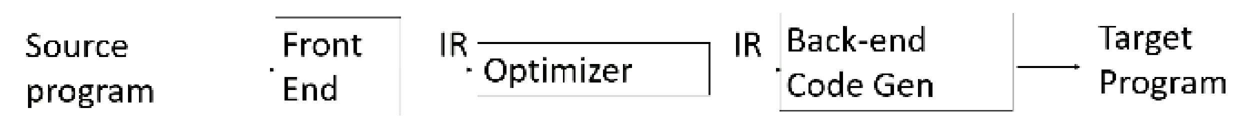
Compiler Background 1

GPU Compiler Flow

Learning Objectives

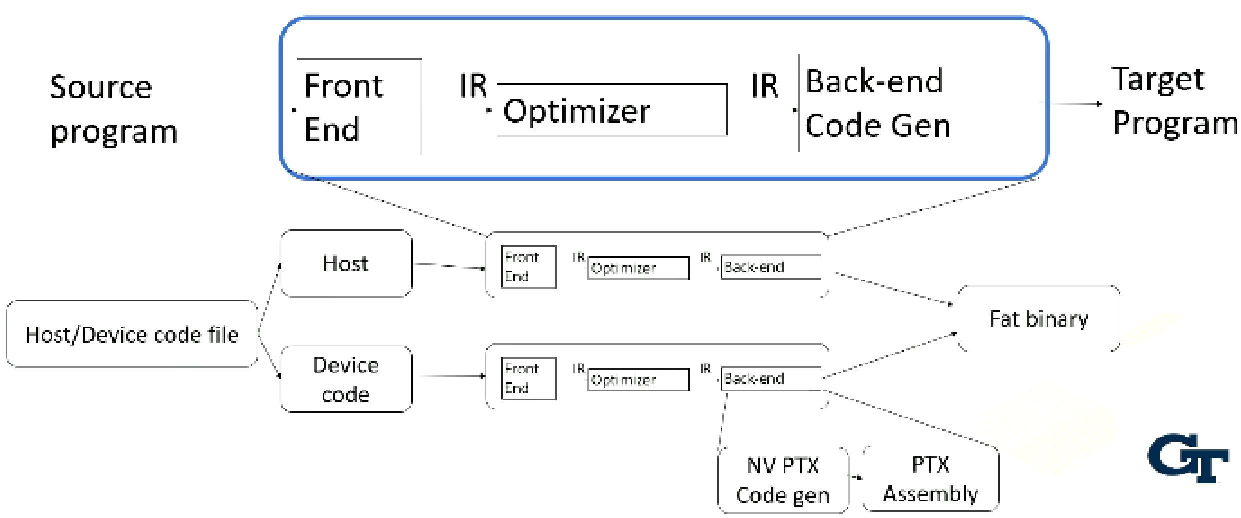
1. Demonstrate comprehension of the fundamental process of GPU program compilation
2. Explore the components and stages involved in GPU compilation

