# PTX Back-End: GPU Programming with LLVM

Justin Holewinski

The Ohio State University

LLVM Developer's Meeting
November 18, 2011

# Outline

# What is PTX?

- Parallel Thread eXecution
- Intermediate representation for nVidia GPUs
- Defines instructions for one thread (out of thousands)
- RISC-style instruction set
- JIT-compiled to device machine code at run-time
- Infinite, typed, declare-before-use register file
  `.f32, .u32, .b32, .s32`

```
1  .reg .u64 %rd<6>;
2  .reg .f32 %f<4>;
3  ld.param.u64    %rd1, [__cudaparm__Z3fooPf_a];
4  cvt.u64.u16     %rd2, %tid.x;
5  mul.lo.u64      %rd3, %rd2, 4;
6  add.u64         %rd4, %rd1, %rd3;
7  ld.global.f32   %f1, [%rd4+0];
8  add.f32         %f2, %f1, %f1;
9  st.global.f32   [%rd4+0], %f2;
10 exit;
```

# Why Target PTX?

GPU Computing

- Generate code for nVidia GPU devices
- Leverage new and existing LLVM IR transformations
- Ease development of new GPU-aware languages

# Why Target PTX?

GPU Computing

- Generate code for nVidia GPU devices
- Leverage new and existing LLVM IR transformations
- Ease development of new GPU-aware languages

Heterogeneous Computing

- Current: 2 TFlop/s of peak compute power
- Generate X86/ARM for bulk of program
- Generate PTX for off-loadable computations

# Outline

# The PTX Back-End

History:

- First commit in September 2010 by Che-Liang Chiou
- Active development in LLVM ToT
  - Holewinski, Chiou, Bailey
- Google Summer of Code 2011 project
- First released with LLVM 2.9 (very incomplete!)

# The PTX Back-End

History:

- First commit in September 2010 by Che-Liang Chiou
- Active development in LLVM ToT
  - Holewinski, Chiou, Bailey
- Google Summer of Code 2011 project
- First released with LLVM 2.9 (very incomplete!)

What Works:

- Most compute functionality
- Branching
- Special register access (thread ID, block ID, etc.)
- Device function calls (basic)
- Interop with Clang OpenCL support
- Shared memory regions

# How to Use the Back-End

Basic Usage:

- Compile LLVM with support for PTX back-end
  - Autotools: `--enable-targets=ptx`
  - CMake: `LLVM_TARGETS_TO_BUILD=PTX`
- Use `llc` with `ptx32` or `ptx64` target
  - `ptx32`: Use 32-bit addresses
  - `ptx64`: Use 64-bit addresses
  - Depends on host code, always 32-bit for OpenCL
- Use CUDA Driver API or nVidia OpenCL SDK to load/execute PTX

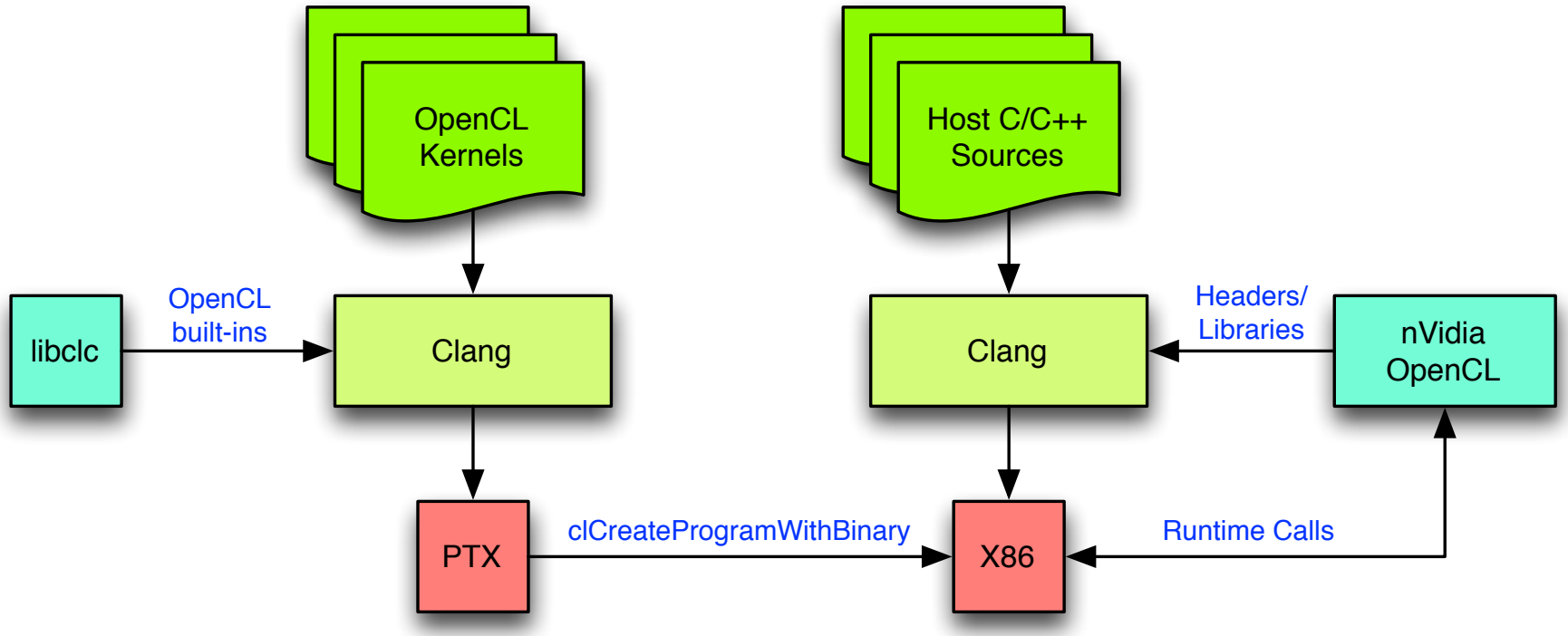
# CPU Options

```
compute_10 - Select the compute_10 processor.
compute_11 - Select the compute_11 processor.
compute_12 - Select the compute_12 processor.
compute_13 - Select the compute_13 processor.
compute_20 - Select the compute_20 processor.
fermi      - Select the fermi processor.
g80        - Select the g80 processor.
generic    - Select the generic processor.
gf100      - Select the gf100 processor.
gt200      - Select the gt200 processor.
sm_10      - Select the sm_10 processor.
sm_11      - Select the sm_11 processor.
sm_12      - Select the sm_12 processor.
sm_13      - Select the sm_13 processor.
sm_20      - Select the sm_20 processor.
sm_21      - Select the sm_21 processor.
sm_22      - Select the sm_22 processor.
sm_23      - Select the sm_23 processor.
```

