A simple tutorial on generating PTX assembler out of Ada source code using LLVM NVPTX backend

Dmitry Mikushin

dmitry.mikushin@usi.ch

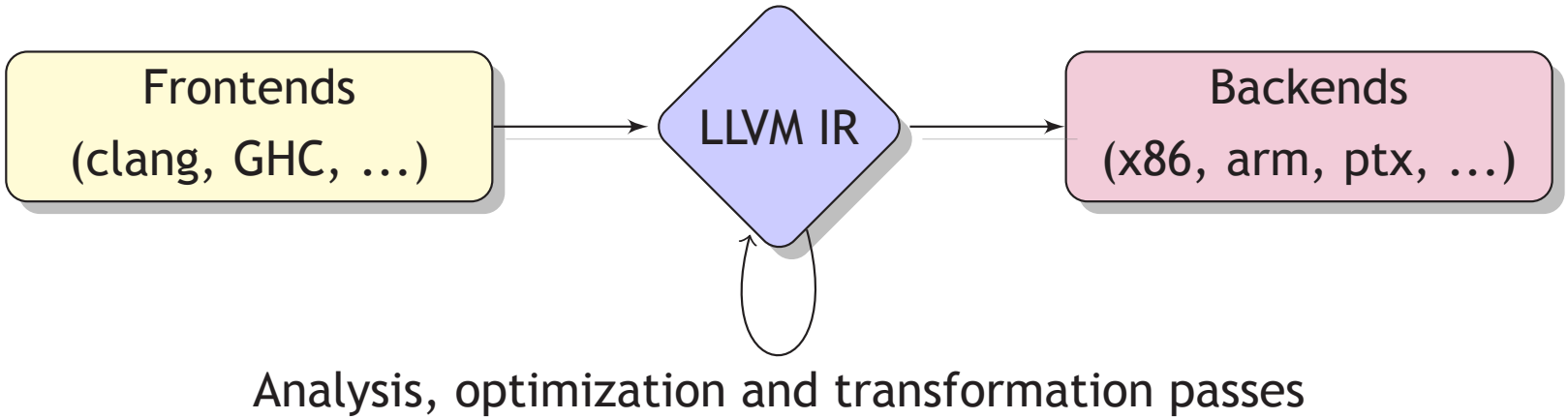
June 14, 2012

Abstract

This presentation provides a simple jump-start tutorial for newbies to start doing something useful with LLVM, DragonEgg, NVPTX backend and some custom high-level language. Along with short software overview from the user perspective it contains instructions on building components necessary to experiment with NVPTX and an example usecase of generating PTX assembler out of the Ada source code with help of DragonEgg.

LLVM

- A universal system of programs analysis, transformation and optimization with RISC-like intermediate representation (LLVM IR SSA)



New NVPTX backend for LLVM

- Recently commited to LLVM trunk, likely to be released with 3.2 (not 3.1!)
- Replaces old PTX backend
- NVPTX is fully compatible with old PTX backend
- Note NVPTX is a **backend**, means you can compile PTX only from LLVM IR

Build LLVM with NVPTX backend

```
$ mkdir -p ~/sandbox/src
$ cd ~/sandbox/src
$ svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
$ cd llvm/
$ mkdir build
$ cd build
$ ../configure --prefix=$HOME/sandbox --enable-shared --enable-targets=host,nvptx
$ make -j24
$ make install
```

Build with NVPTX backend

Where to get a CUDA frontend?

- NVPTX is a backend – means to compile CUDA with LLVM one still needs a frontend for target language
- NVIDIA announced libCUDA.LANG, but is not yet released, even does not have a schedule
- Fortunately, LLVM clang has limited CUDA support remained from the old PTX backend

DragonEgg – GCC as LLVM frontend

- Integrates the LLVM optimizers and code generator with the GCC parsers
- Supports GCC up to 4.6, not 4.7 yet
- Translates gcc gimple IR into LLVM IR:
  - Allows LLVM to compile Ada, Fortran, and other languages supported by the GCC compiler frontends
  - Could provide additional code optimization