Compiler Flow



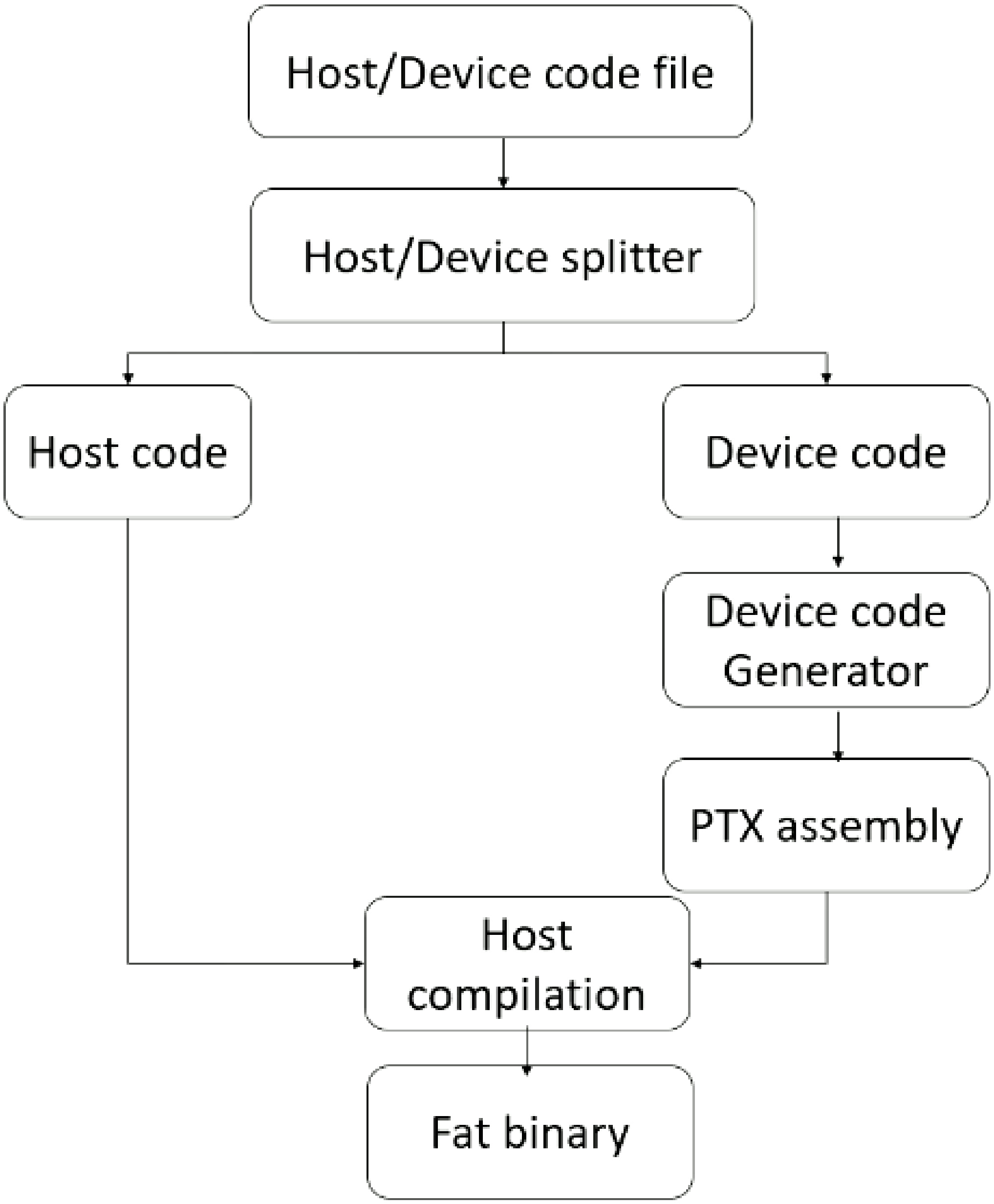
Compiler Flow

GPU Compiler Flow



GPU Compiler Flow

GPU Compiler Pass (Open GPU Compiler)



GPU Compiler Pass

Roles of CLANG

1. Front end parser
2. Tool chain for C-family languages
3. Generating the Abstract Syntax Tree (AST)

C++ PreProcessor

1. Performs text substitution before compilation

IR Optimizations

1. Intermediate representation
2. Back-end compiler
3. IR provides a good abstract to optimize
4. Many compiler optimizations are done in the IR level

PTX vs SASS

1. PTX
    - Parallel Thread Execution
    - PTX is a virtual ISA
    - Architecture independent
    - PTX will be translated to machine code
    - PTX does not have register allocation
2. SASS
    - Low-level assembly language
    - Shader assembly
    - Architecture dependent assembly code
    - Register is allocated

Fat Binaries

1. Contain execution files for multiple architectures
2. Supports multiple GPU versions
3. Also includes CPU code

Summary

1. Recap the terminology of PTX, SASS, CLANG, IR, Fat binary
2. Review the overall compilation process for GPU programins, including its key stages and components

PTX

Learning Objectives

1. Explore the basic of PTX
2. Explore PTX instruction format
3. Describe optional predicate information
4. Understand PTX code examples

