Handling massive concurrency

Development of a programming model for GPU and CPU

Matthias Liedtke, SAP
April 08, 2019

PUBLIC

Agenda

Our requirements and related concepts

The programming model

Results

Our requirements and related concepts

Context of the programming model

Llang compiler

- Just in time compiler in existing server environment using the LLVM backend
- Llang → internal language with little performance overhead compared to C++

Our requirements for the programming model

Ease-of-Use

Achieve comparable performance to CUDA

Write once

Supportability

Existing GPU programming models

OpenMP

- Sequential program
- Added pre-processor directives for parallelization

→Limited expressiveness as parallelization is „on top“ of programming language

Existing GPU programming models

CUDA

- Strong support for hardware capabilities
- Many libraries for special needs
- C-style interface, little abstraction

→Limited to Nvidia GPUs, no CPU execution possible

Existing GPU programming models

OpenCL

- Platform independent programming of highly parallel kernels
- Hardware abstraction
- Mature (but complex) interface, also in C++

→ Very close to what we need

→ No integration into existing environment

The programming model

Example usage of the programming model

```
_acc_kernel Void multiply(acc::GridInfo& gi, ForeignArray<Int32>& data) {
    Size index = gi.getThreadIdX() + gi.getBlockIdX() * gi.getThreadCountX();
    data[index] = data[index] * Int32(index);
}

export Void testMain() {
    Size blockCount = 32z;
    Size threadCount = 32z;
    ForeignArray<Int64> array = /*...*/;
    acc::GridConfig config(blockCount, threadCount);
    acc::gridInvoke(config, _bind(multiply, array));
}
```

Programming model – Kernel invocation

```
_acc_kernel Void multiply(acc::GridInfo& gi, ForeignArray<Int32>& data) { }
```

- Keyword `_acc_kernel`
- Function will be compiled for the CPU and GPU backend

```
acc::GridConfig config(blockCount, threadCount);
acc::gridInvoke(config, _bind(multiply, array));
```

- GridConfig to set number of threads and thread groups
- Kernel function bound with arguments

Programming model – Data transfer

```
acc::gridInvoke(config, _bind(multiply, array));
```

Data transfer handled by invoke mechanism

Programming model – Explicit data transfers

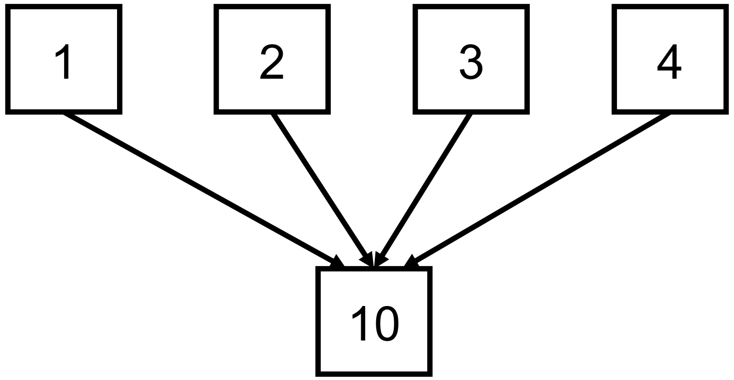
```
_acc_kernel Void multiply(acc::GridInfo& gi, ForeignArray<Int32>& data) {
}

export Void testMain() {
    Size blockCount = 32z;
    Size threadCount = 32z;
    ForeignArray<Int32> array = /*...*/;
    acc::Stream stream;
    {
        acc::GridConfig config(blockCount, threadCount);
        acc::Transfer arrayTransfer(array, stream);
        acc::gridInvoke(config, _bind(multiply, array), stream);
        acc::gridInvoke(config, _bind(multiply, array), stream);
    } // end of lifetime of transfer object triggers transfer
}
```

Programming model – Reduction

Aggregating multiple results into one, e.g. sum

```
_reduce(gridInfo, COMPLETE_GRID, partialResult, add, &result);
```



Programming model – Execution phases

Aim: Avoid self defined locks and dead locks

Concept: Have phases that are handled “sequentially”

