GPU-based image processing with the UFO framework

Matthias Vogelgesang

matthias.vogelgesang@kit.edu

Institute for Data Processing and Electronics

INTRODUCTION

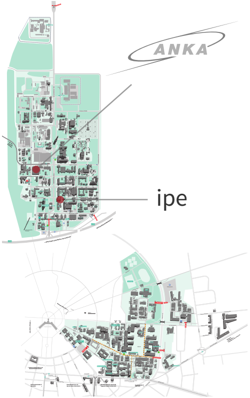
Institute for Data Processing and Electronics

Hardware

- Development (FPGA, ASIC)
- Manufacturing (circuit production, bonding)
- Characterization and long-term tests

Software

- Experiment control and data acquisition
- Analysis of acquired data
- Large scale data storage



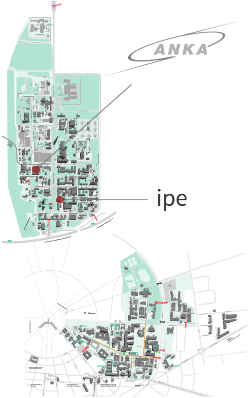
Institute for Data Processing and Electronics

Hardware

- Development (FPGA, ASIC)
- Manufacturing (circuit production, bonding)
- Characterization and long-term tests

Software

- Experiment control and data acquisition
- Analysis of acquired data
- Large scale data storage



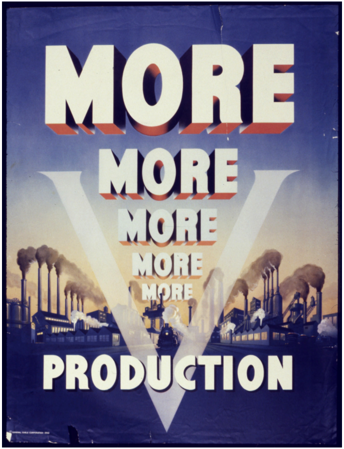
Data analysis for synchrotron μCT

Higher requirements

- Compute-intensive reconstruction
- More pre- and post-processing
- Faster and direct feedback

More data

- Better sensors
- Higher throughput
- Time-resolved scans



Data analysis for synchrotron μCT

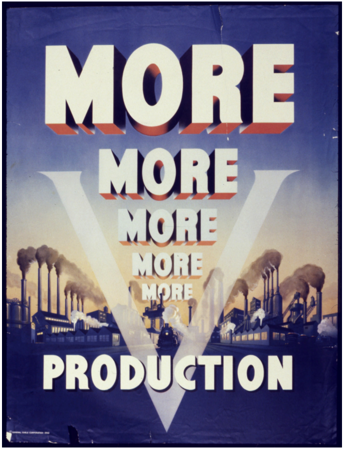
Higher requirements

- Compute-intensive reconstruction
- More pre- and post-processing
- Faster and direct feedback

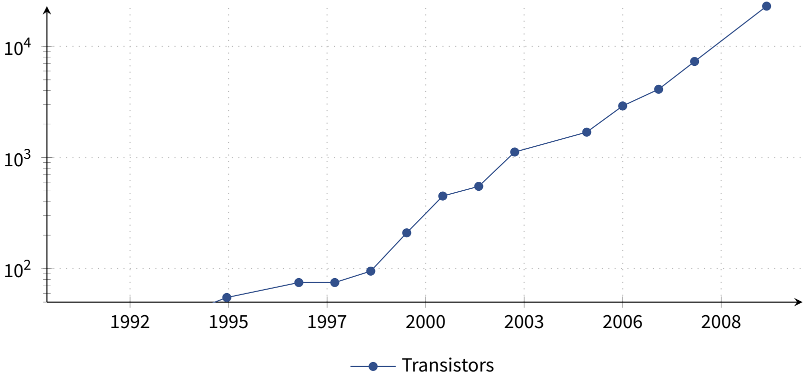
More data

- Better sensors
- Higher throughput
- Time-resolved scans

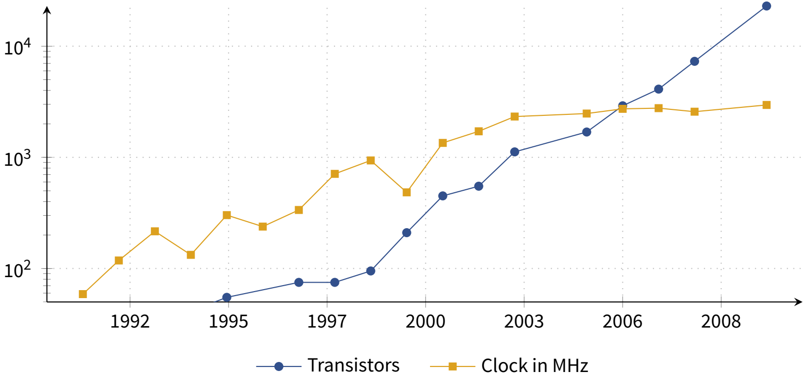
Existing tools can hardly satisfy the demands!