Compiling gcc with Ada frontend

- DragonEgg works on top of GCC and LLVM installations. Here's how to build GCC from source

```
$ svn checkout svn://gcc.gnu.org/svn/gcc/branches/gcc-4_6-branch gcc
$ cd gcc/
$ mkdir build
$ cd build/
$ sudo apt-get install libgmp-dev libmpfr-dev libmpc-dev
$ sudo apt-get install gnat gnat-4.6 flex libc6-dev-i386
$ ../configure --prefix=$HOME/sandbox/ --enable-languages=ada
$ LIBRARY_PATH=/usr/lib/x86_64-linux-gnu C_INCLUDE_PATH=/usr/include/x86_64-linux-↩
    gnu make -j12
$ LIBRARY_PATH=/usr/lib/x86_64-linux-gnu make install
```

Compiling DragonEgg

- Fetch the source code and compile it, pointing to the LLVM and GCC binary roots

```
$ svn co http://llvm.org/svn/llvm-project/dragonegg/trunk dragonegg
$ cd dragonegg
$ LLVM_CONFIG=$HOME/sandbox/bin/llvm-config CPLUS_INCLUDE_PATH=$HOME/sandbox/↩
    include GCC=$HOME/sandbox/bin/gcc make
$ cp dragonegg.so $HOME/sandbox/lib/
$ echo "$HOME/sandbox/bin/gcc -fplugin=$HOME/sandbox/lib/dragonegg.so -fplugin-arg-↩
    dragonegg-emit-ir -fplugin-arg-dragonegg-llvm-ir-optimize=0 -S $@" > $HOME/↩
    sandbox/bin/dragonegg
$ chmod +x $HOME/sandbox/bin/dragonegg
```

Example: GPU kernel in Ada

- Let's use new LLVM PTX to bring GPU support to some non-CUDA language, for example to Ada
- Just a *proof of concept*: only the general language support, no intrinsics or new keywords, i.e. no modifications in Ada frontend
- Use plain functions for reading CUDA compute grid
- Use _kernel/_device suffixes to identify CUDA kernels code
- No address spaces and global data support

sum_kernel.c

For example, let's write the following CUDA C kernel in Ada:

```
struct uint3 { unsigned int x, y, z; };
struct uint3 extern const threadIdx, blockIdx, blockDim;

__attribute__((global)) __attribute__((__used__)) void sum_kernel(float* a, float* ↩
    b, float* c)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    c[idx] = a[idx] + b[idx];
}
```

sum_kernel.ads (Ada header)

```
package sum_kernel is

function threadIdx_x return integer;
function blockIdx_x return integer;
function blockDim_x return integer;

pragma Import (C, threadIdx_x, "threadIdx_x");
pragma Import (C, blockIdx_x, "blockIdx_x");
pragma Import (C, blockDim_x, "blockDim_x");

type vector is array (natural range <>) of float;

procedure sum_kernel (a, b : in vector; c: out vector);

pragma Export (C, sum_kernel, "sum_kernel");

end sum_kernel;
```

sum_kernel.ads (Ada header)

```
package sum_kernel is

function threadIdx_x return integer;
function blockIdx_x return integer;
function blockDim_x return integer;

pragma Import (C, threadIdx_x, "threadIdx_x");
pragma Import (C, blockIdx_x, "blockIdx_x");
pragma Import (C, blockDim_x, "blockDim_x");

type vector is array (natural range <>) of float;

procedure sum_kernel (a, b : in vector; c: out vector);

pragma Export (C, sum_kernel, "sum_kernel");

end sum_kernel;
```

CUDA compute grid accessors

sum_kernel.ads (Ada header)

```
package sum_kernel is

function threadIdx_x return integer;
function blockIdx_x return integer;
function blockDim_x return integer;

pragma Import (C, threadIdx_x, "threadIdx_x");
pragma Import (C, blockIdx_x, "blockIdx_x");
pragma Import (C, blockDim_x, "blockDim_x");

type vector is array (natural range <>) of float;

procedure sum_kernel (a, b : in vector; c: out vector);

pragma Export (C, sum_kernel, "sum_kernel");

end sum_kernel;
```

CUDA kernel function

sum_kernel.adb (Ada source)

```
package body sum_kernel is

pragma suppress(range_check);

procedure sum_kernel (a, b : in vector; c: out vector) is

idx: integer;

begin

idx := threadIdx_x + blockIdx_x * blockDim_x;
c(idx) := a(idx) + b(idx);

end sum_kernel;

end sum_kernel;
```

sum_kernel.adb (Ada source)

```
package body sum_kernel is

pragma suppress(range_check);

procedure sum_kernel (a, b : in vector; c: out vector) is

idx: integer;

begin

idx := threadIdx_x + blockIdx_x * blockDim_x;
c(idx) := a(idx) + b(idx);

end sum_kernel;

end sum_kernel;
```

