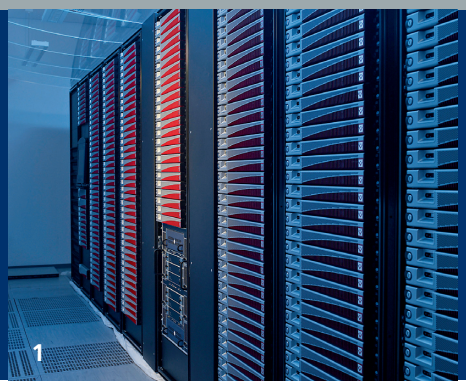
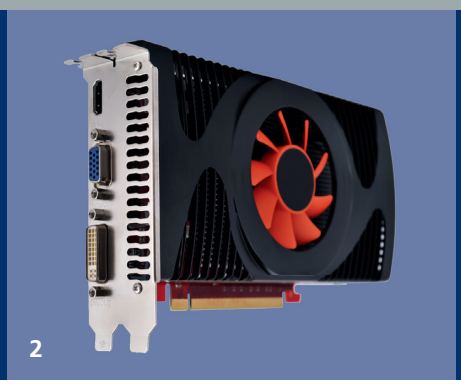
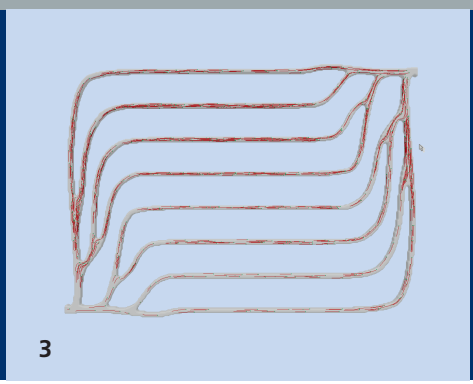
Fraunhofer
ITWM

FRAUNHOFER INSTITUTE FOR INDUSTRIAL MATHEMATICS







1 *x86-Cluster*

2 *GPU*

3 *Heat Exchanger, Lattice-Boltzmann Code*

GRAPA – PARALLELIZATION MADE EASY

Want to write portable and efficient simulation codes but don't know how to face the hardware challenges?

The Problem: Current and Future Hardware Platforms

- Many-Core-Chips: Require good thread programming and an awareness of concurrency and synchronization issues as well as non-uniform memory access.
- Heterogeneous Hardware: With a conventional CPU and additional accelerators (e. g. GPUs) in one single node, problems with code and performance portability arise.
- Clusters: Good domain decomposition and load balancing as well as effective strategies to cope with concurrency and process synchronization become inevitable.

The Solution: The GraPA Middleware

- GraPA aims at solving all the mentioned issues for you.
- It provides an intuitive and easy-to-use interface for scientists developing simulation codes.
- It is designed to fit the needs of typical engineering and natural sciences simulation projects.
- Supported simulation methods include: Molecular Dynamics, Lattice Boltzmann, Finite Differences.

Find out more about GraPA and how it can help you!

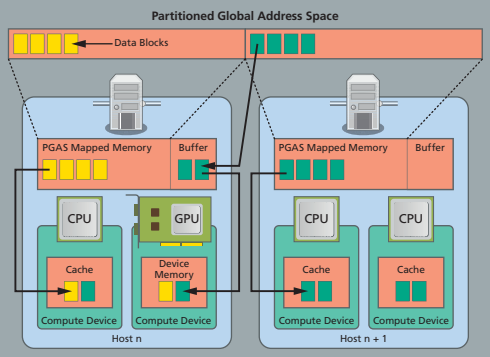
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM

Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

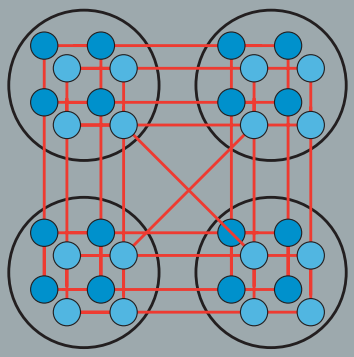
Contact
Dr. Franz-Josef Pfreundt
Phone +49 631 31600-4459
pfreundt@itwm.fraunhofer.de

www.itwm.fraunhofer.de

© Fraunhofer ITWM 2010



1



2

```
Framework<OpenCL::OpenCLDeviceManager,eBDETypes,
  eBDEPolicies,LinkPolicies,LinkTypes>
  F(PGASMode, Graph);

F.registerNodeOperator<Inlet,0>();
F.registerNodeOperator<Pipe,1>();

F.registerLinkOperator<Inlet,Inlet,Inlet_Inlet,0>();
F.registerLinkOperator<Pipe,Pipe,Pipe_Pipe,1>();
F.registerLinkOperator<Inlet,Pipe,Inlet_Pipe,0,1>();

F.registerNodeOutputOperator<OutputOp,0>();
F.registerNodeOutputOperator<OutputOp,1>();

for (int step = 0; step < numSteps; ++step)
{
  F.callLinkAndNode();
}
```

3

1 *GraPA Memory Model (CPU/GPU)*

2 *Graph Partitioning*

3 *GraPA Code-Snippet*

GraPA Details

Domain Decomposition and Load Balancing:

- User input is mapped to a weighted graph.
- This graph is distributed across available hardware.
- Dynamic re-balancing during simulation if required.
- Decomposition, balancing and redistribution completely hidden from the user.

Network Communication and Memory Hierarchies:

- Fast data communication using a partitioned global address space (GPI).
- GraPA manages loading of data blocks across the network.
- GraPA also optimizes usage of caches and device memories.

Concurrency and Synchronization:

- One-sided network communication via GPI allows for asynchronous thread execution.
- Concurrency is detected from graph structure.
- Synchronization costs are minimized.
- Details of thread programming are hidden from the user.

Heterogeneity and Portability:

- GraPA leverages heterogeneous hardware (e. g. traditional CPUs, GPUs, Cell).
- OpenCL provides portability of compute kernels between different hardware.
- Code-generation and auto-tuning techniques for performance portability and simpler implementation of compute kernels (work in progress).