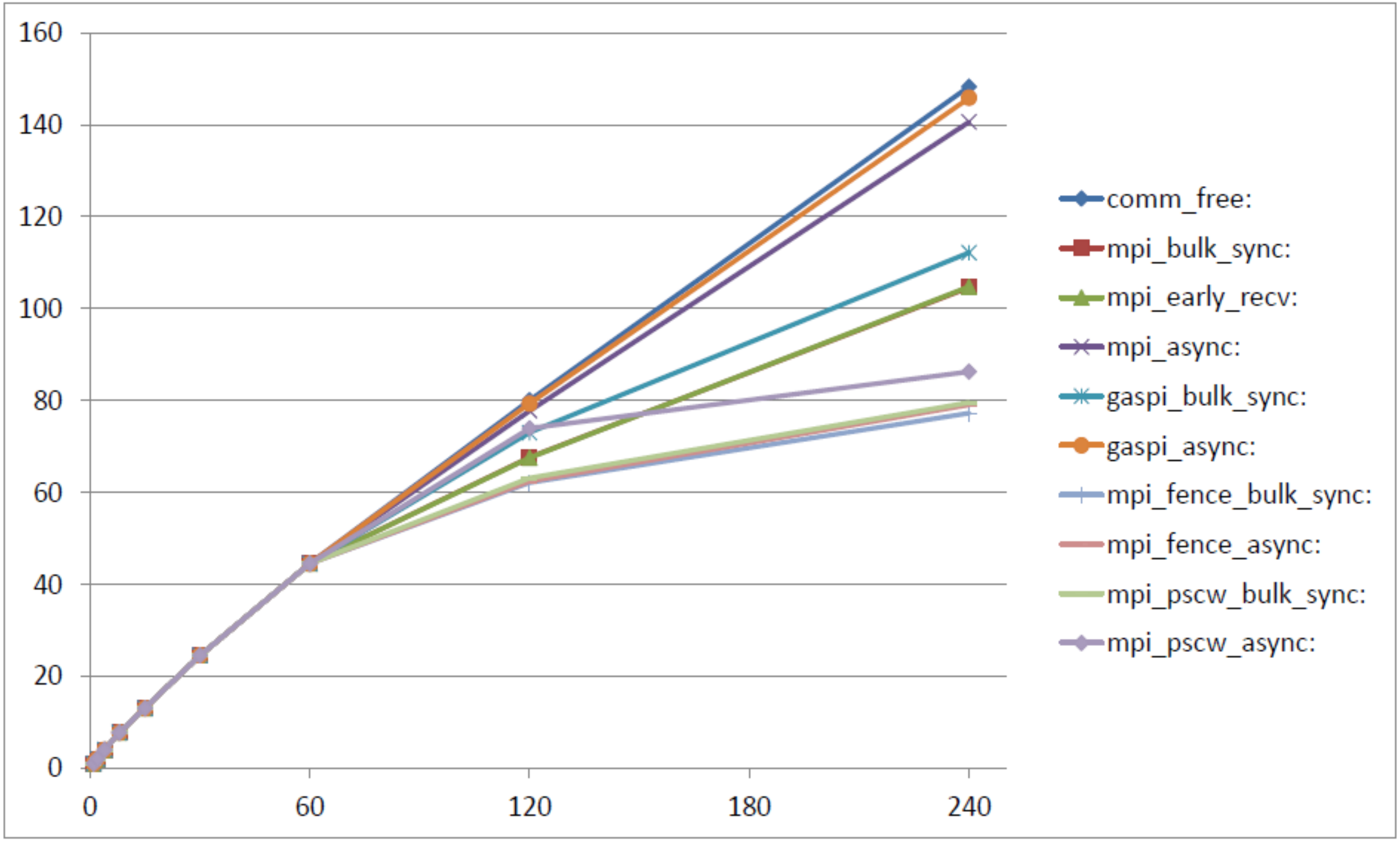


https://github.com/PGAS-community-benchmarks

CFD Proxy Xeon Phi
3V Multigrid Lvl 1



CFD Proxy Xeon Phi
3V Multigrid Lvl 2



CFD Proxy Xeon Phi
3V Multigrid Lvl 3