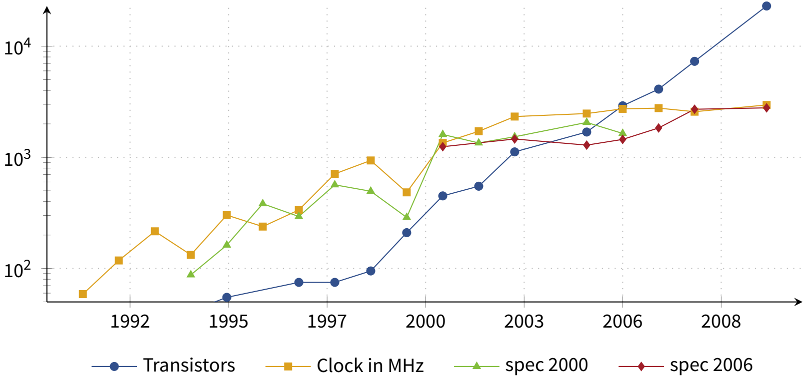
Single core development



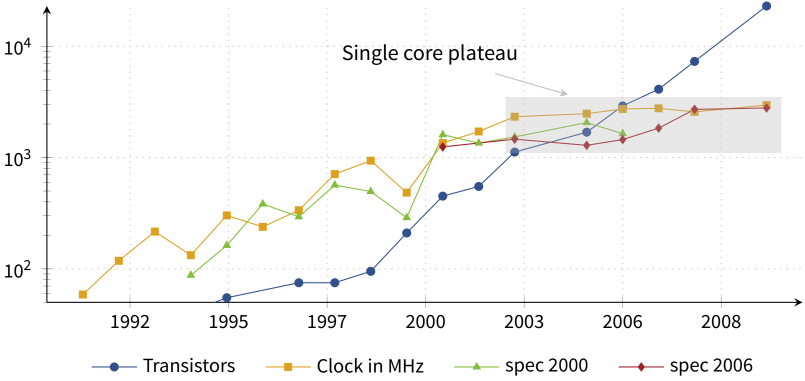
Single core development



Single core development



Single core development



Exploiting Moore's law today

Using larger integration

- Complex instruction sets
- Larger caches
- More cores

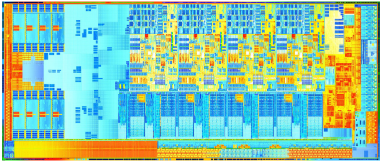


Figure : Intel Haswell cpu-Die

Exploiting Moore's law today

Using larger integration

- Complex instruction sets
- Larger caches
- More cores

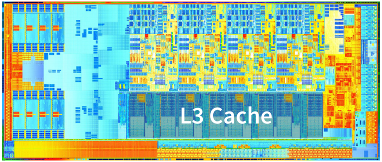


Figure : Intel Haswell cpu-Die

Exploiting Moore's law today

Using larger integration

- Complex instruction sets
- Larger caches
- More cores

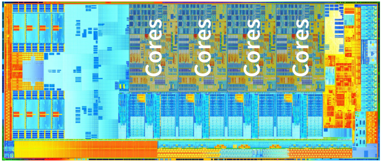


Figure : Intel Haswell cpu-Die

Exploiting Moore’s law today

Using larger integration

- Complex instruction sets
- Larger caches
- More cores

Parallelization required

- Instruction level (sse, avx)
- Multi-core CPUs und many-core GPUs
- Multi-node cluster

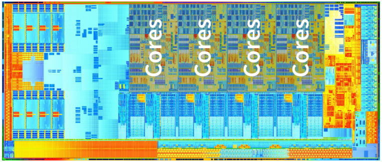


Figure : Intel Haswell cpu-Die

Exploiting Moore's law today

Karlsruhe Institute of Technology

Using larger integration

- Complex instruction sets
- Larger caches
- More cores

Parallelization required

- Instruction level (sse, avx)
- Multi-core CPUs und many-core GPUs
- Multi-node cluster