Suppress range check or otherwise it will introduce host function calls from kernel

Implementing custom cudada LLVM pass

In order to produce valid PTX from Ada-generated LLVM IR, create a new LLVM transformation pass:

- To replace compute grid functions with LLVM ptx intrinsics
- To identify and mark GPU functions
- To strip the rest of the source code

Implementing custom cudada LLVM pass

- Module pass definition

```
#include "llvm/Support/TypeBuilder.h"
#include "llvm/Pass.h"
#include "llvm/Function.h"
#include "llvm/Module.h"
#include "llvm/Support/Debug.h"

using namespace llvm;
using namespace std;
class CudaAda : public ModulePass
{
public:
    static char ID;
    CudaAda() : ModulePass(ID) { }
    virtual bool runOnModule(Module &m);
};

char CudaAda::ID = 0;
static RegisterPass<CudaAda> A("cudada", "Mark CUDA kernels and bind intrinsics for Ada-generated LLVM IR");
Pass* createCudaAdaPass() { return new CudaAda(); }
```

Implementing custom cudada LLVM pass

- Module pass definition

```
#include "llvm/Support/TypeBuilder.h"
#include "llvm/Pass.h"
#include "llvm/Function.h"
#include "llvm/Module.h"
#include "llvm/Support/Debug.h"

using namespace llvm;
using namespace std;
class CudaAda : public ModulePass
{
public:
    static char ID;
    CudaAda() : ModulePass(ID) { }
    virtual bool runOnModule(Module &m);
};

char CudaAda::ID = 0;
static RegisterPass<CudaAda> A("cudada", "Mark CUDA kernels and bind intrinsics for Ada-generated LLVM IR");
Pass* createCudaAdaPass() { return new CudaAda(); }
```

The pass to work at LLVM module scope

Implementing custom cudada LLVM pass

- Module pass definition

```
#include "llvm/Support/TypeBuilder.h"
#include "llvm/Pass.h"
#include "llvm/Function.h"
#include "llvm/Module.h"
#include "llvm/Support/Debug.h"

using namespace llvm;
using namespace std;
class CudaAda : public ModulePass
{
public:
    static char ID;
    CudaAda() : ModulePass(ID) { }
    virtual bool runOnModule(Module &m);
};

char CudaAda::ID = 0;
static RegisterPass<CudaAda> A("cudada", "Mark CUDA kernels and bind intrinsics for Ada-generated LLVM IR");
Pass* createCudaAdaPass() { return new CudaAda(); }
```

Register a pass to make it accessible by a -cudada flag from the shared library

Implementing custom cudada LLVM pass

- runOnModule function implementation

```
string kernel_suffix = "_kernel";
string device_suffix = "_device";
vector<Function*> erase;
for (Module::iterator f = m.begin(), fe = m.end(); f != fe; f++) {
    string name = f->getName();

    if (name == "llvm.ptx.read.tid.x") continue;
    if (name == "llvm.ptx.read.ntid.x") continue;
    if (name == "llvm.ptx.read.ctaid.x") continue;

    if (name.length() >= kernel_suffix.length())
        if (name.compare(name.length() - kernel_suffix.length(),
            kernel_suffix.length(), kernel_suffix) == 0) {
        f->setCallingConv(CallingConv::PTX_Kernel);
        continue;
    }
    if (name.length() >= device_suffix.length())
        if (name.compare(name.length() - device_suffix.length(),
            device_suffix.length(), device_suffix) == 0) {
        f->setCallingConv(CallingConv::PTX_Device);
        continue;
    }
```

Implementing custom cudada LLVM pass

- runOnModule function implementation

```
string kernel_suffix = "_kernel";
string device_suffix = "_device";
vector<Function*> erase;
for (Module::iterator f = m.begin(), fe = m.end(); f != fe; f++) {
    string name = f->getName();

    if (name == "llvm.ptx.read.tid.x") continue;
    if (name == "llvm.ptx.read.ntid.x") continue;
    if (name == "llvm.ptx.read.ctaid.x") continue;

    if (name.length() >= kernel_suffix.length())
        if (name.compare(name.length() - kernel_suffix.length(),
            kernel_suffix.length(), kernel_suffix) == 0) {
        f->setCallingConv(CallingConv::PTX_Kernel);
        continue;
    }
    if (name.length() >= device_suffix.length())
        if (name.compare(name.length() - device_suffix.length(),
            device_suffix.length(), device_suffix) == 0) {
        f->setCallingConv(CallingConv::PTX_Device);
        continue;
    }
```

