SPMDfy - A transpiler from CUDA to ISPC

•••

Pradeep Kumar

July 22, 2019

1 st Real Time Analysis Workshop,

Institut Pascal, Université Paris-Saclay, France

ISPC 101

ISPC - Intel SPMD Program Compiler

- **ISPC** is a compiler for a **C-like language** for **SPMD on SIMD architectures**
- Currently it support x86/64, ARM neon, PTX, Xeon Phi(deprecated)
- It is **SIMT programming for CPUs**

ISPC Programming Model

- **ISPC** models a **forall loop** by assigning each iteration to instances called **Program Instances**
- A group of Program Instances is called a **Gang**
- The size of gang is configured during compilation by specifying target size.

ISPC Programming Model

- The **Program Instances** are guaranteed to execute in lock step, so no explicit synchronization(like syncthreads)
- There are no launch configurations, so there is no way of specifying the number of Program Instances in the code

ISPC Programming Model

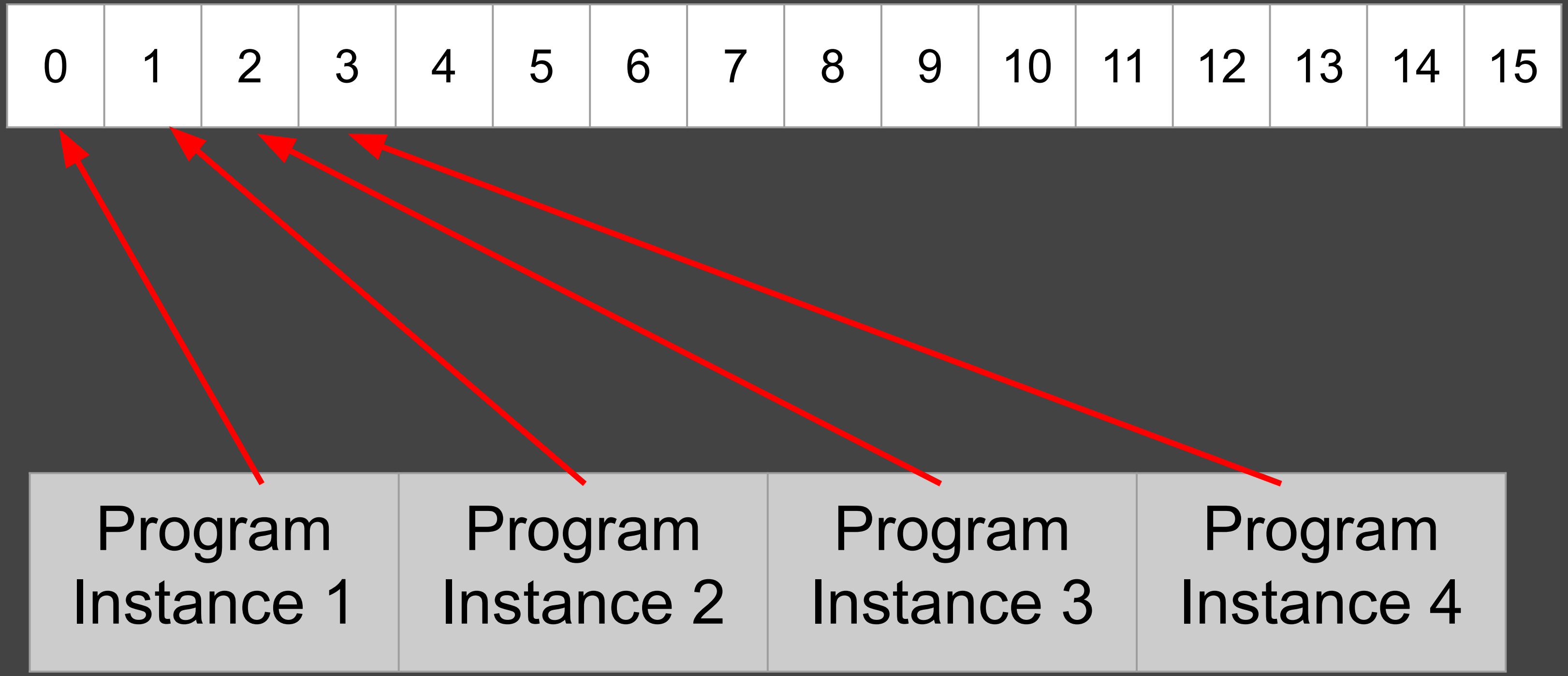
- This removes the dependency of the algorithm on launch configuration. Something is achievable in CUDA and you can find a lot of that in Allen
- So you are stuck with the fixed number of instances and you have to write algorithms around it.

```
export uniform int simple(uniform float vin[],
                          uniform float vout[],
                          uniform int count) {
   foreach (index = 0 ... count) {
       float v = vin[index];
       if (v < 3.)
           v = v * v;
       else
           v = sqrt(v);
       vout[index] = v;
   }
    return 0;
}
```

ISPC Programming Model

```
export uniform int simple(uniform float vin[],
                          uniform float vout[],
                          uniform int count) {
   foreach (index = 0 ... count) {
       float v = vin[index];
       if (v < 3.)
           v = v * v;
       else
           v = sqrt(v);
       vout[index] = v;
   }
    return 0;
}
```