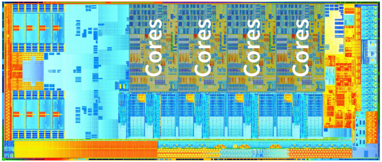


Figure : Intel Haswell cpu-Die

That means software *must* be adapted to gain any performance ☹

HETEROGENEOUS STREAM PROCESSING

Overview of the compute system

Karlsruhe Institute of Technology

Requirements

- Process image streams on the fly
- Use heterogeneous compute systems

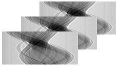
How?

- Define tasks of work
- Connect processing workflows
- Let a run-time schedule tasks

Overview of the compute system

Requirements

- Process image streams on the fly
- Use heterogeneous compute systems



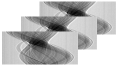
How?

- Define tasks of work
- Connect processing workflows
- Let a run-time schedule tasks

Overview of the compute system

Requirements

- Process image streams on the fly
- Use heterogeneous compute systems



How?

- Define tasks of work
- Connect processing workflows
- Let a run-time schedule tasks

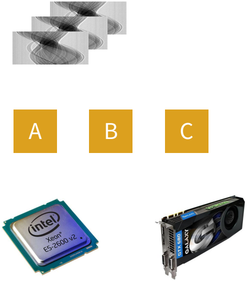
Overview of the compute system

Requirements

- Process image streams on the fly
- Use heterogeneous compute systems

How?

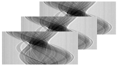
- Define tasks of work
- Connect processing workflows
- Let a run-time schedule tasks



Overview of the compute system

Requirements

- Process image streams on the fly
- Use heterogeneous compute systems



How?

- Define tasks of work
- Connect processing workflows
- Let a run-time schedule tasks

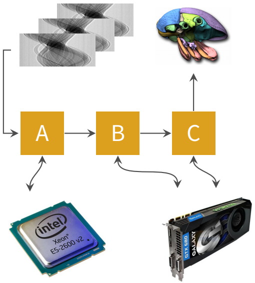
Overview of the compute system

Requirements

- Process image streams on the fly
- Use heterogeneous compute systems

How?

- Define tasks of work
- Connect processing workflows
- Let a run-time schedule tasks



Processing data streams

Execution model

- Sequence of multi-dimensional data x_k
- Tasks t_i generate, process and consume x_k's
- Computation ist inherently *sequential*

Parallelisation opportunities

Tasks have data dependencies but don't share state

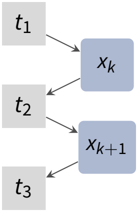
- Pipeline parallelism
- Data parallelism on multiple cores
- Data parallelism on GPUs

Processing data streams

Karlsruhe Institute of Technology

Execution model

- Sequence of multi-dimensional data x_k
- Tasks t_i generate, process and consume x_k's
- Computation ist inherently *sequential*



Parallelisation opportunities

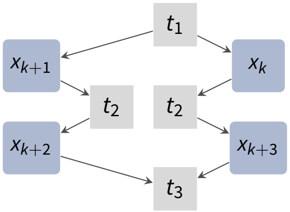
Tasks have data dependencies but don't share state

- Pipeline parallelism
- Data parallelism on multiple cores
- Data parallelism on GPUs

Processing data streams

Execution model

- Sequence of multi-dimensional data x_k
- Tasks t_i generate, process and consume x_k's
- Computation ist inherently *sequential*



Parallelisation opportunities

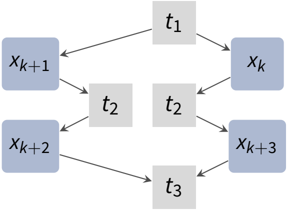
Tasks have data dependencies but don't share state

- Pipeline parallelism
- Data parallelism on multiple cores
- Data parallelism on GPUs

Processing data streams

Execution model

- Sequence of multi-dimensional data x_k
- Tasks t_i generate, process and consume x_k's
- Computation ist inherently *sequential*

