Google

# Optimizing gpucc (Part 1)

**Jingyue Wu**, Artem Belevich, Eli Bendersky, Mark Heffernan, Chris Leary, Jacques Pienaar, Bjarke Roune, Rob Springer, Xuetian Weng, Robert Hundt

# Motivation: CPU vs GPU Characteristics

CPU

- Designed for general purposes
- Optimized for latency
- Heavyweight hardware threads
  - Branch prediction
  - Out-of-order execution
  - Superscalar
- Small number of cores per die

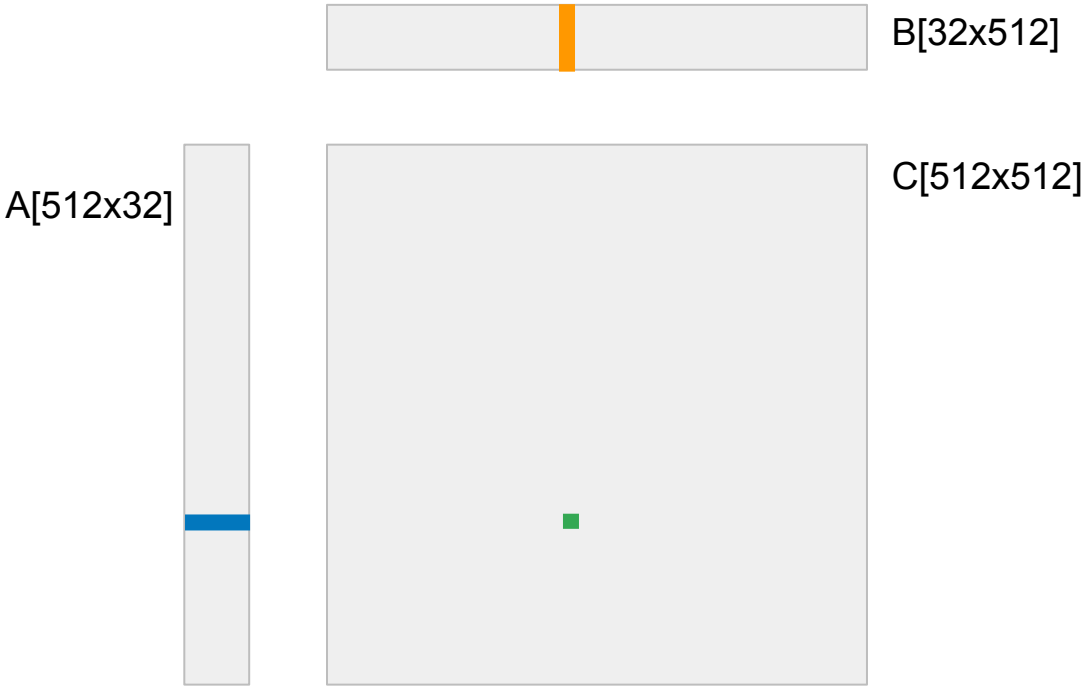
GPU

- Designed for rendering
- Optimized for throughput
- Lightweight hardware threads
- Massive parallelism
  - Can trade latency for throughput