ISPC Programming Model



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Program Instance 1

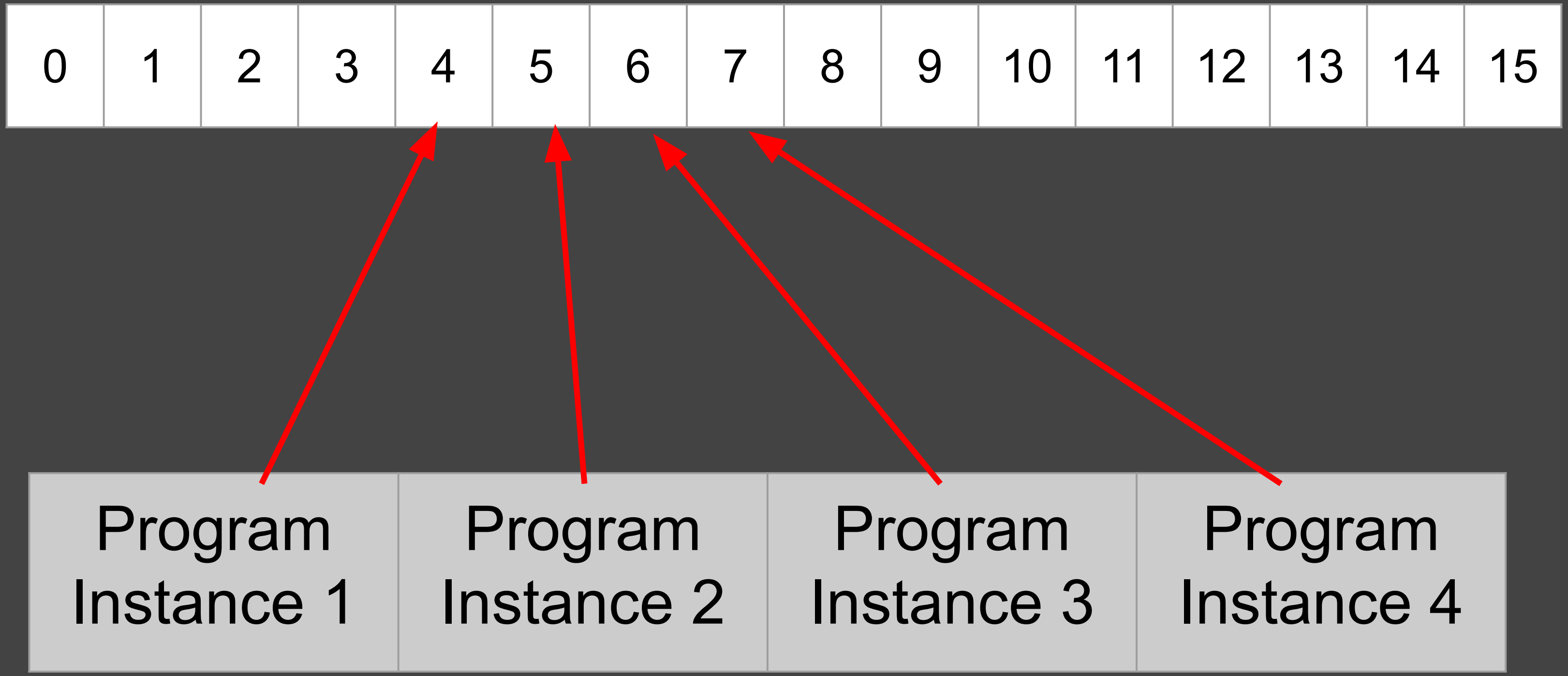
Program Instance 2

Program Instance 3

Program Instance 4

ISPC Execution Model

ISPC Execution Model



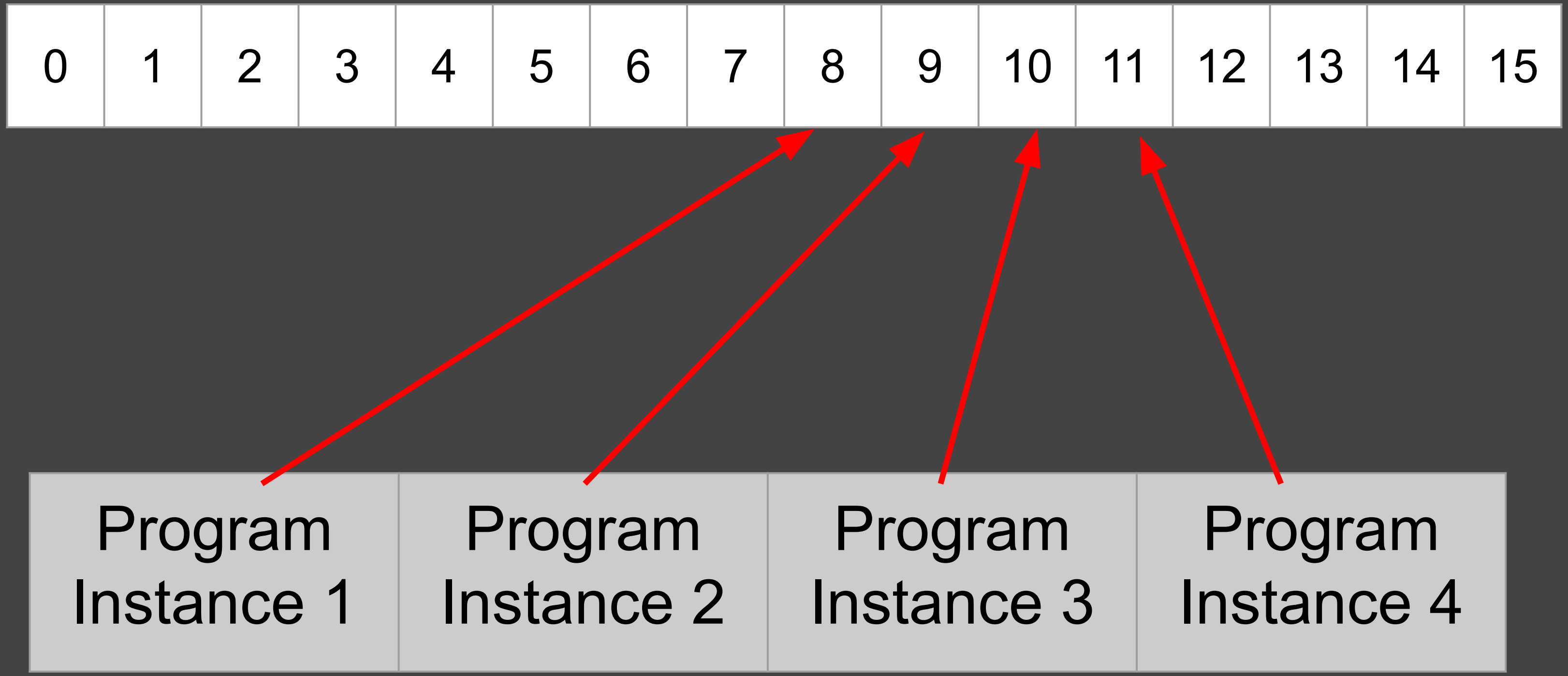
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Program Instance 1