# Target Features

```
compute10 - Use Compute Compatibility 1.0.
compute11 - Use Compute Compatibility 1.1.
compute12 - Use Compute Compatibility 1.2.
compute13 - Use Compute Compatibility 1.3.
compute20 - Use Compute Compatibility 2.0.
double    - Do not demote .f64 to .f32.
no-fma    - Disable Fused-Multiply Add.
ptx20     - Use PTX Language Version 2.0.
ptx21     - Use PTX Language Version 2.1.
ptx22     - Use PTX Language Version 2.2.
ptx23     - Use PTX Language Version 2.3.
sm10      - Use Shader Model 1.0.
sm11      - Use Shader Model 1.1.
sm12      - Use Shader Model 1.2.
sm13      - Use Shader Model 1.3.
sm20      - Use Shader Model 2.0.
sm21      - Use Shader Model 2.1.
sm22      - Use Shader Model 2.2.
sm23      - Use Shader Model 2.3.
```

# OpenCL Work-Flow

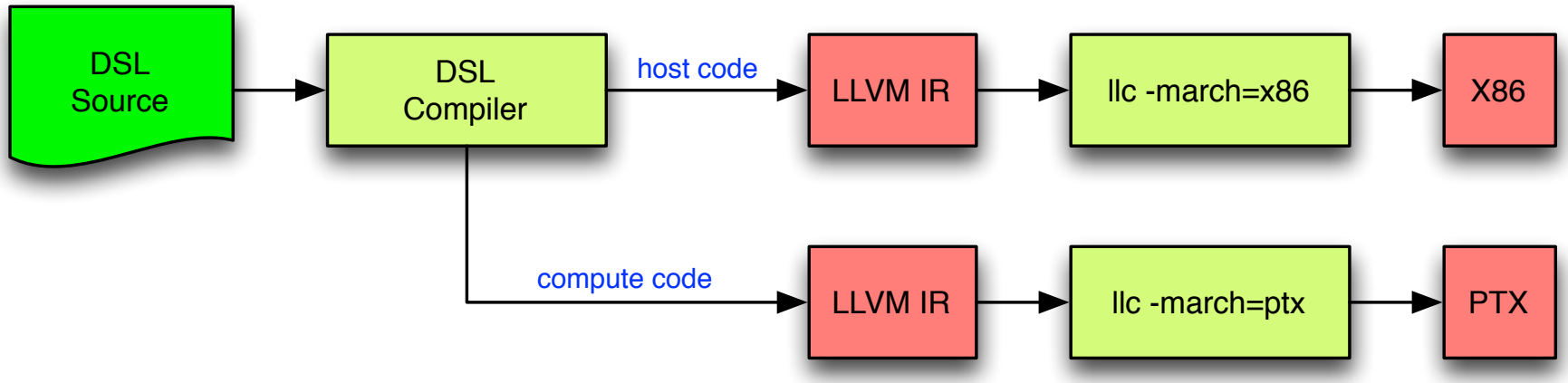



Always use `-march=ptx32` and `-mattr=+ptx23`

```
clang -ccc-host-triple ptx32 \
      -Xclang -target-feature -Xclang +ptx23 \
      -Xclang -target-feature -Xclang +sm20 \
      -I$LIBCLC/include/generic -I$LIBCLC/include/ptx \
      -include clc/clc.h -Dcl_clang_storage_class_specifiers \
      -O3 SOURCE.cl -S
```

libclc: `http://www.pcc.me.uk/~peter/libclc`

# DSL Work-Flow



# LLVM IR Considerations

- Use `ptx_kernel` and/or `ptx_device` calling conventions
  - `ptx_kernel` used for OpenCL `__kernel` and CUDA `__global__`
  - Shader model dictates calling convention details
- Kernel functions must return void
- No calls to kernel functions in LLVM IR
- No recursion if SM $<$ 2.0
- Use intrinsics to access special registers

| Intrinsic | CUDA Equiv. | PTX |
|---|---|---|
| `i32 @llvm.ptx.read.tid.x()` | `threadIdx.x` | `%tid.x` |
| `i32 @llvm.ptx.read.ntid.x()` | `blockDim.x` | `%ntid.x` |
| `i32 @llvm.ptx.read.ctaid.x()` | `blockIdx.x` | `%ctaid.x` |
| `i32 @llvm.ptx.read.nctaid.x()` | `gridDim.x` | `%nctaid.x` |
| `void @llvm.ptx.bar.sync(i32 0)` | `__syncthreads()` | `bar.sync 0;` |