PTX Instruction

1. Zero to four operands
2. Optional predicate information following an @ symbol
3. @p opcode d, a, b, c; // d: destination. a,b,c: source operands
4. setp: Writes to destination register
    - Use "|" to separate multiple destination registers
    - setp.lt.s32 p|q, a, b; // p = (a < b); q = !(a < b);

Predicated Execution

1. Predicated registers can be declared as
    - .reg .pred p, q, r;
2. Predicated registers are virtual and declared with .pred type specifier
3. Predicate variables are optional
4. lt: Less than

```
if (i < n)
    j = j + 1;

setp.lt.s32 p, i, n; // p = (i < n);
@p add.s32 j, j, 1; // if i < n, add 1 to j
```

Example of PTX Code

1. A PTX statement is either a directive or an instruction
2. Example of directive: target .address_size, .function etc.
3. Statements begin with an optional label and end with a semicolon

```
.reg    .b32 r1, r2;
.global .f32 array[N];

start:  mov.b32 r1, %tid.x;
        shl.b32 r1, r1, 2;           // shift thread id by 2 bits
        ld.global.b32 r2, array[r1]; // thread[tid] gets array[tid
        add.f32 r2, r2, 0.5;         // add 1/2
```

Other PTX Instruction Examples

1. Control flow instructions
   - bra targ1;
     - Branch to target label 'targ1'
   - all func;
     - Call function 'func'
   - ret;
     - Return from function call
2. Synchronization instructions
   - membar, fence
3. Atomic instructions
   - Atom prefix
   - Example: atom.add.f16
4. Special PTX Registers
   - ntid: Number of threads in a CTA (Cooperative Thread Array)
   - tid: Thread ID
   - sp: Stack pointer

Summary

1. Reviewed PTX instructions
2. Emphasized the significance of predicated execution and its optional predicate variables
3. Reviewed various examples of PTX instructions, including control flow, synchronization, and atomic instructions

IR and Basic Block

Learning Objectives

1. Describe intermediate representation (IR)
2. Identify basic blocks within code
3. Construct a control flow graph

IR

1. Intemdiate Representation

2. Typical IR uses three-address code
    - A -> B op C
3. LLVM IR version: %result = add i32 %a, %b
    - %result: destination register (target variable)
    - add: operation
    - i32: results are 32-bit integer
    - %a, %b: source operands
    - PTX version: add.u32 %r1, %r2, %r3 or add.s32 %r1, %r2, %r3
        - add.u32 for unsigned, add.s32 for signed

Basic Block

1. A maximum sequence of instruction stream with one entry and one exit
    - Only the first instruction can be reached from outside
    - Once the program enters a basic block, all instructions inside the basic block need to be executed
    - All execution needs to be consecutive
    - Exit instruction is typically a control-flow instruction
2. Optimizations within a basic block are local code optimizations