Program Instance 2

Program Instance 3

Program Instance 4



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

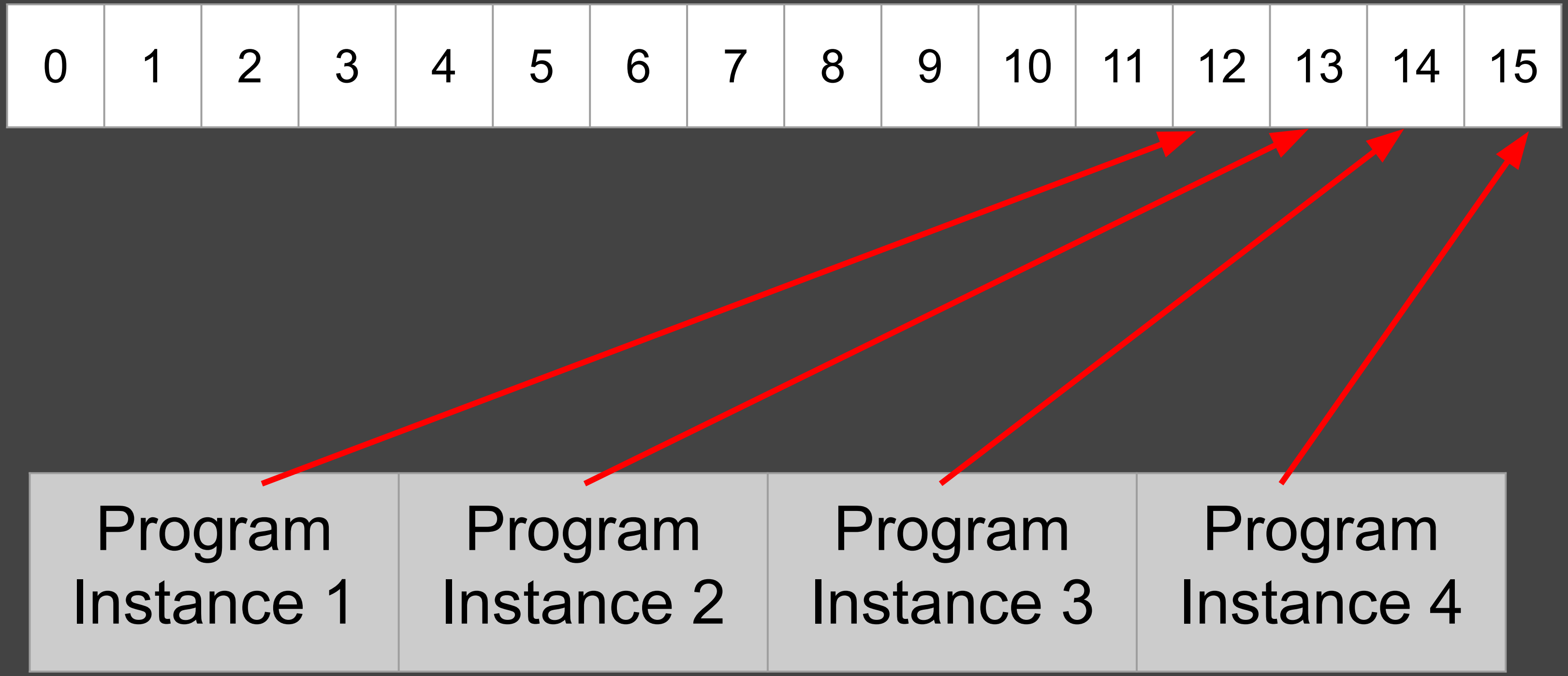
Program Instance 1

Program Instance 2

Program Instance 3

Program Instance 4

ISPC Execution Model



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Program Instance 1

Program Instance 2

Program Instance 3

Program Instance 4

ISPC Execution Model

ISPC Targets

- host
- sse2-i32x4
- sse2-i32x8
- sse4-i32x4
- sse4-i32x8
- sse4-i16x8
- sse4-i8x16
- avx1-i32x4
- avx1-i32x8
- avx1-i32x16
- avx1-i64x4
- **avx2-i32x8**
- avx2-i32x16 ...

avx2-i32x8

ISA : avx2
Mask size: 32 bits
Gang size: 8

```
export uniform int simple(uniform float vin[],
                          uniform float vout[],
                          uniform int count) {
   foreach (index = 0 ... count) {
       float v = vin[index];
       if (v < 3.)
           v = v * v;
       else
           v = sqrt(v);
       vout[index] = v;
   }
    return 0;
}
```

Lock Step Execution

ISPC Programming Model

ISPC vs Intrinsics- Saxpy

```
#include "immintrin.h"

int add_AVX(int size, int *first_array, int *second_array) {
  int i = 0;
  for (; i < size; i += 8) {
   __m256i first_values = _mm256_loadu_si256((__m256i *)&first_array[i]);
   __m256i second_values = _mm256_loadu_si256((__m256i
*)&second_array[i]);

    first_values = _mm256_add_epi32(first_values, second_values);
   _mm256_storeu_si256((__m256i *)&first_array[i], first_values);
  }
  return 0;
}
```