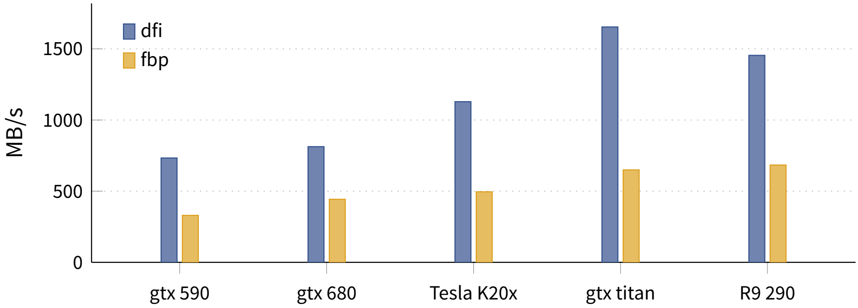


Parallelisation opportunities

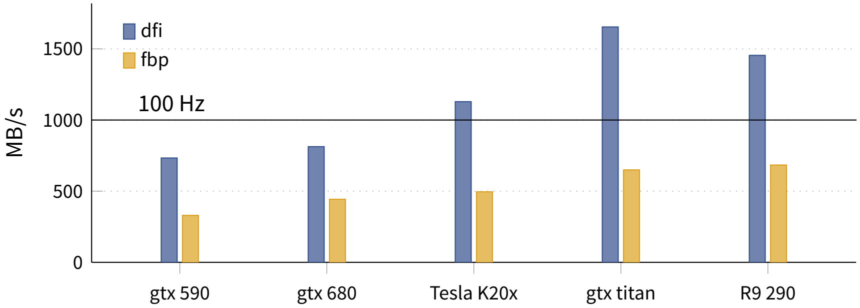
Tasks have data dependencies but don't share state

- Pipeline parallelism
- Data parallelism on multiple cores
- Data parallelism on GPUs

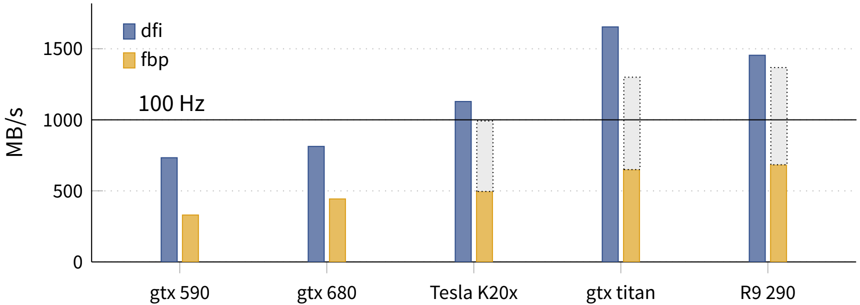
Reconstruction throughput



Reconstruction throughput



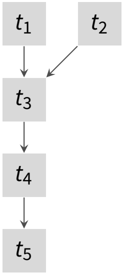
Reconstruction throughput



Multi GPU expansion

Static assignment

- Partitioning of tasks
- Identification of processing units
- Depth-first search for path identification
- Insertion of duplicates
- Mapping and execution



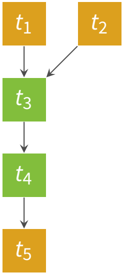
Benefits

- Avoids unnecessary data transfers
- Automatic scaling

Multi GPU expansion

Static assignment

- Partitioning of tasks
- Identification of processing units
- Depth-first search for path identification
- Insertion of duplicates
- Mapping and execution



Benefits

- Avoids unnecessary data transfers
- Automatic scaling

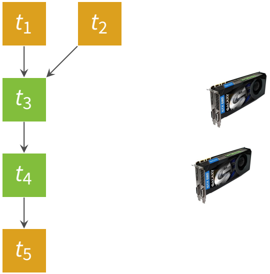
Multi GPU expansion

Static assignment

- Partitioning of tasks
- Identification of processing units
- Depth-first search for path identification
- Insertion of duplicates
- Mapping and execution

Benefits

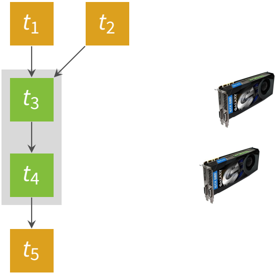
- Avoids unnecessary data transfers
- Automatic scaling



Multi GPU expansion

Static assignment

- Partitioning of tasks
- Identification of processing units
- Depth-first search for path identification
- Insertion of duplicates
- Mapping and execution



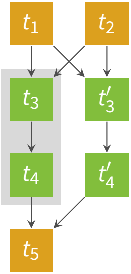
Benefits

- Avoids unnecessary data transfers
- Automatic scaling

Multi GPU expansion

Static assignment

- Partitioning of tasks
- Identification of processing units
- Depth-first search for path identification
- Insertion of duplicates
- Mapping and execution