Iterate through all module functions and find those with _kernel/_device prefix

Implementing custom cudada LLVM pass

```
    // Replace particular Ada functions with intrinsics.
    if (name == "threadIdx_x")
        f->replaceAllUsesWith(llvm_ptx_read_tid_x);
    if (name == "blockIdx_x")
        f->replaceAllUsesWith(llvm_ptx_read_ntid_x);
    if (name == "blockDim_x")
        f->replaceAllUsesWith(llvm_ptx_read_ctaid_x);
    erase.push_back(f);
}
for (vector<Function*>::iterator i = erase.begin(), e = erase.end(); i != e; i++)
    (*i)->eraseFromParent();

while (m.global_begin() != m.global_end())
    m.global_begin()->eraseFromParent();
```

- Compile LLVM pass into a shared library

```
$ g++ -D__STDC_LIMIT_MACROS -D__STDC_CONSTANT_MACROS -I/home/dmikushin/sandbox/bin↩
    /../include -fPIC -shared cudada.cpp -o libcudada.so -L/home/dmikushin/sandbox↩
    /bin/../lib/ -lLLVM-3.2svn
```

Implementing custom cudada LLVM pass

```
    // Replace particular Ada functions with intrinsics.
    if (name == "threadIdx_x")
        f->replaceAllUsesWith(llvm_ptx_read_tid_x);
    if (name == "blockIdx_x")
        f->replaceAllUsesWith(llvm_ptx_read_ntid_x);
    if (name == "blockDim_x")
        f->replaceAllUsesWith(llvm_ptx_read_ctaid_x);
    erase.push_back(f);
}
for (vector<Function*>::iterator i = erase.begin(), e = erase.end(); i != e; i++)
    (*i)->eraseFromParent();

while (m.global_begin() != m.global_end())
    m.global_begin()->eraseFromParent();
```

Replace certain function calls with LLVM ptx intrinsics

- Compile LLVM pass into a shared library

```
$ g++ -D__STDC_LIMIT_MACROS -D__STDC_CONSTANT_MACROS -I/home/dmikushin/sandbox/bin↩
    /../include -fPIC -shared cudada.cpp -o libcudada.so -L/home/dmikushin/sandbox↩
    /bin/../lib/ -lLLVM-3.2svn
```

Implementing custom cudada LLVM pass

```
    // Replace particular Ada functions with intrinsics.
    if (name == "threadIdx_x")
        f->replaceAllUsesWith(llvm_ptx_read_tid_x);
    if (name == "blockIdx_x")
        f->replaceAllUsesWith(llvm_ptx_read_ntid_x);
    if (name == "blockDim_x")
        f->replaceAllUsesWith(llvm_ptx_read_ctaid_x);
    erase.push_back(f);
}
for (vector<Function*>::iterator i = erase.begin(), e = erase.end(); i != e; i++)
    (*i)->eraseFromParent();

while (m.global_begin() != m.global_end())
    m.global_begin()->eraseFromParent();
```

Erase all other functions,
erase global variables

- Compile LLVM pass into a shared library

```
$ g++ -D__STDC_LIMIT_MACROS -D__STDC_CONSTANT_MACROS -I/home/dmikushin/sandbox/bin↩
    /../include -fPIC -shared cudada.cpp -o libcudada.so -L/home/dmikushin/sandbox↩
    /bin/../lib/ -lLLVM-3.2svn
```

LLVM IR for sum_kernel in Ada

```
$ ~/sandbox/bin/dragonegg -c sum_kernel.adb -o sum_kernel.ada.ll
$ LD_LIBRARY_PATH=LD_LIBRARY_PATH:. ~/sandbox/bin/opt -load libcudada.so -O3 -cudada -print-module sum_kernel.ada.ll -o sum_kernel.ada.ll.opt
```

```
define ptx_kernel void @sum_kernel(float* nocapture %a, float* nocapture %b, float* nocapture %c) uwtable {
entry:
  %0 = tail call i32 @llvm.ptx.read.tid.x()
  %1 = tail call i32 @llvm.ptx.read.ntid.x()
  %2 = tail call i32 @llvm.ptx.read.ctaid.x()
  %3 = mul i32 %2, %1
  %4 = add i32 %3, %0
  %5 = sext i32 %4 to i64
  %6 = add i64 %5, 2147483648
  %7 = getelementptr float* %a, i64 %6
  %8 = load float* %7, align 4
  %9 = getelementptr float* %b, i64 %6
  %10 = load float* %9, align 4
  %11 = fadd float %8, %10
  %12 = getelementptr float* %c, i64 %6
  store float %11, float* %12, align 4
  ret void
}
```