# Back-End Design

## Design

- Based on LLVM SelectionDAG infrastructure
- PTX instructions and selection patterns defined in TableGen
- Use MC infrastructure for assembly printing
  - No object file support
- Address spaces used to differentiate memory spaces
  - `addrspace(0)` = global, `addrspace(4)` = shared
  - TODO: PTX unified addressing

# Back-End Design

## Design

- Based on LLVM SelectionDAG infrastructure
- PTX instructions and selection patterns defined in TableGen
- Use MC infrastructure for assembly printing
  - No object file support
- Address spaces used to differentiate memory spaces
  - `addrspace(0)` = global, `addrspace(4)` = shared
  - TODO: PTX unified addressing

## Still To-Do

- Texture/Sampler support
- Vector support
  - PTX has limited support for vectors
  - Mainly used for texture sampling
- Atomics
- Math functions in libclc

# Back-End Issues: PTX IR

PTX is an IR

- Relies on ``smart'' assembler (`ptxas`)
- PTX register allocation is not final
- Final code generation is black box (proprietary)
- Machine code format is undocumented

```
1  .reg .u64 %rd<6>;
2  .reg .f32 %f<4>;
3  ld.param.u64    %rd1, [__cudaparm__Z3fooPf_a];
4  cvt.u64.u16     %rd2, %tid.x;
5  mul.lo.u64      %rd3, %rd2, 4;
6  add.u64         %rd4, %rd1, %rd3;
7  ld.global.f32   %f1, [%rd4+0];
8  add.f32         %f2, %f1, %f1;
9  st.global.f32   [%rd4+0], %f2;
10 exit;
```

PTX

```
1  MOV R1, c [0x1] [0x100];
2  S2R R0, SR_Tid_X;
3  MOV R2, c [0x0] [0x20];
4  IMAD.U32.U32 R2.CC, R0, 0x4, R2;
5  SHR.U32.W R0, R0, 0x1e;
6  IADD.X R3, R0, c [0x0] [0x24];
7  LD.E R0, [R2];
8  FADD.FTZ R0, R0, R0;
9  ST.E [R2], R0;
10 EXIT;
```

Device

# Back-End Issues: Infinite Register File

Infinite register file

- No preset register names (except special registers)
- Declare before use

```
1 .reg .f32 %f<3>;  // Defines %f0-%f2
2 .reg .f32 %myreg; // Registers can be named anything
3
4 mul.f32 %f2, %f1, %f0;
5 add.f32 %myreg, %f2, %f2;
```

- Similar to LLVM virtual registers
- Final register allocation in `ptxas`

```
1  [jholewinski@cerberus bin]$ ptxas -arch sm_20 matmul_kernel.ptx -v
2  ptxas info    : Compiling entry function 'matmul' for 'sm_20'
3  ptxas info    : Function properties for matmul
4      0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
5  ptxas info    : Used 23 registers, 2048+0 bytes smem, 44 bytes cmem[0]
6
7  [jholewinski@cerberus bin]$ ptxas -arch sm_20 matmul_kernel.ptx -v -maxrregcount=16
8  ptxas info    : Compiling entry function 'matmul' for 'sm_20'
9  ptxas info    : Function properties for matmul
10     8 bytes stack frame, 4 bytes spill stores, 4 bytes spill loads
11 ptxas info    : Used 16 registers, 2048+0 bytes smem, 44 bytes cmem[0]
```

# Back-End Issues: Infinite Register File

Solutions:

1. Define very large register file in `PTXRegisterInfo.td`
   - Easy to implement
   - May need spill slots

# Back-End Issues: Infinite Register File

Solutions:

1. Define very large register file in `PTXRegisterInfo.td`
   - Easy to implement
   - May need spill slots
2. Define small register file and assign spill slots to new registers
   - Non-trivial to implement
   - Spills implemented as register copies
   - Lots of extraneous register copies

# Back-End Issues: Infinite Register File

Solutions:

1. Define very large register file in `PTXRegisterInfo.td`
   - Easy to implement
   - May need spill slots
2. Define small register file and assign spill slots to new registers
   - Non-trivial to implement
   - Spills implemented as register copies
   - Lots of extraneous register copies
3. Emit virtual registers (no register allocation)
   - Surprisingly easy to implement
   - Practically no register re-use
   - Leads to best post-`ptxas` register usage

# Back-End Issues: Infinite Register File

Solutions:

1. Define very large register file in `PTXRegisterInfo.td`
   - Easy to implement
   - May need spill slots
2. Define small register file and assign spill slots to new registers
   - Non-trivial to implement
   - Spills implemented as register copies
   - Lots of extraneous register copies
3. Emit virtual registers (no register allocation)
   - Surprisingly easy to implement
   - Practically no register re-use
   - Leads to best post-`ptxas` register usage

Back-end currently uses option 3

# Back-End Issues: Typed Stack Variables

Typed Stack Variables

- PTX does not have stack frames
- No %sp like with other back-ends
- Options
  - Allocate buffer in `.local` address space of appropriate size
  - Allocate a `.local` variable for each stack slot
  - Allocate a `.reg` variable for each stack slot
- Infinite register file $\Rightarrow$ no spill slots
- Still need slots for `allocas`
- Currently: Use `.local` variables for `allocas`