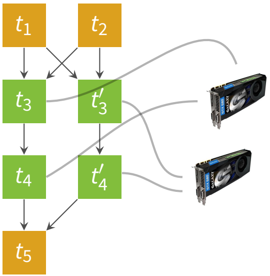
Benefits

- Avoids unnecessary data transfers
- Automatic scaling

Multi GPU expansion

Static assignment

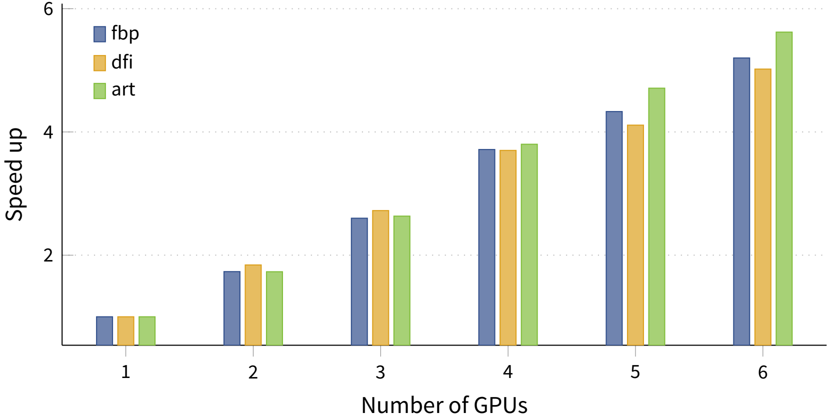
- Partitioning of tasks
- Identification of processing units
- Depth-first search for path identification
- Insertion of duplicates
- Mapping and execution



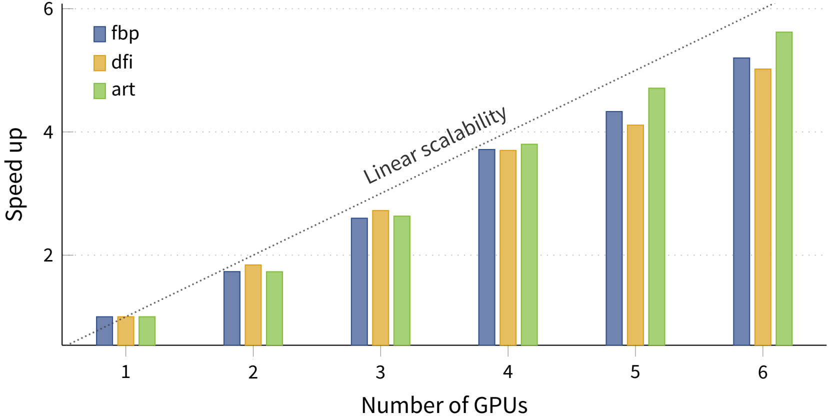
Benefits

- Avoids unnecessary data transfers
- Automatic scaling

Reconstruction scalability



Reconstruction scalability



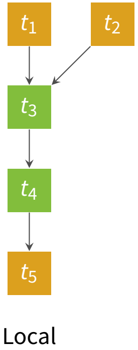
Multi node expansion

Strategy

- Again, we assume existing task graph
- Proxy task represents subpath
- Instantiate subpath remotely
- Send and receive input and output

Benefits

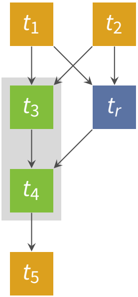
- Local multi gpu optimization
- Abstracts from network communication (InfiniBand via MPI, Zeromq)



Multi node expansion

Strategy

- Again, we assume existing task graph
- Proxy task represents subpath
- Instantiate subpath remotely
- Send and receive input and output



Local

Benefits

- Local multi gpu optimization
- Abstracts from network communication (InfiniBand via MPI, Zeromq)

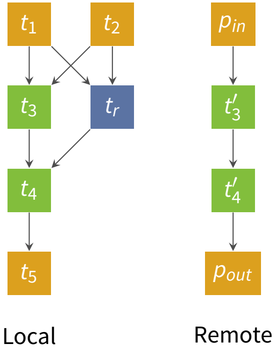
Multi node expansion

Strategy

- Again, we assume existing task graph
- Proxy task represents subpath
- Instantiate subpath remotely
- Send and receive input and output

