Compiling Parallel Languages with the NVIDIA Compiler SDK

Mark Harris, Chief Technologist GPU Computing
A Platform for *Diverse* Parallel Computing

Developers want to build front-ends for Java, Python, R, DSLs ... Target other processors like ARM, FPGAs, GPUs, x86 ...
LLVM Open-Source Compiler Infrastructure

- Commercially Popular
  AMD, Apple, Adobe, Cray, Intel, NVIDIA...
- Modular & Reusable
- Extensible Optimization Suite
- Easy Development
- Multiple Language Front-ends
- Simple low-level language (IR)
Typical LLVM Compiler Architecture

- Language Source
- Front End
- LLVM IR
- LLVM Optimizer
- Optimized IR
- Assembly CodeGen
- Machine Assembly

IR = Intermediate Representation
CUDA Compiler Architecture (4.1, 4.2, 5.0...)

NVVM = LLVM with NVIDIA parallel extensions

PTX = NVIDIA Virtual Machine Assembly
NVIDIA Open Compiler Architecture

- CUDA Front End
- NVVM IR
- LLVM Optimizer
- Optimized IR
- NVPTX CodeGen
- PTX Assembly
- PTXAS

Libraries:
- libCUDA.LANG
- libNVVM
- Open Sourced

NVCC

© 2012 NVIDIA
Scenarios
Building Production-Quality Compilers

- NVCC
- CUDA Runtime
- libCUDA.LANG
- libNVVM
- OpenACC
- Fortran
- Python / Numpy

© 2012 NVIDIA
Building Domain-Specific Languages

- libCUDA.LANG
- libNVVM
- NVCC
- DSL Front End
- DSL Runtime
- CUDA Runtime
- MATLAB
- JET
- HALIDE
Enabling Other Platforms

- **libCUDA.LANG**
- **libNVVM**
- **NVCC**
- **x86 LLVM Backend**
- **x86 CUDA Runtime**

CUDA Runtime
Enabling Research in GPU Computing

CU++

CLANG'

x86 LLVM Backend

libNVVM

Custom Runtime

© 2012 NVIDIA
Some Current NVVM / LLVM PTX Users

- Numba Pro: array-oriented compiler for Numpy/Python
- Halide image processing language: MIT (halide-lang.org)
- Jet fluid dynamics DSL: Double Negative
- Alea.CUDA, F# on the GPU: QuantAlea
- Delite parallel EDSL framework: Stanford PPL
Contributed NVPTX code generator sources back to LLVM trunk

Sources available directly from LLVM!
  - Standard LLVM license

Invite community to:
  - Use it!
  - Contribute improvements to it!
What’s in the NVIDIA Compiler SDK?

- libNVVM
- NVVM IR specification
- Code Samples
NVVM Intermediate Representation

- NVVM IR is based on LLVM IR
  - Targeted at GPU computing

- IR Specification describes:
  - Address spaces
  - Kernels and device functions
  - Intrinsics
  - ... more
NVVM IR and LLVM IR

- NVVM IR
  - Based on LLVM IR
  - With a set of rules and intrinsics

No new types. No new operators. No new reserved words.

- An NVVM IR program can work with any standard LLVM IR tool
  - `llvm-as`
  - `llvm-link`
  - `llvm-extract`
  - `llvm-dis`
  - `llvm-ar`
  - `...`

- An NVVM IR program can be built with the standard LLVM distribution.
  - `svn co http://llvm.org/svn/llvm-project/llvm/branches/release_30 llvm`
A binary component library targeting PTX 3.1
- Fermi, Kepler and later architectures
- 32-bit and 64-bit hosts, Windows, Linux and MacOS
- NVVM IR verifier

API document
Sample applications
libCUDA.LANG

CUDA C++ front-end binary library
- takes CUDA source program, generates NVVM IR and host C++ program for supported platforms

A future part of the NVIDIA compiler SDK
- Not released with the current preview SDK
Availability Roadmap

- **Open source NVPTX backend**
  - May 2012

- **NVVM IR spec, libNVVM, samples**
  - Preview release (May 2012)
  - Production release (Next CUDA release after version 5.0)

- **libCUDA.LANG**
  - To be announced
Example: “Rg”

Dynamic compilation of R on a GPU
R is a language and environment for statistical computing and graphics - www.r-project.org

Dynamically compile R code and execute it on a GPU
  - Supports a useful subset of R

An example of how to accelerate a DSL using libNVVM
Rg

> v1 = c(1, 2, 3, 4)
> dv1 = rg.dv(c(1, 2, 3, 4))
> dv2 = rg.dv(c(10, 20, 30, 40))
> dv3 = rg.gapply(function(x, y) {x+y;}, dv1, dv2)
> as.double(dv3)

[1] 11 22 33 44
# define the code for Mandelbrot
mandelbrot <- function(x0, y0) {
    iteration <- 0L;
    max_iteration <- 50L;
    x <- 0;
    y <- 0;
    while ( (x*x + y*y < 4) && (iteration < max_iteration) ) {
        xtemp <- x*x - y*y + x0;
        y = 2*x*y + y0;
        x = xtemp;
        iteration = iteration + 1L;
    }
    color = iteration;
    color;
}

# create data
dv_points_x=rg.dv(points_x);
dv_points_y=rg.dv(points_y);

# compile and run and get results!
dv_points_color = rg.gapply(mandelbrot, dv_points_x, dv_points_y);
colorvec = as.integer(dv_points_color);
Rg Mandelbrot Results

- Run on a laptop...
  - Lenovo T520

- CPU (vanilla R): \(~6\) seconds
  - Intel i7-2620 2.6GHz (1 core)

- GPU (rg.gapply): \(~0.6\) seconds including JIT compile
  - NVIDIA NVS 4200 (1 Fermi SM)

- 10x speedup
  - Huge performance potential in accelerating high-productivity languages
Try the NVIDIA Compiler SDK

  Open to all NVIDIA registered developers

File bugs and post questions to NVIDIA forums
  http://devtalk.nvidia.com