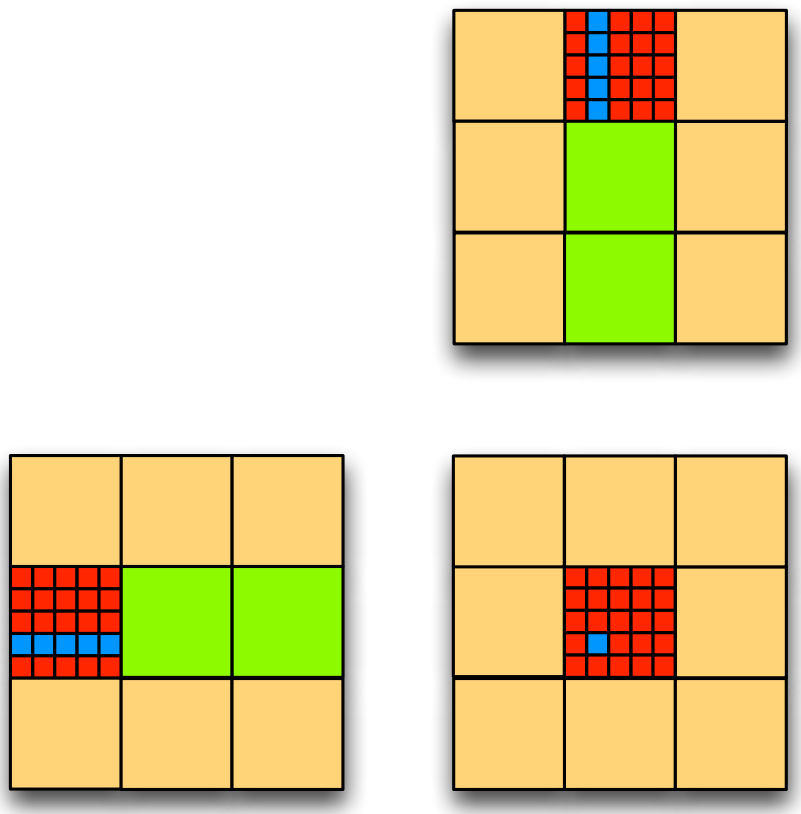
# Outline

# OpenCL Example

Matrix-Multiplication Example

- Standard example for all GPU programming courses
- Given N $\times$ N matrices A and B, compute C $=$ A $\times$ B
- Use one GPU thread for each output matrix element
- Use `__local` memory to cache matrix tiles
  - Proceeds tile-by-tile
  - Forms partial result as dot-product
  - Greatly reduces off-chip memory bandwidth requirement

# OpenCL Matrix Multiplication



```
1  __local float scratchA[BLOCK_SIZE][BLOCK_SIZE];
2  __local float scratchB[BLOCK_SIZE][BLOCK_SIZE];
3
4  for(b = 0; b < numBlocks; ++b)
5  {
6    // Populate a cache for A/B
7    int x, y;
8    x                    = b * BLOCK_SIZE + tidX;
9    y                    = globalY;
10   scratchA[tidY][tidX] = A[y * size + x];
11
12   x                    = globalX;
13   y                    = b * BLOCK_SIZE + tidY;
14   scratchB[tidY][tidX] = B[y * size + x];
15
16   barrier(CLK_LOCAL_MEM_FENCE);
17
18   for(k = 0; k < BLOCK_SIZE; ++k)
19   {
20     float myA, myB;
21     myA  = scratchA[tidY][k];
22     myB  = scratchB[k][tidX];
23     sum += myA * myB;
24   }
25
26   barrier(CLK_LOCAL_MEM_FENCE);
27 }
28 C[globalY * size + globalX] = sum;
```

# OpenCL Matrix Multiplication: LLVM IR 1

```
1  ; ModuleID = '/home/jholewinski/projects/llvm/ptx-samples/opencl/matmul/matmul_kernel.cl'
2  target datalayout = "e-p:32:32-i64:64:64-f64:64:64-n1:8:16:32:64"
3  target triple = "ptx32--"
4
5  @matmul.scratchA =
6    internal addrspace(4) unnamed_addr global [16 x [16 x float]] zeroinitializer, align 4
7  @matmul.scratchB =
8    internal addrspace(4) unnamed_addr global [16 x [16 x float]] zeroinitializer, align 4
9
10 define ptx_kernel void @matmul(float* nocapture %A,
11                               float* nocapture %B,
12                               float* nocapture %C) nounwind noinline {
13 get_local_id.exit:
14   %0 = tail call i32 @llvm.ptx.read.ctaid.x() nounwind
15   %1 = tail call i32 @llvm.ptx.read.ntid.x() nounwind
16   %mul.i = mul nsw i32 %1, %0
17   %2 = tail call i32 @llvm.ptx.read.tid.x() nounwind
18   %add.i = add nsw i32 %mul.i, %2
19   %3 = tail call i32 @llvm.ptx.read.ctaid.y() nounwind
20   %4 = tail call i32 @llvm.ptx.read.ntid.y() nounwind
21   %mul2.i = mul nsw i32 %4, %3
22   %5 = tail call i32 @llvm.ptx.read.tid.y() nounwind
23   %add3.i = add nsw i32 %mul2.i, %5
24   %6 = tail call i32 @llvm.ptx.read.nctaid.x() nounwind
25   %mul.i34 = mul nsw i32 %6, %1
26   %div = sdiv i32 %mul.i34, 16
27   %cmp39 = icmp sgt i32 %mul.i34, 15
28   %mul5 = mul nsw i32 %mul.i34, %add3.i
29   br i1 %cmp39, label %for.body.lr.ph, label %for.end27
```