In Case If the Internet doesn’t work

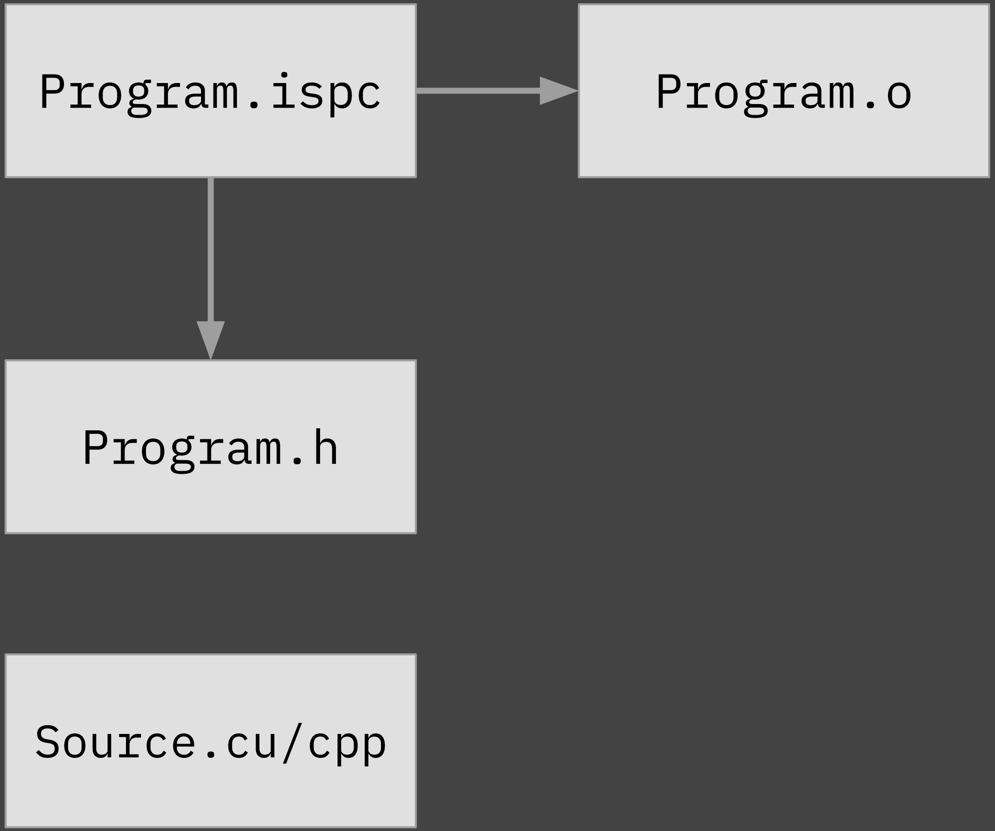
```
export void add(uniform int size,
                uniform int first_array[],
                uniform int second_array[]){
     foreach(i = 0 ... size){
           first_array[i] += second_array[i];
      }
}
```

In Case If the Internet doesn’t work

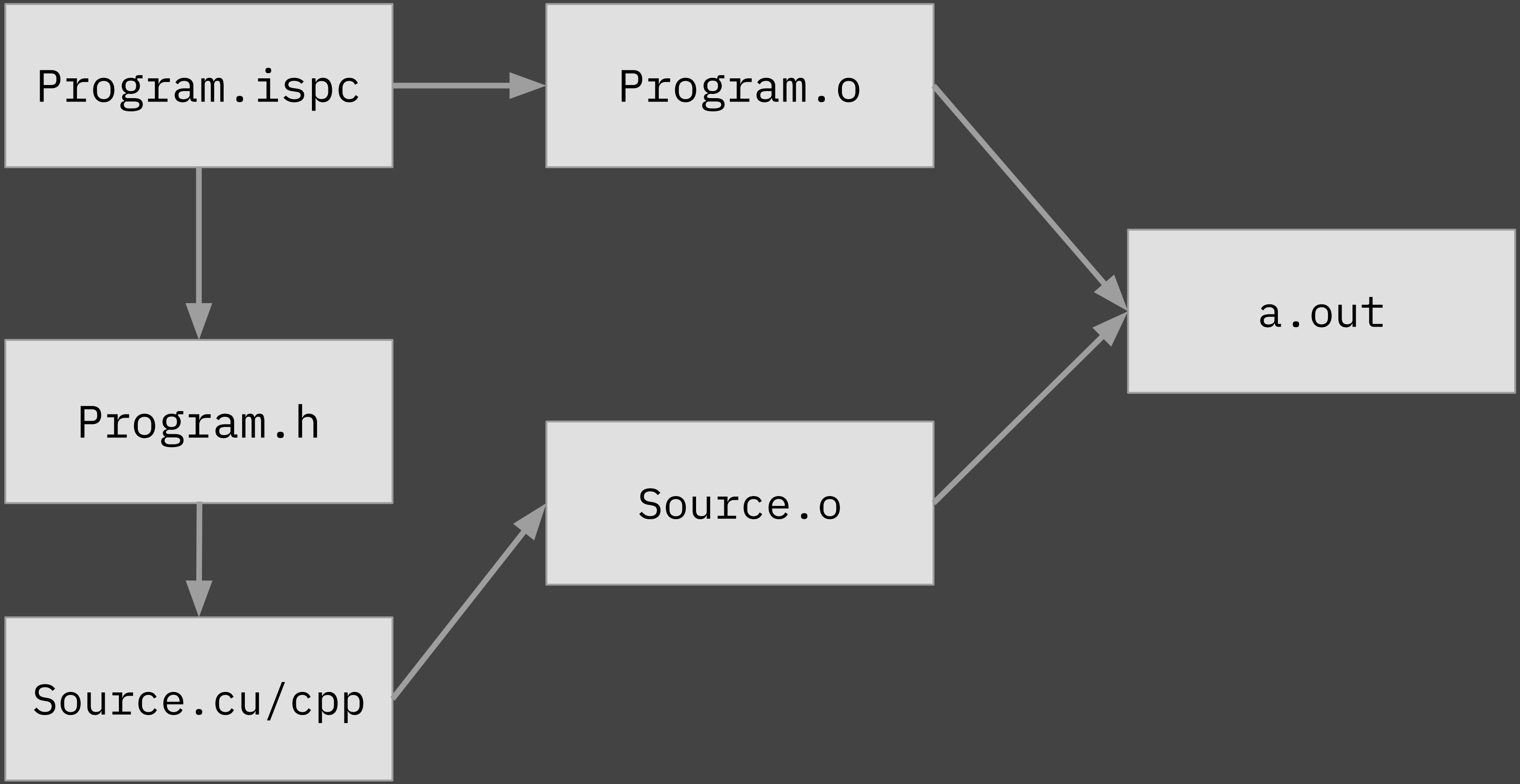
Program.ispc

Source.cu/cpp

Typical ISPC Build phases



Typical ISPC Build phases


Program.ispc

Program.o

a.out

Program.h

Source.o

Source.cu/cpp

Typical ISPC Build phases

Where CUDA and ISPC meet - Saxpy Example

CUDA 101

- CUDA is a programming model for **General Purpose Computation on GPU**
- CUDA exposes GPU as a **heavily multi-threaded coprocessor** with threads executing in groups with hierarchy
- Every thread has a state defined by the **Program Counter(PC)** and are executed in warps of 32(typically)
- The underlying execution model is SIMD