Benefits

- Local multi gpu optimization
- Abstracts from network communication (InfiniBand via MPI, Zeromq)



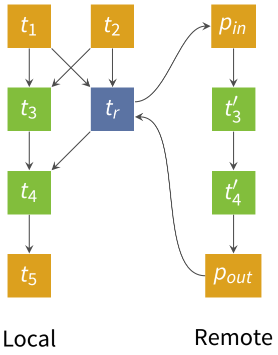
Multi node expansion

Strategy

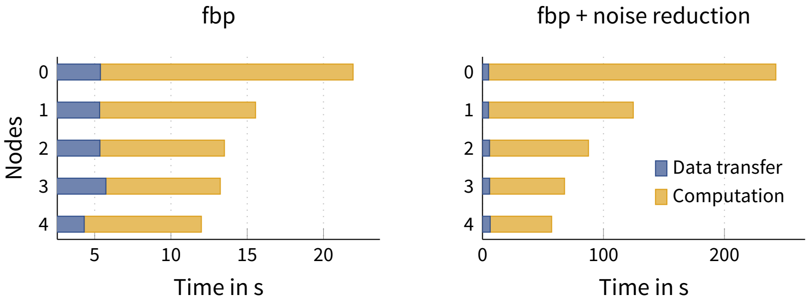
- Again, we assume existing task graph
- Proxy task represents subpath
- Instantiate subpath remotely
- Send and receive input and output

Benefits

- Local multi gpu optimization
- Abstracts from network communication (InfiniBand via MPI, Zeromq)



Scalability



Scalability limited by compute and data transfer ratio

Implementation

Karlsruhe Institute of Technology

Framework

- Concepts are implemented as a C library
- Accelerator devices are accessed through OpenCL
- Language bindings are provided through GObject

OpenCL

Applications

- Flat-correction, denoising, data conversion
- Filtered backprojection for tomographic and laminographic reconstruction
- Direct Fourier methods and algebraic techniques
- Feature detection, particle tracking

Usage

As a user

- Command line

```
$ ufo-launch read ! blur !  write filename=foo.tif
```

- Pre-defined JSON

```
$ ufo-runjson pipeline.json
```

- tango interface

Usage

As a user

- Command line

```
$ ufo-launch read ! blur !  write filename=foo.tif
```

- Pre-defined JSON

```
$ ufo-runjson pipeline.json
```

- tango interface *new*

Usage

Karlsruhe Institute of Technology

As a user

- Command line

```
$ ufo-launch read ! blur !  write filename=foo.tif
```

- Pre-defined JSON

```
$ ufo-runjson pipeline.json
```

- tango interface new

As a developer

- Directly via C API
- Through language bindings, e.g. standard Python
- High-level Python interface

Standard Python interface

```python
from gi.repository import Ufo

pm = Ufo.PluginManager()
read = pm.get_task('read')
opencl = pm.get_task('opencl')
write = pm.get_task('write')

read.set_properties(path='/home/data/*.tiff')
opencl.set_properties(source='...', kernel='...')
write.set_properties(filename='/home/out.tiff')

g = Ufo.TaskGraph()
g.connect_nodes(read, opencl)
g.connect_nodes(opencl, write)

sched = Ufo.Scheduler()
sched.run(g)
```

High-level Python

```python
from ufo import Read, Write, Opencl

read = Read(path='/home/data/*.tiff')
opencl = Opencl(source='...', kernel='...')
write = Write(filename='/home/out.tiff')

# write to disk
write(opencl(read())).run().wait()

# or use result
for image in opencl(read()):
    print(np.mean(image))
```

JSON representation

Karlsruhe Institute of Technology

```
{
  "nodes": [
    {"plugin": "read", "name": "read",
     "properties": {"path": "/home/data/*.tiff"}},
    {"plugin": "opencl", "name": "opencl",
     "properties": {"source": "...", "kernel": "..."}},
    {"plugin": "write", "name": "write",
     "properties": {"filename": "/home/out.tiff"}}
  ],
  "edges": [
    {"from": "read", "to": "opencl", "input": 0},
    {"from": "opencl", "to": "write", "input": 0}
  ]
}
```

OPENCL

Implementations

OpenCL is widely supported but …

Vendor	Rev.	GPU	CPU	FPGA	OS
NVIDIA	1.1	✓	–	–	🐧 🪟 🍎
AMD	2.0	✓	✓	–	🐧 🪟 🍎
Intel	2.0	✓	✓	–	🐧 🪟
Apple	1.2	✓	✓	–	🍎
Altera	1.0	–	–	✓	🐧 🪟