# OpenCL Matrix Multiplication: LLVM IR II

```
30
31 for.body.lr.ph:                                   ; preds = %get_local_id.exit
32   %add = add i32 %mul5, %2
33   %arrayidx8 =
34     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 %2
35   %arrayidx15 =
36     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 %5, i32 %2
37   %arrayidx20 =
38     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 0
39   br label %for.body
40
41 for.body:                                         ; preds = %for.body.lr.ph, %for.body
42   %b.041 = phi i32 [ 0, %for.body.lr.ph ], [ %inc26, %for.body ]
43   %sum.040 = phi float [ 0.000000e+00, %for.body.lr.ph ], [ %add24.15, %for.body ]
44   %mul = shl nsw i32 %b.041, 4
45   %add6 = add i32 %add, %mul
46   %arrayidx = getelementptr inbounds float* %A, i32 %add6
47   %7 = load float* %arrayidx, align 4, !tbaa !1
48   store float %7, float addrspace(4)* %arrayidx8, align 4, !tbaa !1
49   %add10 = add nsw i32 %mul, %5
50   %mul11 = mul nsw i32 %add10, %mul.i34
51   %add12 = add nsw i32 %mul11, %add.i
52   %arrayidx13 = getelementptr inbounds float* %B, i32 %add12
53   %8 = load float* %arrayidx13, align 4, !tbaa !1
54   store float %8, float addrspace(4)* %arrayidx15, align 4, !tbaa !1
55   tail call void @llvm.ptx.bar.sync(i32 0) nounwind
56   %9 = load float addrspace(4)* %arrayidx20, align 4, !tbaa !1
57   %arrayidx22 =
58     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 0, i32 %2
```

# OpenCL Matrix Multiplication: LLVM IR III

```
59  %10 = load float addrspace(4)* %arrayidx22, align 4, !tbaa !1
60  %mul23 = fmul float %9, %10
61  %add24 = fadd float %sum.040, %mul23
62  %arrayidx20.1 =
63    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 1
64  %11 = load float addrspace(4)* %arrayidx20.1, align 4, !tbaa !1
65  %arrayidx22.1 =
66    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 1, i32 %2
67  %12 = load float addrspace(4)* %arrayidx22.1, align 4, !tbaa !1
68  %mul23.1 = fmul float %11, %12
69  %add24.1 = fadd float %add24, %mul23.1
70  %arrayidx20.2 =
71    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 2
72  %13 = load float addrspace(4)* %arrayidx20.2, align 4, !tbaa !1
73  %arrayidx22.2 =
74    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 2, i32 %2
75  %14 = load float addrspace(4)* %arrayidx22.2, align 4, !tbaa !1
76  %mul23.2 = fmul float %13, %14
77  %add24.2 = fadd float %add24.1, %mul23.2
78  %arrayidx20.3 =
79    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 3
80  %15 = load float addrspace(4)* %arrayidx20.3, align 4, !tbaa !1
81  %arrayidx22.3 =
82    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 3, i32 %2
83  %16 = load float addrspace(4)* %arrayidx22.3, align 4, !tbaa !1
84  %mul23.3 = fmul float %15, %16
85  %add24.3 = fadd float %add24.2, %mul23.3
86  %arrayidx20.4 =
87    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 4
```

# OpenCL Matrix Multiplication: LLVM IR IV

```
88  %17 = load float addrspace(4)* %arrayidx20.4, align 4, !tbaa !1
89  %arrayidx22.4 =
90    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 4, i32 %2
91  %18 = load float addrspace(4)* %arrayidx22.4, align 4, !tbaa !1
92  %mul23.4 = fmul float %17, %18
93  %add24.4 = fadd float %add24.3, %mul23.4
94  %arrayidx20.5 =
95    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 5
96  %19 = load float addrspace(4)* %arrayidx20.5, align 4, !tbaa !1
97  %arrayidx22.5 =
98    getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 5, i32 %2
99  %20 = load float addrspace(4)* %arrayidx22.5, align 4, !tbaa !1
100 %mul23.5 = fmul float %19, %20
101 %add24.5 = fadd float %add24.4, %mul23.5
102 %arrayidx20.6 =
103   getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 6
104 %21 = load float addrspace(4)* %arrayidx20.6, align 4, !tbaa !1
105 %arrayidx22.6 =
106   getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 6, i32 %2
107 %22 = load float addrspace(4)* %arrayidx22.6, align 4, !tbaa !1
108 %mul23.6 = fmul float %21, %22
109 %add24.6 = fadd float %add24.5, %mul23.6
110 %arrayidx20.7 =
111   getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 7
112 %23 = load float addrspace(4)* %arrayidx20.7, align 4, !tbaa !1
113 %arrayidx22.7 =
114   getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 7, i32 %2
115 %24 = load float addrspace(4)* %arrayidx22.7, align 4, !tbaa !1
116 %mul23.7 = fmul float %23, %24
```

# OpenCL Matrix Multiplication: LLVM IR V

```
117   %add24.7 = fadd float %add24.6, %mul23.7
118   %arrayidx20.8 =
119     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 8
120   %25 = load float addrspace(4)* %arrayidx20.8, align 4, !tbaa !1
121   %arrayidx22.8 =
122     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 8, i32 %2
123   %26 = load float addrspace(4)* %arrayidx22.8, align 4, !tbaa !1
124   %mul23.8 = fmul float %25, %26
125   %add24.8 = fadd float %add24.7, %mul23.8
126   %arrayidx20.9 =
127     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 9
128   %27 = load float addrspace(4)* %arrayidx20.9, align 4, !tbaa !1
129   %arrayidx22.9 =
130     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 9, i32 %2
131   %28 = load float addrspace(4)* %arrayidx22.9, align 4, !tbaa !1
132   %mul23.9 = fmul float %27, %28
133   %add24.9 = fadd float %add24.8, %mul23.9
134   %arrayidx20.10 =
135     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 10
136   %29 = load float addrspace(4)* %arrayidx20.10, align 4, !tbaa !1
137   %arrayidx22.10 =
138     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 10, i32 %2
139   %30 = load float addrspace(4)* %arrayidx22.10, align 4, !tbaa !1
140   %mul23.10 = fmul float %29, %30
141   %add24.10 = fadd float %add24.9, %mul23.10
142   %arrayidx20.11 =
143     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 11
144   %31 = load float addrspace(4)* %arrayidx20.11, align 4, !tbaa !1
145   %arrayidx22.11 =
```