Intuition about CUDA

- In CUDA, A GPU is thought of as a big forall loop implemented in the hardware

```
//Saxpy example
forall(i in range(n)){
  y[i] = y[i] + a * x[i];
}
```

Intuition about CUDA

- We don’t need the for all loop

```
//Saxpy example
~~forall(i in range(n)){~~
  y[i] = y[i] + a * x[i];
}
```

Intuition about CUDA

- But we need the index. The hardware generates the index for us based on the launch config.

```
{
 for(int i =
            threadIdx.x;
            i < n;
      i += blockDim.x){
  y[i] = y[i] + a * x[i];
 }
}
```

Intuition about CUDA

- We need to tell the compiler this code run on the GPU. So we wrap it inside a function

```
__global__
void saxpy(float *y,
     float *x, float a) {
for(int i =
            threadIdx.x;
            i < n;
      i += blockDim.x){
  y[i] = y[i] + a * x[i];
 }
}
```

Intuition about ISPC

- ISPC is limited in it’s SIMD length. So we cannot eliminate the for-loop entirely. foreach does a blocked iteration over N

```
{
foreach(i = 0 ... N){
  y[i] = y[i] + a * x[i];
 }
}
```

Intuition about ISPC

- We should also need to say this piece of code must be vectorized to the ISPC compiler which is marked by export

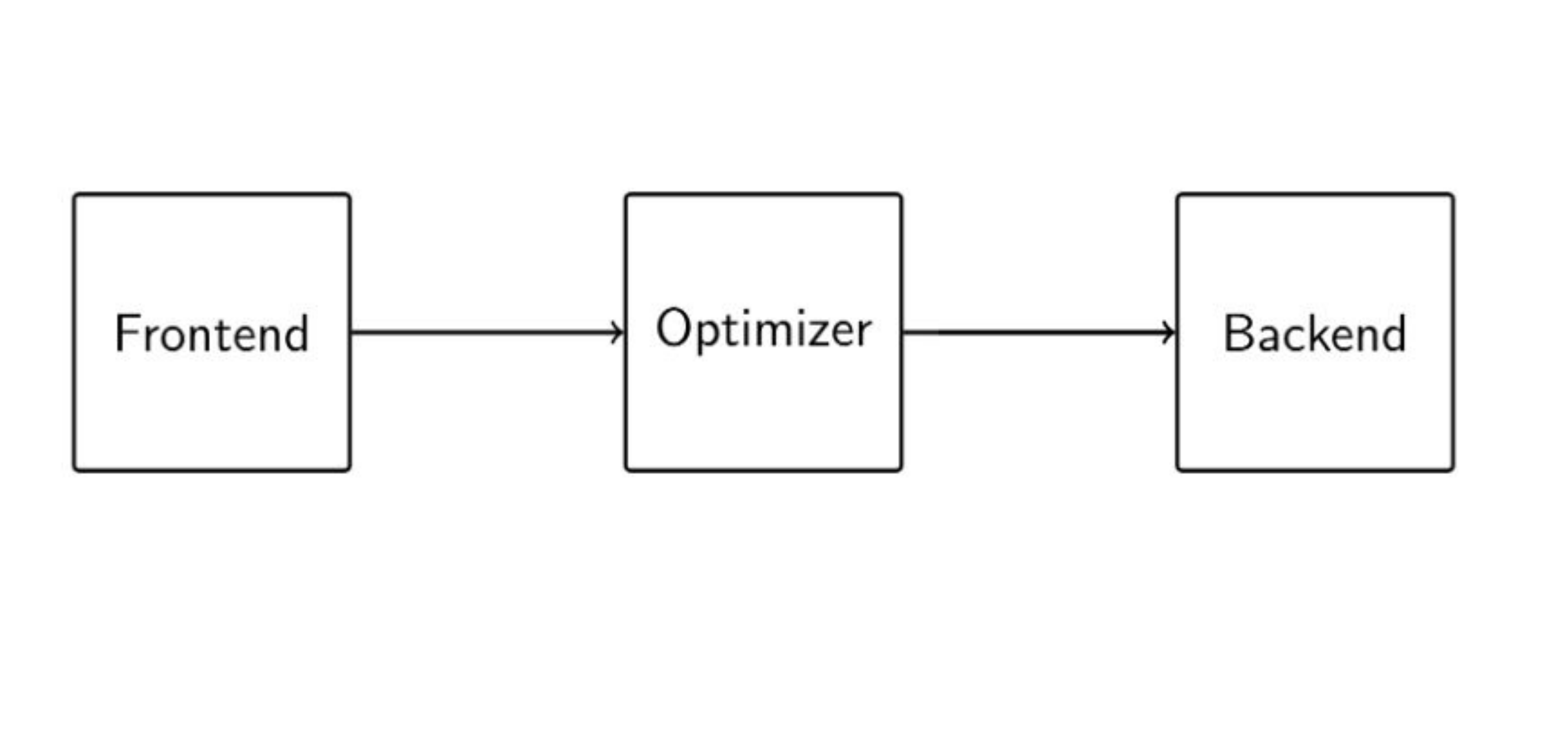
```
export void saxpy(
       uniform int N,
       uniform float *y,
       uniform float *x,
       uniform float a) {
 foreach(i = 0 … N){
  y[i] = y[i] + a * x[i];
 }
}
```