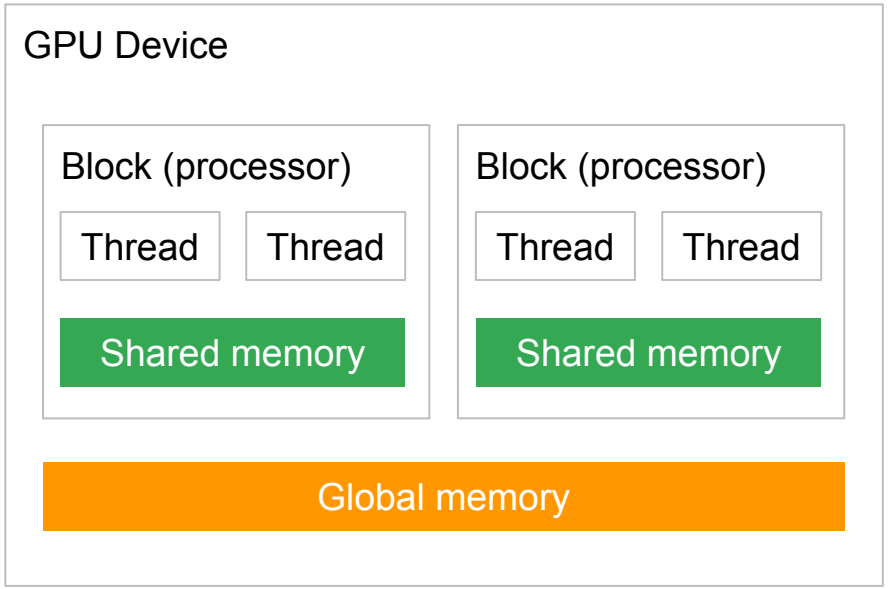
# Debug gpucc Performance

- Profiling metrics (`nvprof --metrics`)
  - Instructions executed: load/store, integer, floating point, control flow, etc.
  - Instruction stalls: execution dependency, memory dependency, etc.
  - Occupancy
- Intermediate files and machine code (`-save-temps`)
  - IR
  - PTX
  - SASS using ptxas and nvdisasm

