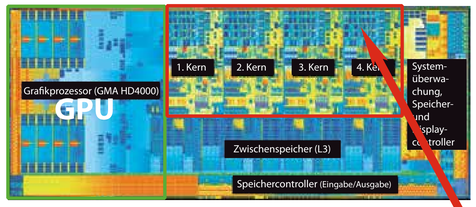
High-Performance Domain-Specific Languages for GPU Computing

Richard Membarth, Philipp Slusallek
Computer Graphics Lab, Saarland University

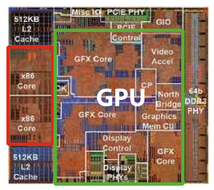
Marcel Köster, Roland Leißa, Sebastian Hack
Compiler Design Lab, Saarland University

Motivation

- Many-Core HW is everywhere
  - But cannot be programmed well

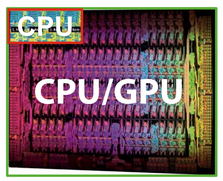


Intel Haswell Architecture (1.4B Transistors)

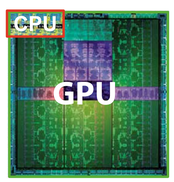


AMD Brazo

Traditional Programs run only on a single core



Intel KnightsFerry
(~5B Transistors)

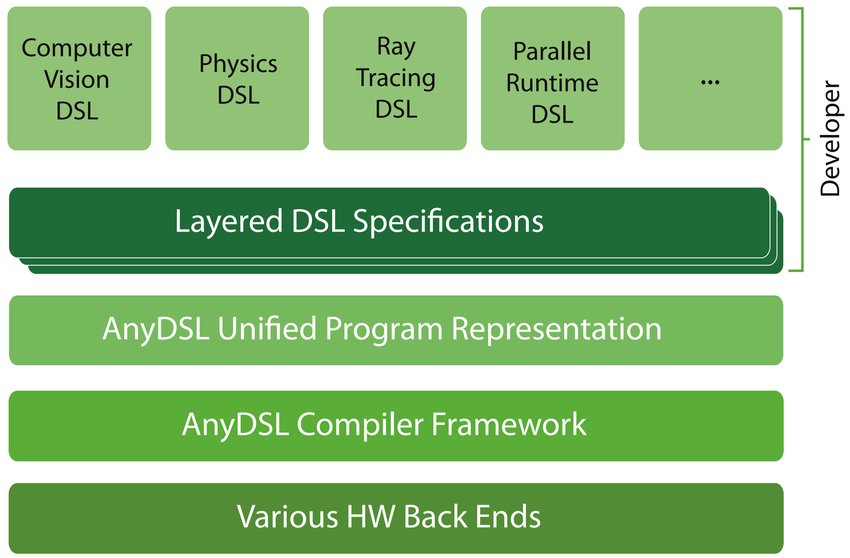


Nvidia Kepler
(~7B Transistors)

CPU: getting smaller & smaller (constant # of transistors)

Future massively parallel many-core processor

AnyDSL Architecture



- Back Ends
  - GPU (NVVM, SPIR)
  - CPU (x86, ARM, …)
- DSL Embedding
  - Impala as host-language
    - Stencil computation
    - Image processing

A DSL for Stencil Codes

- Embedded in Impala
- Impala features a partial evaluator
  - Partial evaluation is triggered by annotations

```
fn main() {
  let mut arr: [float] = array(width, height);
  let mut out: [float] = array(width, height);
  let a = 0.2f, b = 1.0f - 4.0f * a;
  let stencil = [
    [0, b, 0],
    [b, a, b],
    [0, b, 0]];

  foreach i in iteration(width, height) {
    out[i] = @apply_stencil(arr, stencil, i);
  }
}
```

- Application-specific code
  - Applies the stencil to a single pixel

Mapping to Target Hardware

- Scheduling & optimization
  - Target-specific implementations for the iteration function
  - Compiler exposes NVVM code generation through nvvm function

```
fn iteration(width : int, height : int,
             body : fn(int) -> void
            ) -> void {
  nvvm(width * height, || -> void {
    let tid_x = nvvm_tid_x() + nvvm_ntid_x() * nvvm_ctaid_x();
    let tid_y = nvvm_tid_y() + nvvm_ntid_y() * nvvm_ctaid_y();
    let index = tid_y * width + tid_x;
    body(index);
  });
}
```

- Also support for vectorization and SPIR

First Results

- Compilation process: Impala → LLVM IR
- Mapping for GPU execution:
  - Annotated LLVM IR
    - → NVVM IR for CUDA
    - → SPIR for OpenCL 1.2
- Results for Jacobi Kernel

	GTX 580	GTX 680
CUDA (hand-specialized)	0.32	0.35
CUDA (hand-tuned)	0.26	0.23
Impala (specialized)	0.32	0.35
Impala (+ tuned)	0.24	0.23

Time in ms for the Jacobi kernel an image of 2048x2048 pixels