Intuition about ISPC

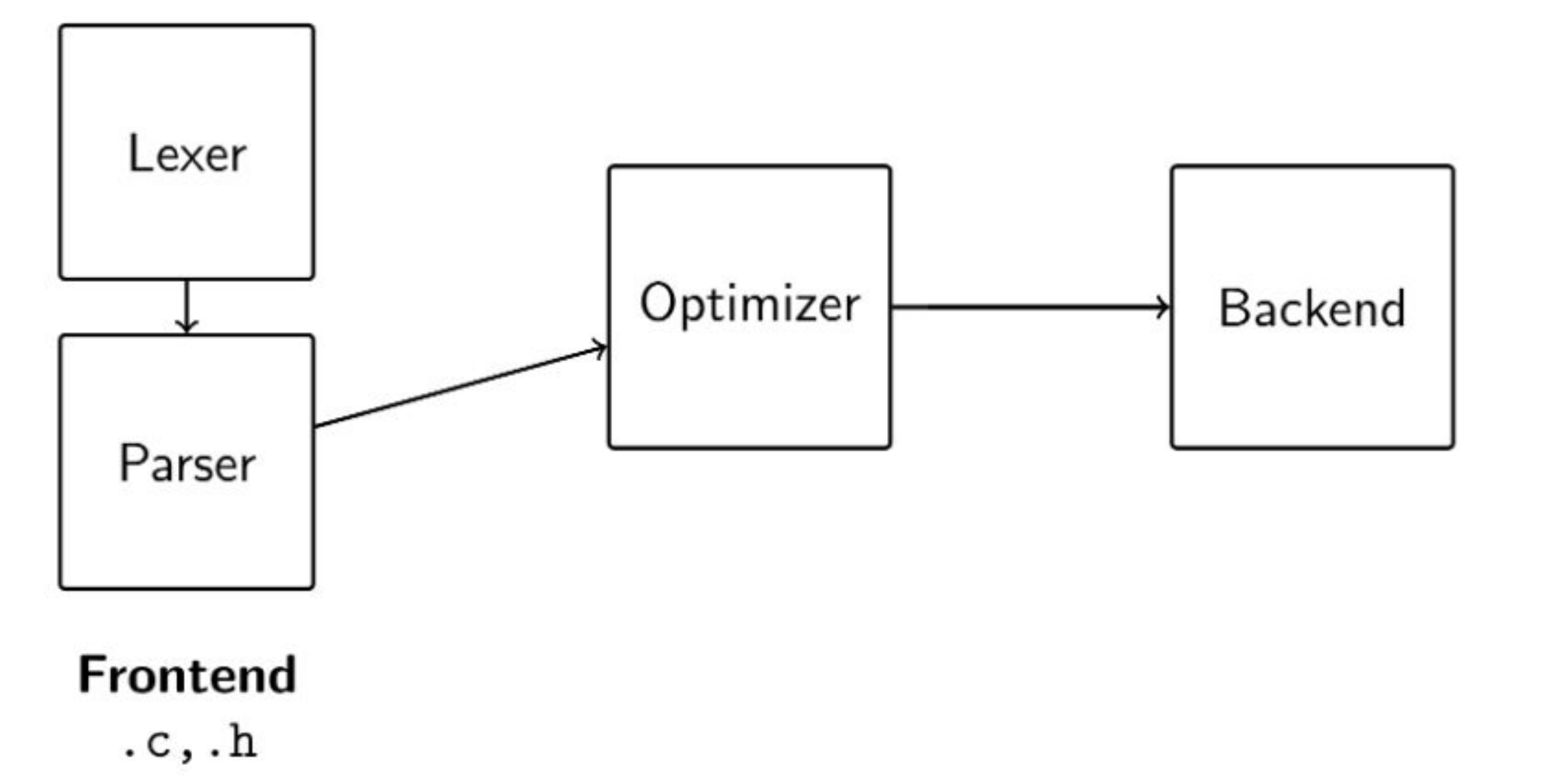
- **uniform** declares that the variable is common to all the program instances. It becomes a scalar variable

```
export void saxpy(
       uniform int N,
       uniform float *y,
       uniform float *x,
       uniform float a) {
 foreach(i = 0 … N){
  y[i] = y[i] + a * x[i];
 }
}
```