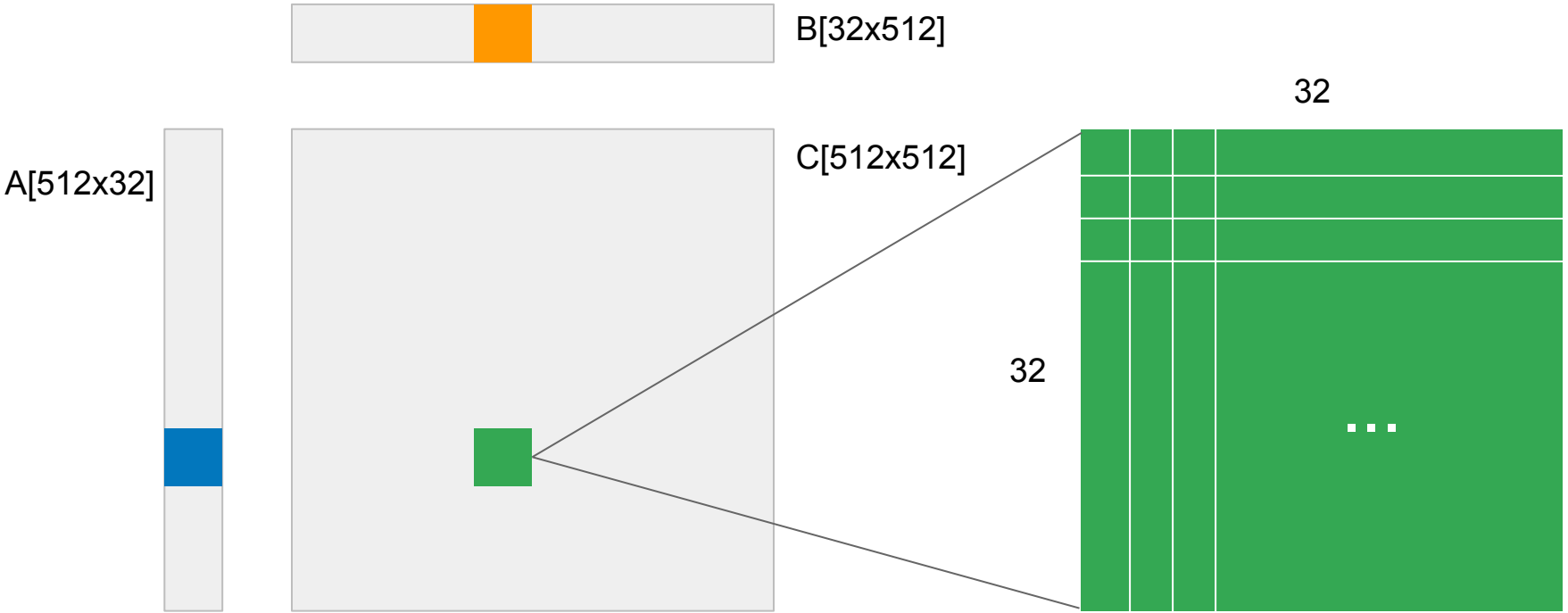
# Example: Matrix Multiply



# Shared Memory on GPU



# Matrix Multiply Using Shared Memory



# Matrix Multiply Using Shared Memory

```
__global__ void SharedMultiply(float* a, float* b, float* c) {
  __shared__ float a_tile[kWidth][kWidth], b_tile[kWidth][kWidth];
  int row = blockIdx.y * blockDim.y + threadIdx.y;
  int col = blockIdx.x * blockDim.x + threadIdx.x;
  float sum = 0.0f;
  a_tile[threadIdx.y][threadIdx.x] = a[row * kWidth + threadIdx.x];
  b_tile[threadIdx.y][threadIdx.x] = b[threadIdx.y * kMatrixSize + col];
  __syncthreads();
#pragma unroll 1
  for (int i = 0; i < kWidth; i++) {
    sum += a_tile[threadIdx.y][i] * b_tile[i][threadIdx.x];
  }
  c[row * kMatrixSize + col] = sum;
}
```

Copied and slightly modified from http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/

Full source code can be found at https://gist.github.com/wujingyue/72815bce202841753cbf

# Performance Comparison with nvcc

```
$ nvcc matmul.cu -o matmul-nvcc
$ nvprof ./matmul-nvcc

67.106us  SharedMultiply(float*, float*, float*)

$ clang++ matmul.cu -o matmul-gpucc
$ nvprof ./matmul-gpucc

91.042us  SharedMultiply(float*, float*, float*)
```

# Profiling Comparison with nvcc

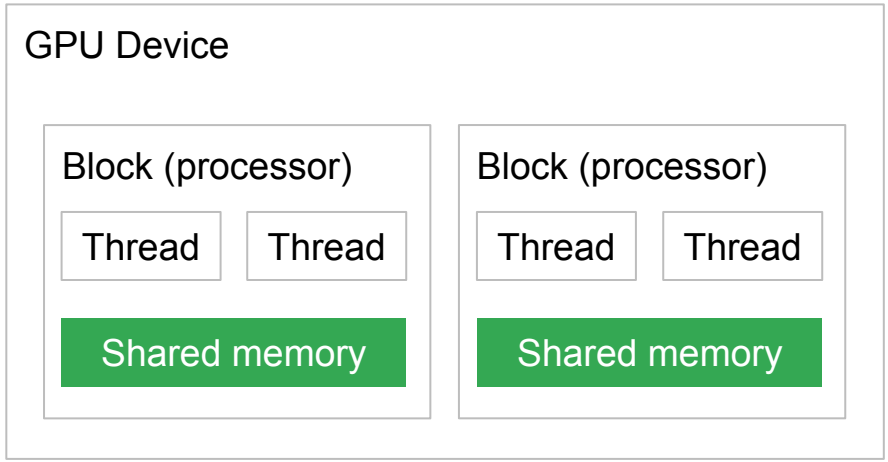
```
$ nvprof --metrics ldst_executed,inst_integer,inst_fp_32,inst_control ./matmul-nvcc

ldst_executed   Executed Load/Store Instructions     565248
inst_integer    Integer Instructions               37486592
inst_fp_32      FP Instructions(Single)             8388608
inst_control    Control-Flow Instructions          8388608

$ nvprof --metrics ldst_executed,inst_integer,inst_fp_32,inst_control ./matmul-gpucc

ldst_executed   Executed Load/Store Instructions     565248
inst_integer    Integer Instructions               82051072
inst_fp_32      FP Instructions(Single)             8388608
inst_control    Control-Flow Instructions          8388608
```