Programming model

Platform

- *Host* controls ≥ 1 *platforms* (i.e. vendor SDKs)
- A platform consists of ≥ 1 *devices* (CPU, GPU, FPGA)
- Host allocates resources and schedules execution
- Devices execute code assigned to them by the host

Programming model

Karlsruhe Institute of Technology

Platform

- *Host* controls ≥ 1 *platforms* (i.e. vendor SDKs)
- A platform consists of ≥ 1 *devices* (CPU, GPU, FPGA)
- Host allocates resources and schedules execution
- Devices execute code assigned to them by the host

Devices

- A single device has ≥ 1 *compute units*
- Each CU has ≥ 1 *processing elements*
- Mapping of CUs and PEs to hardware is *not* specified

Programming model II

Context

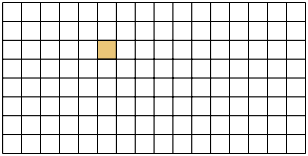
- A *context* encompasses devices of a *single* platform that want to share data
- Memory buffers are created within a context and *not* per device

Command queues

- Communication with a device is only possible through *command queues*
- Created within a context for a specific device
- Commands are data transfers and kernel executions
- Implicit and explicit synchronization of commands

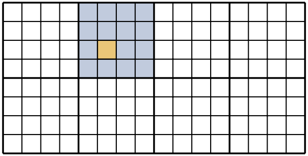
Execution model

- Work is arranged as work items on a 1D, 2D or 3D grid



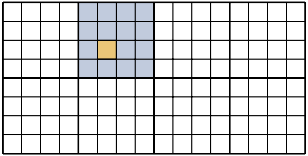
Execution model

- Work is arranged as work items on a 1D, 2D or 3D grid
- Grid is split into work groups



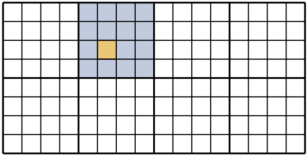
Execution model

- Work is arranged as work items on a 1D, 2D or 3D grid
- Grid is split into work groups
- Work groups are scheduled on one or more CUs



Execution model

- Work is arranged as work items on a 1D, 2D or 3D grid
- Grid is split into work groups
- Work groups are scheduled on one or more CUs
- Each work item executes a *kernel* on a PEs



Memory model

Memory, buffers and images

- Host *cannot* access device memory and vice versa
- Buffers transfer data between host and device memory
- Images are specially typed buffers

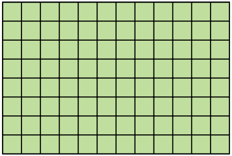
Memory model

Memory, buffers and images

- Host *cannot* access device memory and vice versa
- Buffers transfer data between host and device memory
- Images are specially typed buffers

Device memory

- Global, host-accessible, modifiable by *all* work items



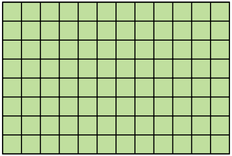
Memory model

Memory, buffers and images

- Host *cannot* access device memory and vice versa
- Buffers transfer data between host and device memory
- Images are specially typed buffers

Device memory

- Global, host-accessible, modifiable by *all* work items
- Constant, host-accessible, readable by *all* work items



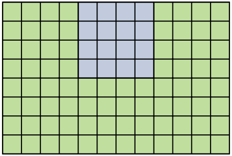
Memory model

Memory, buffers and images

- Host *cannot* access device memory and vice versa
- Buffers transfer data between host and device memory
- Images are specially typed buffers

Device memory

- Global, host-accessible, modifiable by *all* work items
- Constant, host-accessible, readable by *all* work items
- Local, modifiable by work group



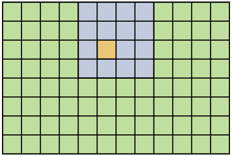
Memory model

Memory, buffers and images

- Host *cannot* access device memory and vice versa
- Buffers transfer data between host and device memory
- Images are specially typed buffers

Device memory

- Global, host-accessible, modifiable by *all* work items
- Constant, host-accessible, readable by *all* work items
- Local, modifiable by work group
- Private, modifiable by single work item



Kernel

- A kernel is a piece of C code executed by a work item

```
kernel void saxpy(float a, global float *x,
                  global float *y, global float *r) {
```

Kernel

- A kernel is a piece of C code executed by a work item

```
kernel void saxpy(float a, global float *x,
                  global float *y, global float *r) {
```