# OpenCL Matrix Multiplication: LLVM IR VI

```
146     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 11, i32 %2
147   %32 = load float addrspace(4)* %arrayidx22.11, align 4, !tbaa !1
148   %mul23.11 = fmul float %31, %32
149   %add24.11 = fadd float %add24.10, %mul23.11
150   %arrayidx20.12 =
151     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 12
152   %33 = load float addrspace(4)* %arrayidx20.12, align 4, !tbaa !1
153   %arrayidx22.12 =
154     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 12, i32 %2
155   %34 = load float addrspace(4)* %arrayidx22.12, align 4, !tbaa !1
156   %mul23.12 = fmul float %33, %34
157   %add24.12 = fadd float %add24.11, %mul23.12
158   %arrayidx20.13 =
159     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 13
160   %35 = load float addrspace(4)* %arrayidx20.13, align 4, !tbaa !1
161   %arrayidx22.13 =
162     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 13, i32 %2
163   %36 = load float addrspace(4)* %arrayidx22.13, align 4, !tbaa !1
164   %mul23.13 = fmul float %35, %36
165   %add24.13 = fadd float %add24.12, %mul23.13
166   %arrayidx20.14 =
167     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 14
168   %37 = load float addrspace(4)* %arrayidx20.14, align 4, !tbaa !1
169   %arrayidx22.14 =
170     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 14, i32 %2
171   %38 = load float addrspace(4)* %arrayidx22.14, align 4, !tbaa !1
172   %mul23.14 = fmul float %37, %38
173   %add24.14 = fadd float %add24.13, %mul23.14
174   %arrayidx20.15 =
```

# OpenCL Matrix Multiplication: LLVM IR VII

```
175     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchA, i32 0, i32 %5, i32 15
176   %39 = load float addrspace(4)* %arrayidx20.15, align 4, !tbaa !1
177   %arrayidx22.15 =
178     getelementptr inbounds [16 x [16 x float]] addrspace(4)* @matmul.scratchB, i32 0, i32 15, i32 %2
179   %40 = load float addrspace(4)* %arrayidx22.15, align 4, !tbaa !1
180   %mul23.15 = fmul float %39, %40
181   %add24.15 = fadd float %add24.14, %mul23.15
182   tail call void @llvm.ptx.bar.sync(i32 0) nounwind
183   %inc26 = add nsw i32 %b.041, 1
184   %cmp = icmp slt i32 %inc26, %div
185   br i1 %cmp, label %for.body, label %for.end27
186
187 for.end27:                                        ; preds = %get_local_id.exit, %for.body
188   %sum.0.lcssa = phi float [ %add24.15, %for.body ], [ 0.000000e+00, %get_local_id.exit ]
189   %add29 = add nsw i32 %mul5, %add.i
190   %arrayidx30 = getelementptr inbounds float* %C, i32 %add29
191   store float %sum.0.lcssa, float* %arrayidx30, align 4, !tbaa !1
192   ret void
193 }
194
195 declare void @llvm.ptx.bar.sync(i32) nounwind
196
197 declare i32 @llvm.ptx.read.tid.x() nounwind readnone
198
199 declare i32 @llvm.ptx.read.tid.y() nounwind readnone
200
201 declare i32 @llvm.ptx.read.nctaid.x() nounwind readnone
202
203 declare i32 @llvm.ptx.read.ntid.x() nounwind readnone
```

# OpenCL Matrix Multiplication: LLVM IR VIII

```
204
205 declare i32 @llvm.ptx.read.ntid.y() nounwind readnone
206
207 declare i32 @llvm.ptx.read.ctaid.x() nounwind readnone
208
209 declare i32 @llvm.ptx.read.ctaid.y() nounwind readnone
210
211 !opencl.kernels = !{!0}
212
213 !0 = metadata !{void (float*, float*, float*)* @matmul}
214 !1 = metadata !{metadata !"float", metadata !2}
215 !2 = metadata !{metadata !"omnipotent char", metadata !3}
216 !3 = metadata !{metadata !"Simple C/C++ TBAA", null}
```

# OpenCL Matrix Multiplication: PTX I

```
1         .reg .pred %p<2>;
2         .reg .b32 %r<60>;
3         .reg .f32 %f<56>;
4 // BB#0:                                 // %get_local_id.exit
5         mov.u32         %r7, %nctaid.x;
6         mov.u32         %r1, %ntid.x;
7         mul.lo.u32         %r22, %r7, %r1;
8         mov.u32         %r5, %ntid.y;
9         mov.u32         %r4, %ctaid.y;
10        mov.u32         %r0, %ctaid.x;
11        mov.u32         %r6, %tid.y;
12        mul.lo.u32         %r25, %r5, %r4;
13        mov.u32         %r2, %tid.x;
14        ld.param.u32         %r19, [__param_1];
15        ld.param.u32         %r20, [__param_2];
16        ld.param.u32         %r21, [__param_3];
17        mov.f32         %f3, 0D0000000000000000;
18        setp.lt.s32         %p0, %r22, 16;
19        mov.f32         %f55, %f3;
20 @%p0         bra         $L__BB0_3;
21 // BB#1:                                 // %for.body.lr.ph
22        shr.s32         %r23, %r22, 31;
23        shr.u32         %r24, %r23, 28;
24        add.u32         %r27, %r22, %r24;
25        shr.s32         %r8, %r27, 4;
26        mul.lo.u32         %r31, %r7, %r6;
27        add.u32         %r32, %r0, %r31;
28        add.u32         %r34, %r6, %r25;
29        mul.lo.u32         %r35, %r1, %r32;
```