LLVM IR for sum_kernel in Ada

```
$ ~/sandbox/bin/dragonegg -c sum_kernel.adb -o sum_kernel.ir
$ LD_LIBRARY_PATH=LD_LIBRARY_PATH:. ~/sandbox/bin/opt -load libcudada.so -O3 -cudada -print-module sum_kernel.ada.ll -o sum_kernel.ada.ll.opt
```

Use our custom LLVM in optimizaion pipeline

```
define ptx_kernel void @sum_kernel(float* nocapture %a, float* nocapture %b, float* nocapture %c) uwtable {
entry:
  %0 = tail call i32 @llvm.ptx.read.tid.x()
  %1 = tail call i32 @llvm.ptx.read.ntid.x()
  %2 = tail call i32 @llvm.ptx.read.ctaid.x()
  %3 = mul i32 %2, %1
  %4 = add i32 %3, %0
  %5 = sext i32 %4 to i64
  %6 = add i64 %5, 2147483648
  %7 = getelementptr float* %a, i64 %6
  %8 = load float* %7, align 4
  %9 = getelementptr float* %b, i64 %6
  %10 = load float* %9, align 4
  %11 = fadd float %8, %10
  %12 = getelementptr float* %c, i64 %6
  store float %11, float* %12, align 4
  ret void
}
```

Finally, codegen Ada to PTX

```
$ ~/sandbox/bin/llc -march=nvptx64 sum_kernel.ada.ll.opt -o sum_kernel.ada.ptx
$ cat sum_kernel.ada.ptx
```

```
.entry sum_kernel(
.param .u64 .ptr .align 4 sum_kernel_param_0,
.param .u64 .ptr .align 4 sum_kernel_param_1,
.param .u64 .ptr .align 4 sum_kernel_param_2)
{
.reg .pred %p<396>;
.reg .s16 %rc<396>;
.reg .s16 %rs<396>;
.reg .s32 %r<396>;
.reg .s64 %rl<396>;
.reg .f32 %f<396>;
.reg .f64 %fl<396>;
// BB#0:                                // %entry
mov.u32 %r0, %tid.x;
mov.u32 %r1, %ntid.x;
mov.u32 %r2, %ctaid.x;
```

Finally, codegen Ada to PTX

```
$ ~/sandbox/bin/llc -march=nvptx64 sum_kernel.ada.ll.opt -o sum_kernel.ada.ptx
$ cat sum_kernel.ada.ptx
```

```
mad.lo.s32  %r0, %r2, %r1, %r0;
cvt.s64.s32  %rl0, %r0;
add.s64  %rl0, %rl0, 2147483648;
shl.b64  %rl0, %rl0, 2;
ld.param.u64  %rl1, [sum_kernel_param_0];
add.s64  %rl1, %rl1, %rl0;
ld.param.u64  %rl2, [sum_kernel_param_1];
add.s64  %rl2, %rl2, %rl0;
ld.global.f32  %f0, [%rl2];
ld.global.f32  %f1, [%rl1];
add.f32  %f0, %f1, %f0;
ld.param.u64  %rl1, [sum_kernel_param_2];
add.s64  %rl0, %rl1, %rl0;
st.global.f32  [%rl0], %f0;
ret;
}
```

It loads and works correctly

- A loader application deploys the resulting PTX with CUDA Driver API (see source code shipped together with this presentation)

```
$ ./loader 512 sum_kernel.ada.ptx
n = 512
Max diff = 0.000000 @ i = 0: 1.234571 != 1.234571
```

Conclusion

- LLVM NVPTX is quite usable in its current state
- However, NVPTX is not a full compiler, but a building block to create custom GPU compilers
- No full-featured CUDA frontend for LLVM yet

References

- The LLVM Compiler Infrastructure
- DragonEgg - Using LLVM as a GCC backend