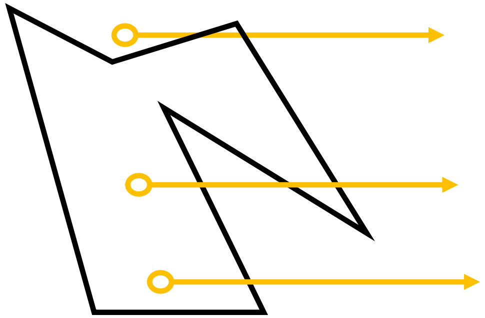
```
_acc_kernel Void kernel(acc::GridInfo& gi, ForeignArray<Int32> in, ForeignArray<Int32>& out) {
  _acc_shared ForeignArray<Int32> inShared;
  _acc_shared ForeignArray<Int32> outShared;
  _phased_execution "load" {
    // load data from in to inShared
  }
  _phased_execution "process" {
    // execution operation reading data from inShared and storing results in outShared
  }
  _phased_execution "aggregate" {
    // aggregate results in outShared
  }
}
```

Results

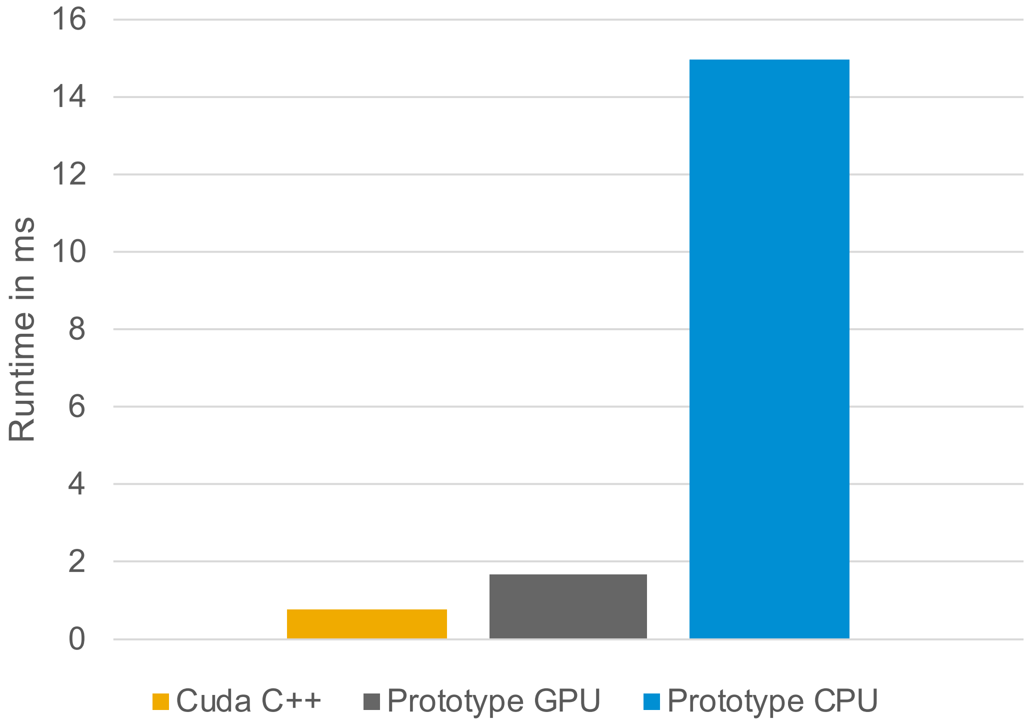
Points-in-polygon

For each point p:

- Count intersections of ray starting at p with polygon
- Even number: outside
- Odd number: inside



Kernel runtime
with 5’000 points and 10’000 edges
GPU: Nvidia Tesla P100
CPU: 4 x Intel Xeon E7-8880 v2 @2.5 GHz



Summary

Main concepts of our programming model

- Worker / kernel function like in CUDA / OpenCL
- Context object for multiple kernel calls (“Stream”); comparable to CUDA stream
- Object for kernel invocation configuration
- Object to handle explicit GPU transfer for an existing variable on CPU
- Execution phases to avoid explicit locks
- GPU and CPU backend with GPU focus

Thank you.

Contact information:

Matthias Liedtke
matthias.Liedtke@sap.com

THE BEST RUN SAP