Flow Graph

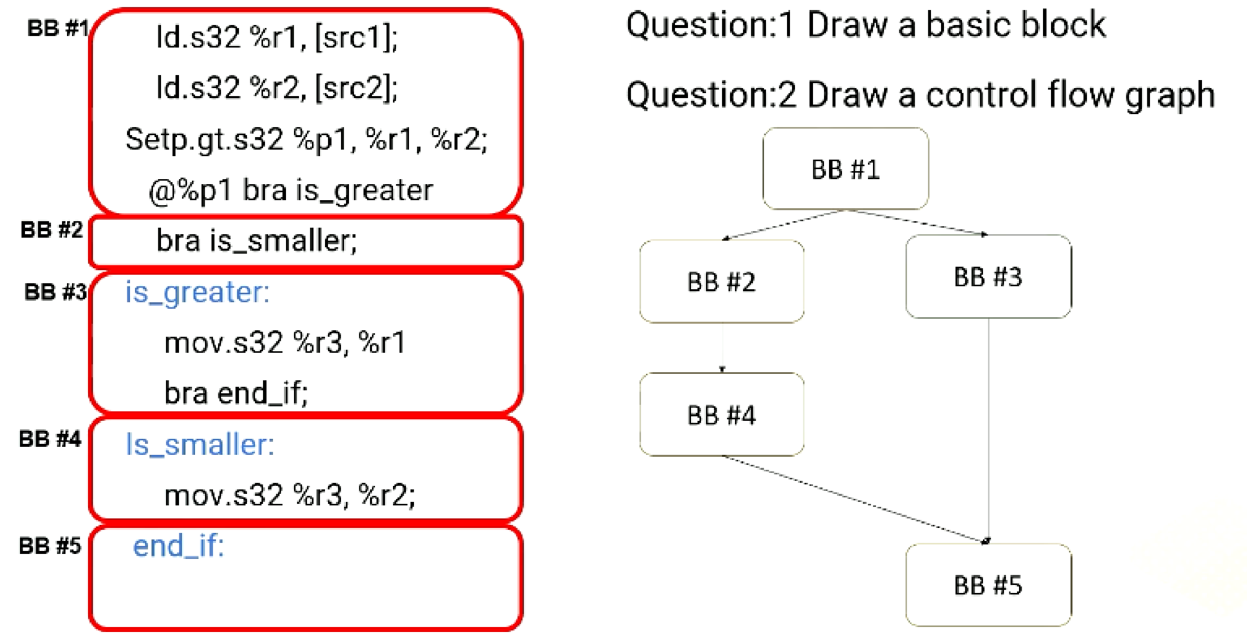
1. Flow graph: Each node represents a basic block, and path indicates possible program execution path
2. Entry node: The first statement of the program

Control Flow Graph

Algorithm to Find Basic Blocks

1. Identify a leader
    - Leader: The first instruction in a program
    - Any instruction that is the target of a conditional or unconditional jump
    - Any instruction that immediately follows a conditional or unconditional jump
2. Group instructions from a leader to the next leader. The group becomes a basic block

Example of if-else PTX Code



PTX Code Example

Summary

1. Covered intermediate representation (IR) and its significance
2. Explored techniques for identifying and defining basic blocks
3. Demonstrated how to construct control flow graphs

Introduction to Data Flow Analysis

Learning Objectives

1. Explain global code optimization
2. Understand example code optimizations
3. Explain the basic concept of data flow analysis
4. Explain the concept of reach definitions

Global Code Optimizations

1. Local code optimization: Optimization within a basic block
2. Global code optimization: Optimization across basic blocks
3. Most global code optimization is based on data-flow analyses
4. Data-flow analysis:
   - Analyze the effect of each basic block
   - Analyses differ by examining properties
5. Principal sources of optimization
   - Compiler optimization must preserve the semantics of the original program

Examples of Code Optimizations

1. Removing redundant instructions
2. Copy propagation
3. Dead code eliminations
4. Code motion

5. Induction variable detection
6. Reduction strength

Data-Flow Analysis Abstraction

1. Execution of a program: Transformations of the program state
2. Input state: Program point before the statement
3. Output state: Program point after the statement

Transfer Functions

1. Use Transfer Functions notation
2. OUT[B] = fb(IN[B])
3. IN[B]: Immediate before a basic block
4. OUT[B]: Immediate after a basic block
   - fs: Transfer function of statement s
   - fb: fsn * ... * fs2 * fs1
   - IN[B] = Union(OUT[P]) where p is a predecessor of B
   - OUT[B] = Union(IN[S]) where s is a successor of B
   - Predecessor of B: All blocks that are executed before the basic block B
   - Successor of B: All blocks that are executed after the basic block B