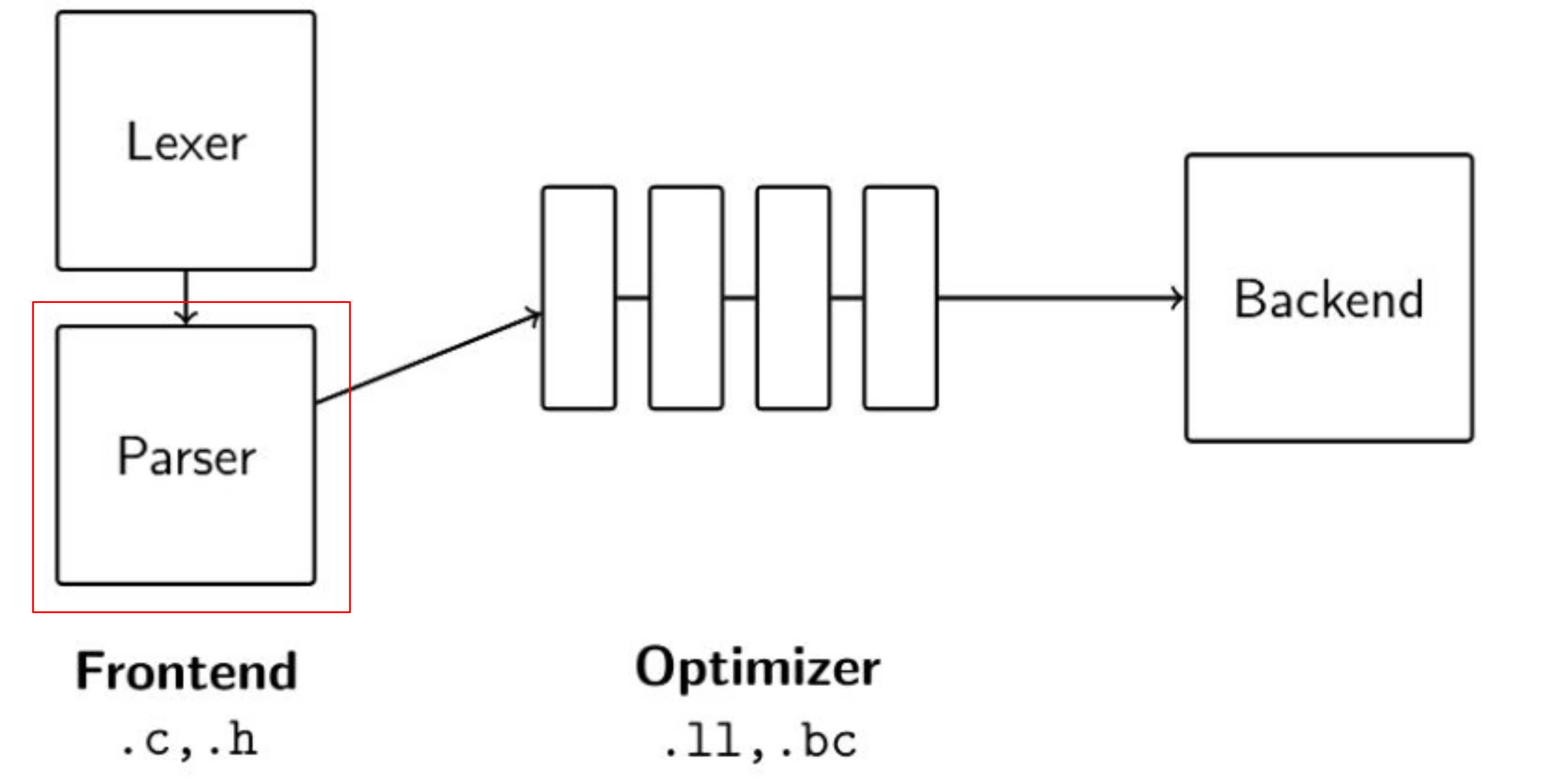
Design of the tool - LibTooling



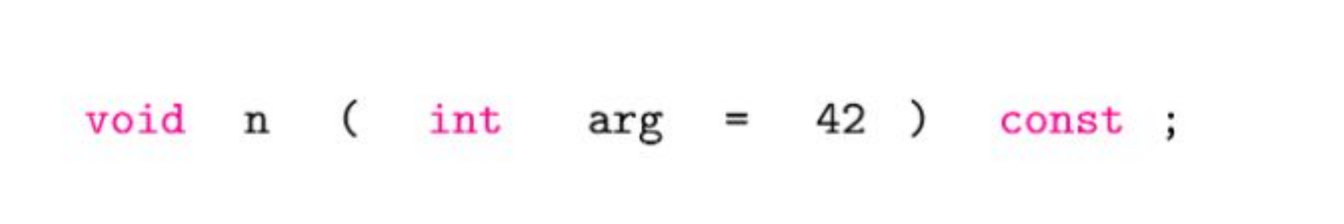
Typical Compiler Pipeline



Typical Compiler Pipeline



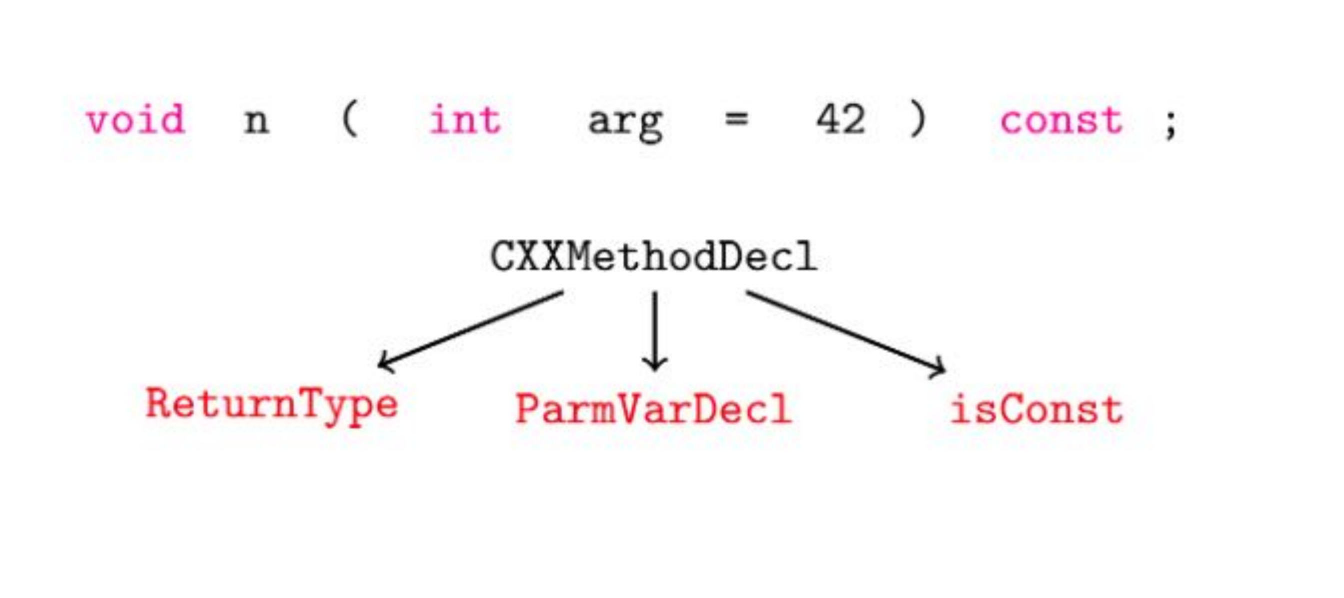