- To address data the work item identifies its position on the grid

```
    /* Global grid index */
    int gid = get_global(1) * get_global_size(0) +
              get_global_id(0);

    r[gid] = a * x[gid] + y[gid];
}
```

Kernel

- A kernel is a piece of C code executed by a work item

```
kernel void saxpy(float a, global float *x,
                  global float *y, global float *r) {
```

- To address data the work item identifies its position on the grid

```
    /* Global grid index */
    int gid = get_global(1) * get_global_size(0) +
              get_global_id(0);

    r[gid] = a * x[gid] + y[gid];
}
```

- It is crucial to map work items to data according to the task and constraints

Porting code to GPUs

1. Look for massive data parallel sections of code, i.e. `for` loop over large array is a prime example

   ```
   for (int i = 1; i < N-1; i++)
       x[i] = sin(y[i]) + 0.5 * (x[i-1] + x[i+1]);
   ```

2. Create kernel and compile that replaces the inner loop body

   ```
   i = get_global_id(0);
   x[i] = sin(y[i]) + 0.5 * (x[i-1] + x[i+1]);
   ```

3. Move data to device
4. Create, compile and run kernel
5. Move result to CPU

Pitfalls and solutions

- Not enough work causes underutilized PEs and poor latency hiding

  → Use finer grid to increase number of work items
- Poor data locality reduces attainable bandwidth

  → Adjacent work items should access adjacent memory locations
- PCIe bus can become a bottleneck (16 GB/s PCIe vs. 340 GB/s global)

  → Keep data on GPU for successive kernel executions
- Conditional execution serializes work item execution

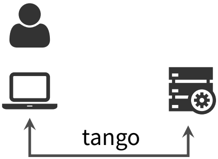
  → Put condition into computation

TANGO INTEGRATION

Preliminary draft

Protocol

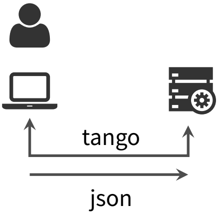
- tango server accepts compute requests



Preliminary draft

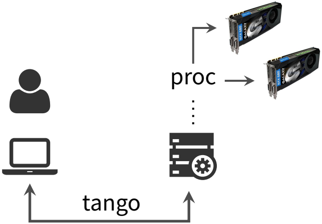
Protocol

- tango server accepts compute requests
- Client sets the `json` attribute and calls the `Run` or `RunContinuous` command



Protocol

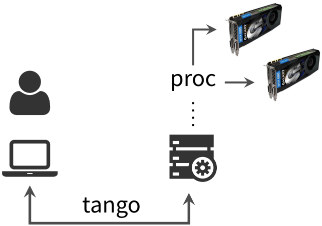
- tango server accepts compute requests
- Client sets the `json` attribute and calls the `Run` or `RunContinuous` command
- The server spawns a new compute process identified by a process id



Preliminary draft

Protocol

- tango server accepts compute requests
- Client sets the `json` attribute and calls the `Run` or `RunContinuous` command
- The server spawns a new compute process identified by a process id



Execution models

1. Single-run processes (“fire and forget”)
2. Continuous processes (update description and re-run)

Single-run processes

Karlsruhe Institute of Technology

Interface

```
process = PyTango.DeviceProxy('hzgctkit/process/1')
process.json = "{ ... }"
pid = process.Run()
print(process.Running(pid))   # status of, e.g. True
print(process.jobs)           # active jobs, e.g. [7041]
process.Wait(pid)
print(process.ExitCode(pid))  # return code of job
```

Remarks

- Simple to use and understand
- No prolonged hogging of resources

Continuous processes

Karlsruhe Institute of Technology

Interface

```
pid = process.RunContinuous()
process.Continue(pid)       # trigger execution
process.json = "{ ... }"    # update description
process.Continue(pid)
process.Stop(pid)           # terminate process
```

Remarks

- Allows for quicker results
- Resources are allocated as long as process is running
- Forgetting to call `Stop` leaks resources
- Real concurrency *not* solved yet

Future efforts

Karlsruhe Institute of Technology

Framework

- Additional pure InfiniBand messenger besides Zeromq and MPI
- Enhance scheduling with run-time information

Tools on top

- Update TomoPy integration (interfaces are breaking constantly …)
- Finish web-based reconstruction and visualization prototype
- Stabilize tango interface and improve error handling

Thanks for your attention.