Reaching Definitions

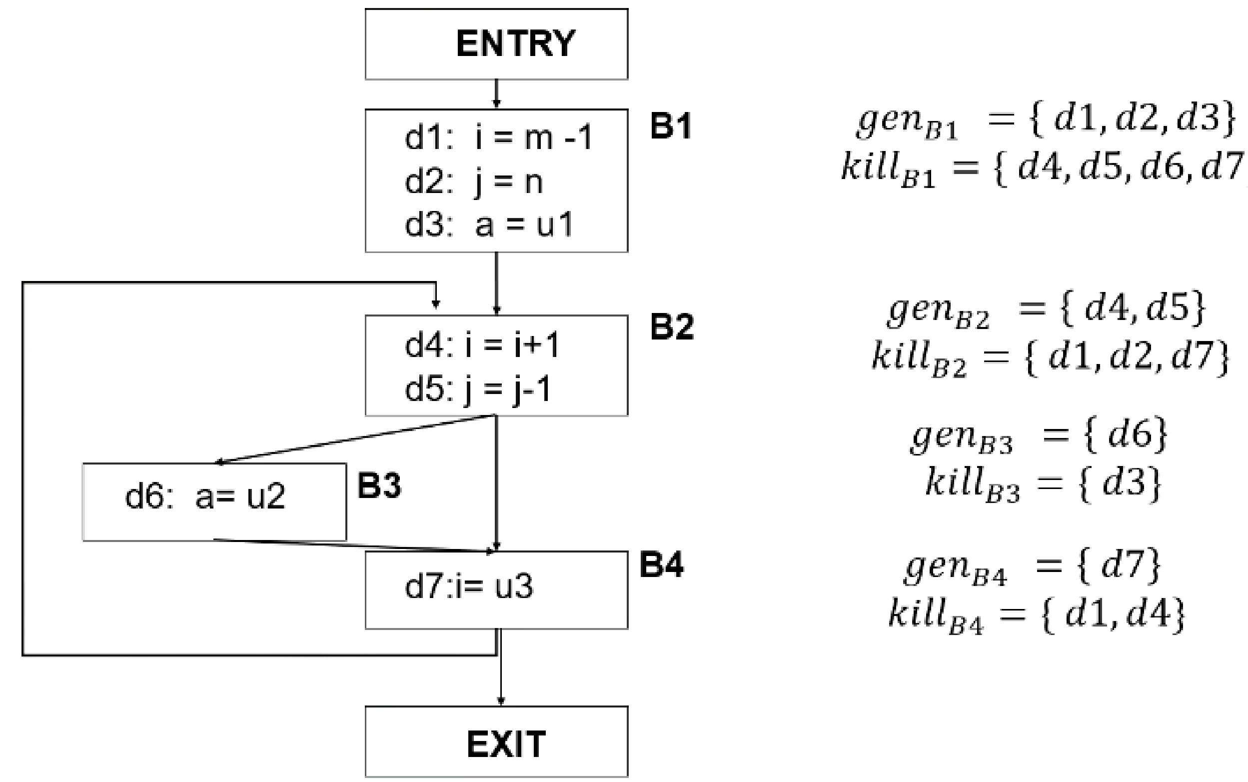
1. Analyze whether a definition reaches
2. A definition d reaches a point p if there is a path from the point immediately following d to p, without begin killed (overwritten)
3. Definitions: a variable is defined when it receives a value
4. Use; when its value is read
   - a = x + y -> definition: a, use: x, y

Gen and Kill

1. d: u = v + w

- Generates the definition d of variable u and kills all other definitions in the program that define u
  - fd(x) = gend U (x - killd)
    - gend = {d}: The set of definitions generated by the statement
    - killk = the set of all other definitions of u in the program

Example Gen and Kill Sets



Example of Gen and Kill Sets

Generalized Transfer Functions

$$f_d(x) = gen_d \cup (x - kill_d)$$

$$f_1(x) = gen_1 \cup (x - kill_1)$$

$$f_2(x) = gen_2 \cup (x - kill_2)$$

$$\begin{aligned} f_2(f_1(x)) &= gen_2 \cup (gen_1 \cup (x - kill_1) - kill_2) \\ &= (gen_2 \cup (gen_1 - kill_2)) \cup (x - (kill_1 \cup - kill_2)) \end{aligned}$$

$$f_B(x) = gen_B \cup (x - kill_B)$$

$$kill_B = kill_1 \cup kill_2 \cup \ldots\ldots \cup kill_n$$

$$gen_B = gen_u \cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup \ldots \cup (gen_1 - kill_2 - \ldots - kill_n)$$