# PTX Assembly

```
// matmul-nvcc.ptx

BB0_1:
      .pragma "nounroll";
      ld.shared.f32    %f6, [%rd32];
      ld.shared.f32    %f7, [%rd31];
      fma.rn.f32       %f8, %f7, %f6, %f8;
      add.s64    %rd32, %rd32, 128;
      add.s64    %rd31, %rd31, 4;
      add.s32    %r18, %r18, 1;
      setp.ne.s32      %p1, %r18, 0;
      @%p1 bra   BB0_1;
```

```
// matmul-gpucc.ptx

LBB1_1:
      .pragma "nounroll";
      add.s64    %rd28, %rd2, %rd35;
      cvta.shared.u64 %rd29, %rd28;
      ld.f32     %f4, [%rd29];
      cvta.shared.u64 %rd30, %rd36;
      ld.f32     %f5, [%rd30];
      fma.rn.f32       %f6, %f4, %f5, %f6;
      add.s64    %rd36, %rd36, 128;
      add.s64    %rd35, %rd35, 4;
      cvt.u32.u64      %r15, %rd35;
      setp.eq.s32      %p1, %r15, 128;
      @%p1 bra   LBB1_2;
      bra.uni    LBB1_1;
```

# Specific vs Generic Memory Access




- Specific
  - `ld.shared/st.shared`
- Generic
  - `ld/st`
  - Overhead in checking
  - Alias analysis suffers

# PTX Assembly

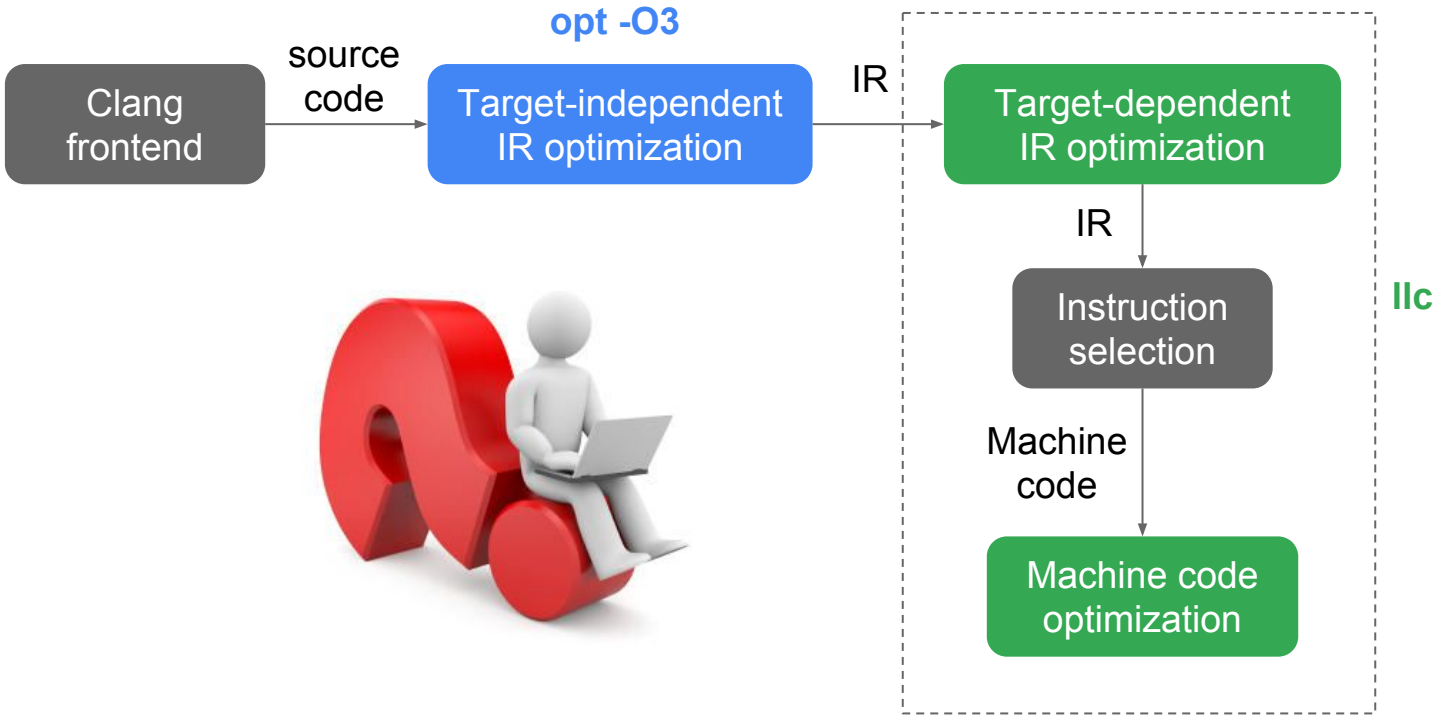
```
// matmul-nvcc.ptx

BB0_1:
      .pragma "nounroll";
      ld.shared.f32    %f6, [%rd32];
      ld.shared.f32    %f7, [%rd31];
      fma.rn.f32       %f8, %f7, %f6, %f8;
      add.s64    %rd32, %rd32, 128;
      add.s64    %rd31, %rd31, 4;
      add.s32    %r18, %r18, 1;
      setp.ne.s32      %p1, %r18, 0;
      @%p1 bra   BB0_1;
```

```
// matmul-gpucc.ptx

LBB1_1:
      .pragma "nounroll";
      add.s64    %rd28, %rd2, %rd35;
      cvta.shared.u64 %rd29, %rd28;
      ld.f32     %f4, [%rd29];
      cvta.shared.u64 %rd30, %rd36;
      ld.f32     %f5, [%rd30];
      fma.rn.f32       %f6, %f4, %f5, %f6;
      add.s64    %rd36, %rd36, 128;
      add.s64    %rd35, %rd35, 4;
      cvt.u32.u64      %r15, %rd35;
      setp.eq.s32      %p1, %r15, 128;
      @%p1 bra   LBB1_2;
      bra.uni    LBB1_1;
```