# OpenCL Matrix Multiplication: PTX II

```
30          mul.lo.u32          %r36, %r22, %r34;
31          add.u32         %r37, %r2, %r35;
32          add.u32         %r38, %r2, %r36;
33          mov.u32         %r39, matmul_2E_scratchA;
34          shl.b32         %r40, %r6, 6;
35          mov.u32         %r41, matmul_2E_scratchB;
36          add.u32         %r42, %r39, %r40;
37          shl.b32         %r43, %r2, 2;
38          add.u32         %r44, %r41, %r40;
39          shl.b32         %r45, %r37, 2;
40          shl.b32         %r46, %r38, 2;
41          add.u32         %r10, %r42, %r43;
42          add.u32         %r11, %r44, %r43;
43          add.u32         %r12, %r20, %r45;
44          shl.b32         %r13, %r22, 6;
45          add.u32         %r14, %r19, %r46;
46          mov.u32         %r29, 0;
47          mov.f32         %f4, 0D0000000000000000;
48          mov.u32         %r58, %r12;
49          mov.u32         %r59, %r29;
50          mov.f32         %f54, %f4;
51  $L__BB0_2:                              // %for.body
52                                          // =>This Inner Loop Header: Depth=1
53          mov.u32         %r15, %r58;
54          mov.u32         %r16, %r59;
55          mov.f32         %f0, %f54;
56          shl.b32         %r47, %r16, 6;
57          add.u32         %r48, %r14, %r47;
58          ld.global.f32       %f5, [%r48];
```

# OpenCL Matrix Multiplication: PTX III

```
59        st.shared.f32       [%r10], %f5;
60        ld.global.f32       %f6, [%r15];
61        st.shared.f32       [%r11], %f6;
62        bar.sync        0;
63        add.u32       %r54, %r41, %r43;
64        ld.shared.f32       %f7, [%r42];
65        ld.shared.f32       %f8, [%r54];
66        mad.f32       %f9, %f7, %f8, %f0;
67        ld.shared.f32       %f10, [%r42+4];
68        ld.shared.f32       %f11, [%r54+64];
69        mad.f32       %f12, %f10, %f11, %f9;
70        ld.shared.f32       %f13, [%r42+8];
71        ld.shared.f32       %f14, [%r54+128];
72        mad.f32       %f15, %f13, %f14, %f12;
73        ld.shared.f32       %f16, [%r42+12];
74        ld.shared.f32       %f17, [%r54+192];
75        mad.f32       %f18, %f16, %f17, %f15;
76        ld.shared.f32       %f19, [%r42+16];
77        ld.shared.f32       %f20, [%r54+256];
78        mad.f32       %f21, %f19, %f20, %f18;
79        ld.shared.f32       %f22, [%r42+20];
80        ld.shared.f32       %f23, [%r54+320];
81        mad.f32       %f24, %f22, %f23, %f21;
82        ld.shared.f32       %f25, [%r42+24];
83        ld.shared.f32       %f26, [%r54+384];
84        mad.f32       %f27, %f25, %f26, %f24;
85        ld.shared.f32       %f28, [%r42+28];
86        ld.shared.f32       %f29, [%r54+448];
87        mad.f32       %f30, %f28, %f29, %f27;
```