Generalized Transfer Functions

Summary

1. Global code optimization involves analyzing code across basic blocks
2. Data flow analysis relies on transfer functions

3. Reaching definition within data-flow analysis is illustrated as an example

Example of Reaching Definitions

Learning Objectives

1. Apply transfer functions for reaching definitions analysis
2. Explore an example of reaching definitions in control flow analysis

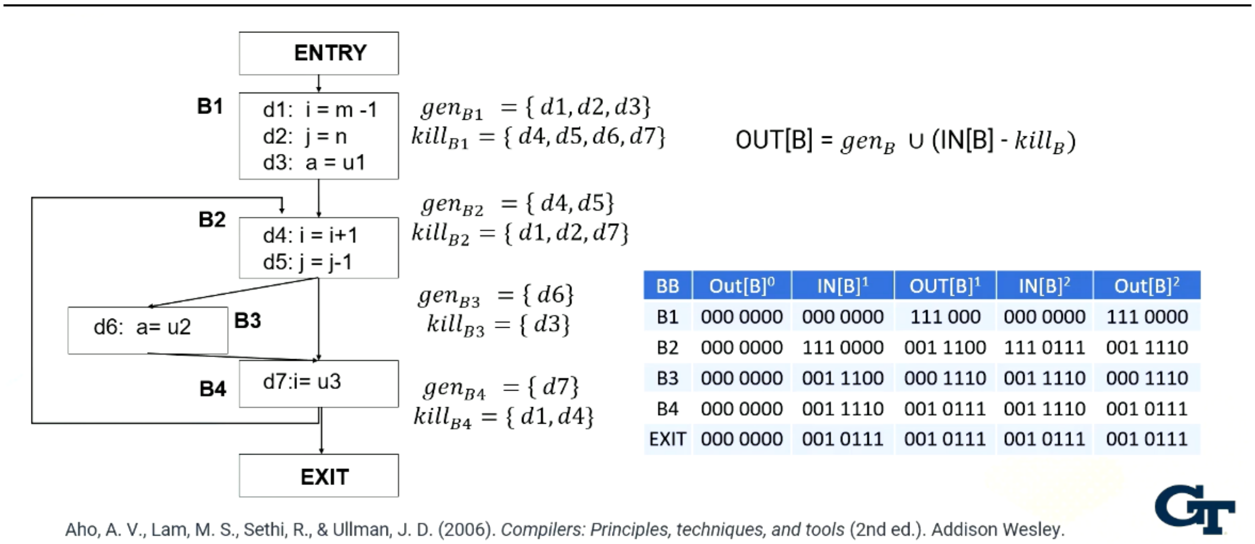
Control Flow Equations

1. IN[B] = Union(OUT[P]) where P is a predecessor of B
2. OUT[ENTRY] = NULL: Boundary condition
3. OUT[B] = gen(B) U (IN[B] - kill(B))

Algorithm

```
OUT[ENTRY] = NULL;
for (each basic block B other than ENTRY) OUT[B} = NULL;
while (changes to any OUT occur) {
    for (each basic block B other than ENTRY) {
        IN[B] = Union(OUT[P])
        OUT[B] = gen(B) U (IN[B] - kill(B))
    }
}
```

Illustration of Reaching Definitions



BB	$Out[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$Out[B]^2$
B1	000 0000	000 0000	111 000	000 0000	111 0000
B2	000 0000	111 0000	001 1100	111 0111	001 1110
B3	000 0000	001 1100	000 1110	001 1110	000 1110
B4	000 0000	001 1110	001 0111	001 1110	001 0111
EXIT	000 0000	001 0111	001 0111	001 0111	001 0111

Reaching Definitions

Summary

1. Reviewed the reaching definitions analysis with an example
2. First, we compute Gen and Kill sets for each basic block and identify predecessors of each basic block
3. Then, we apply the transfer function to all basic blocks
4. The iterative process stops when there are no changes in the OUT for all basic blocks