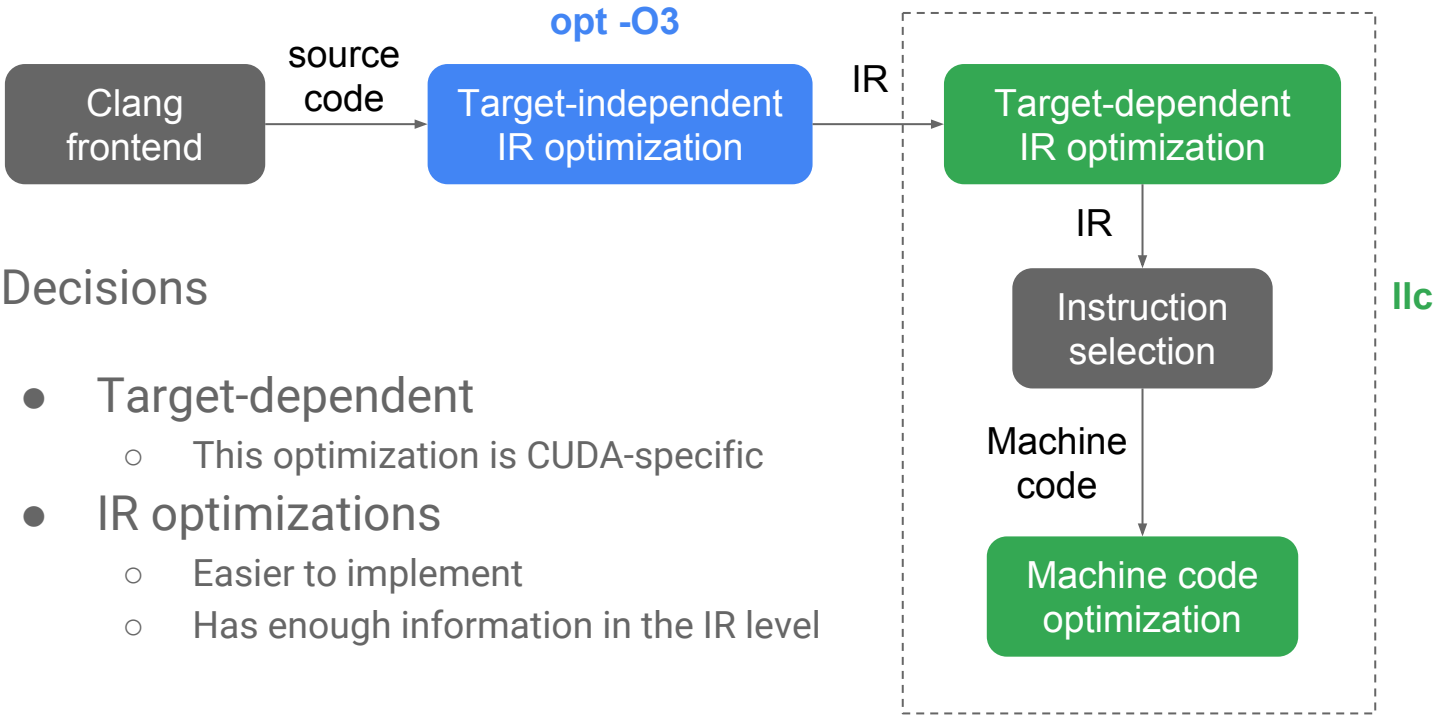
# LLVM IR for a_tile[threadIdx.y][i]

```
@_ZZ14SharedMultiplyPfS_S_E6a_tile = internal unnamed_addr addrspace(3) global [32 x
[32 x float]] zeroinitializer, align 4

// float* p = &a_tile[threadIdx.y][i]
%arrayidx32 = getelementptr inbounds [32 x [32 x float]],
    addrspacecast (
        [32 x [32 x float]] addrspace(3)* @_ZZ14SharedMultiplyPfS_S_E6a_tile
        to [32 x [32 x float]]*),
    i64 0, i64 %idxprom14, i64 %idxprom28
// *p
%14 = load float, float* %arrayidx32, align 4, !tbaa !7
```