# OpenCL Matrix Multiplication: PTX IV

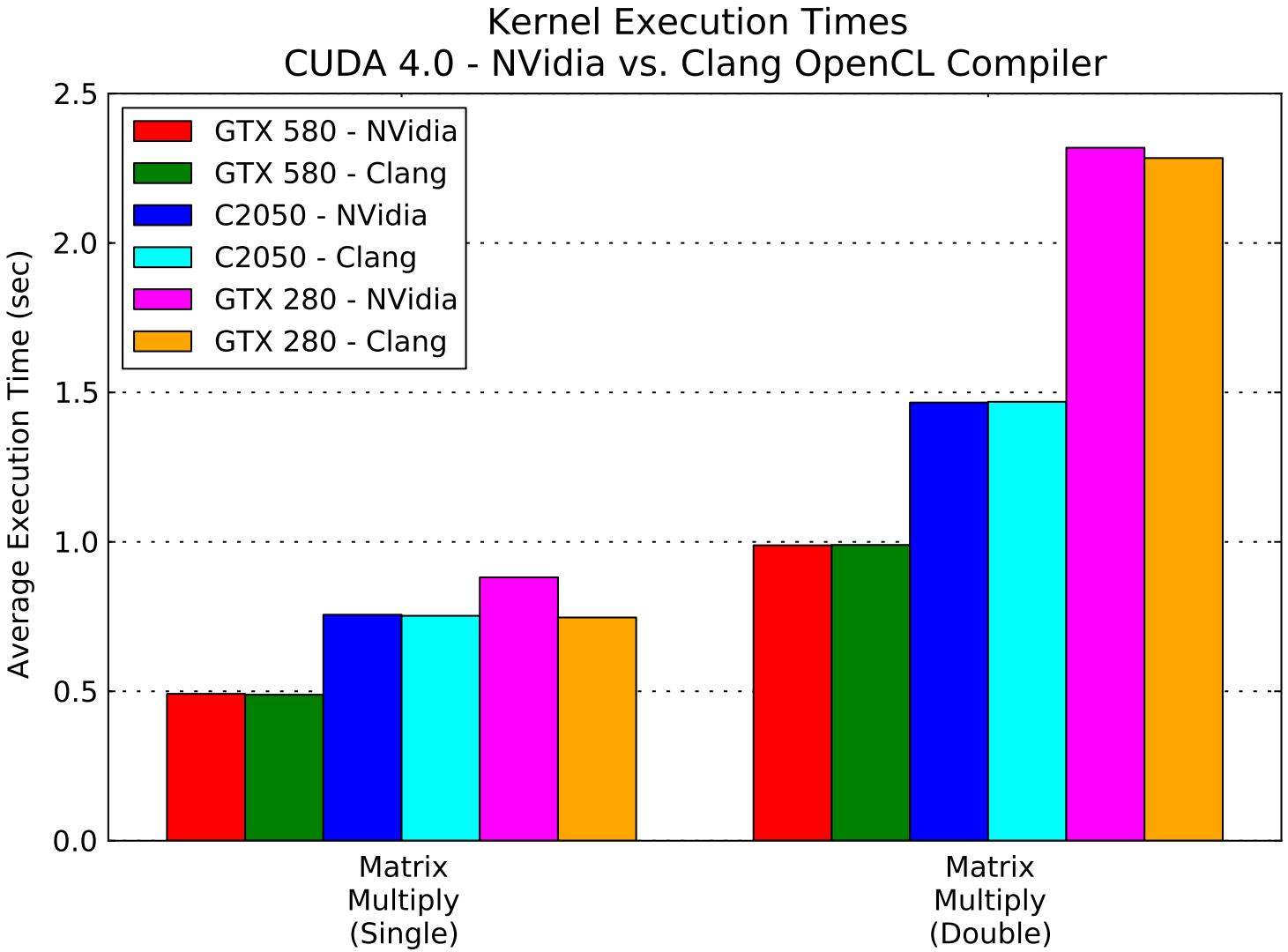
```
88        ld.shared.f32         %f31, [%r42+32];
89        ld.shared.f32         %f32, [%r54+512];
90        mad.f32       %f33, %f31, %f32, %f30;
91        ld.shared.f32         %f34, [%r42+36];
92        ld.shared.f32         %f35, [%r54+576];
93        mad.f32       %f36, %f34, %f35, %f33;
94        ld.shared.f32         %f37, [%r42+40];
95        ld.shared.f32         %f38, [%r54+640];
96        mad.f32       %f39, %f37, %f38, %f36;
97        ld.shared.f32         %f40, [%r42+44];
98        ld.shared.f32         %f41, [%r54+704];
99        mad.f32       %f42, %f40, %f41, %f39;
100       ld.shared.f32         %f43, [%r42+48];
101       ld.shared.f32         %f44, [%r54+768];
102       mad.f32       %f45, %f43, %f44, %f42;
103       ld.shared.f32         %f46, [%r42+52];
104       ld.shared.f32         %f47, [%r54+832];
105       mad.f32       %f48, %f46, %f47, %f45;
106       ld.shared.f32         %f49, [%r42+56];
107       ld.shared.f32         %f50, [%r54+896];
108       mad.f32       %f51, %f49, %f50, %f48;
109       ld.shared.f32         %f52, [%r42+60];
110       ld.shared.f32         %f53, [%r54+960];
111       mad.f32       %f1, %f52, %f53, %f51;
112       add.u32       %r17, %r16, 1;
113       add.u32       %r18, %r15, %r13;
114       setp.lt.s32       %p1, %r17, %r8;
115       bar.sync      0;
116       mov.u32       %r58, %r18;
```

# OpenCL Matrix Multiplication: PTX V

```
117         mov.u32         %r59, %r17;
118         mov.f32         %f54, %f1;
119         mov.f32         %f55, %f1;
120 @%p1        bra         $L__BB0_2;
121 $L__BB0_3:                                  // %for.end27
122         mov.f32         %f2, %f55;
123         mul.lo.u32          %r26, %r1, %r0;
124         add.u32         %r28, %r25, %r6;
125         add.u32         %r3, %r26, %r2;
126         mul.lo.u32          %r9, %r22, %r28;
127         add.u32         %r55, %r9, %r3;
128         shl.b32         %r56, %r55, 2;
129         add.u32         %r57, %r21, %r56;
130         st.global.f32           [%r57], %f2;
131         exit;
```

```
1  [jholewinski@cerberus bin]$ ptxas -arch sm_20 matmul_kernel.ptx -v
2  ptxas info    : Compiling entry function 'matmul' for 'sm_20'
3  ptxas info    : Function properties for matmul
4      0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
5  ptxas info    : Used 23 registers, 2048+0 bytes smem, 44 bytes cmem[0]
6
7  [jholewinski@cerberus bin]$ ptxas -arch sm_20 matmul_kernel.ptx -v -maxrregcount=16
8  ptxas info    : Compiling entry function 'matmul' for 'sm_20'
9  ptxas info    : Function properties for matmul
10     8 bytes stack frame, 4 bytes spill stores, 4 bytes spill loads
11 ptxas info    : Used 16 registers, 2048+0 bytes smem, 44 bytes cmem[0]
```

# Performance



# Outline

# Conclusion

- PTX back-end is being actively developed
- LLVM 3.0 marks the first usable release
- Already competitive with nVidia OpenCL SDK
- Great platform for GPU compiler research
- Collection of samples:
  `https://github.com/jholewinski/llvm-ptx-samples`
- Feel free to submit bugs against the back-end

Special Thanks:
Che-Liang Chiou
Dan Bailey

# Conclusion

Questions?