# gpucc Optimization Pipeline



# gpucc Optimization Pipeline




Decisions

- Target-dependent
  - This optimization is CUDA-specific
- IR optimizations
  - Easier to implement
  - Has enough information in the IR level

# Implement Address Space Inference

```
// lib/Target/NVPTX/NVPTXInferAddressSpaces.cpp
class NVPTXInferAddressSpaces: public FunctionPass {
public:
  static char ID;
  NVPTXInferAddressSpaces() : FunctionPass(ID) {}
  bool runOnFunction(Function &F) override;
};

INITIALIZE_PASS(NVPTXInferAddressSpaces,
                "nvptx-infer-addrspace",
                "Infer address spaces", false, false)

FunctionPass *llvm::createNVPTXInferAddressSpacesPass()
{
  return new NVPTXInferAddressSpaces();
}
```

```
// lib/Target/NVPTX/NVPTXTargetMachine.cpp
extern "C" void LLVMInitializeNVPTXTarget()
{
  ...
  initializeNVPTXInferAddressSpacesPass(PR);
  ...
}

void NVPTXPassConfig::addIRPasses() {
  ...
  addPass(
      createNVPTXInferAddressSpacesPass());
  ...
}
```

- http://llvm.org/docs/WritingAnLLVMPass.html
- Learn from other passes under lib/Transforms/

# Performance Comparison with nvcc Again

```
$ nvcc matmul.cu -o matmul-nvcc
$ nvprof ./matmul-nvcc

67.106us  SharedMultiply(float*, float*, float*)

$ clang++ matmul.cu -o matmul-gpucc
$ nvprof ./matmul-gpucc

68.273us  SharedMultiply(float*, float*, float*)
```

# Profiling Comparison with nvcc

```
$ nvprof --metrics ldst_executed,inst_integer,inst_fp_32,inst_control ./matmul-nvcc

ldst_executed   Executed Load/Store Instructions     565248
inst_integer    Integer Instructions               37486592
inst_fp_32      FP Instructions(Single)             8388608
inst_control    Control-Flow Instructions           8388608

$ nvprof --metrics ldst_executed,inst_integer,inst_fp_32,inst_control ./matmul-gpucc

ldst_executed   Executed Load/Store Instructions     565248
inst_integer    Integer Instructions               37486592
inst_fp_32      FP Instructions(Single)             8388608
inst_control    Control-Flow Instructions           8388608
```

# Summary of This Session

- Why GPU-specific optimizations
- How to debug gpucc performance
- How to add new optimizations
- More optimizations
  - Straight-line strength reduction
  - Bypassing 64-bit divides
  - Speculative execution
  - ...
  - http://bit.ly/llvm-cuda and tomorrow's talk (Session 3: GPU)