We use the parse the get source info



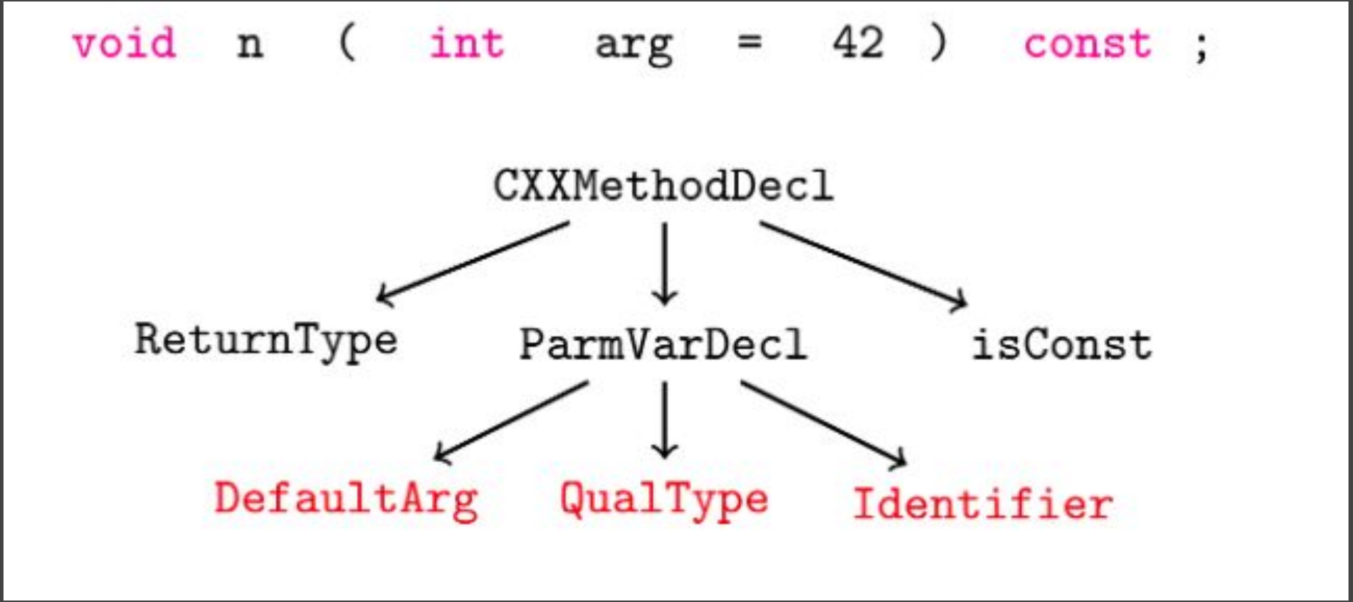
```
void  n  (  int   arg  =  42 )  const ;
```

Clang AST

Clang AST

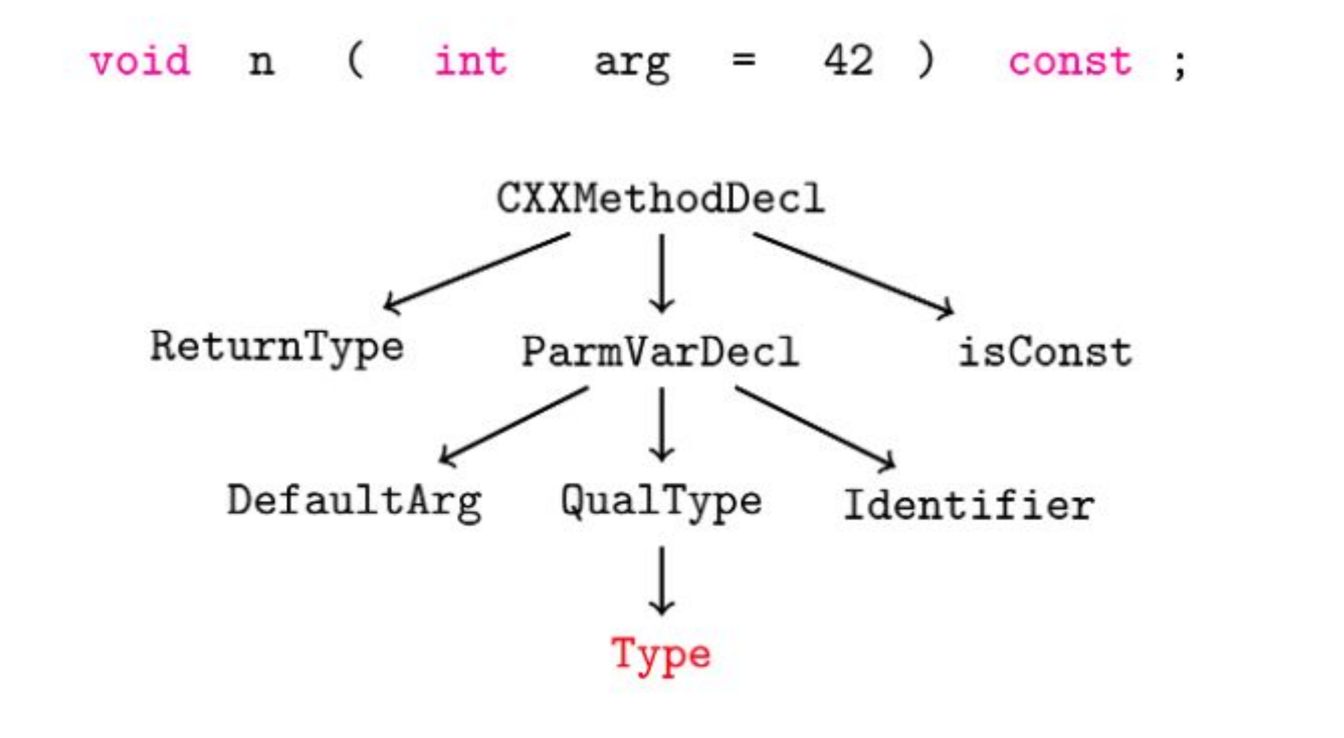


```
void  n  (  int  arg  =  42 )  const ;
```

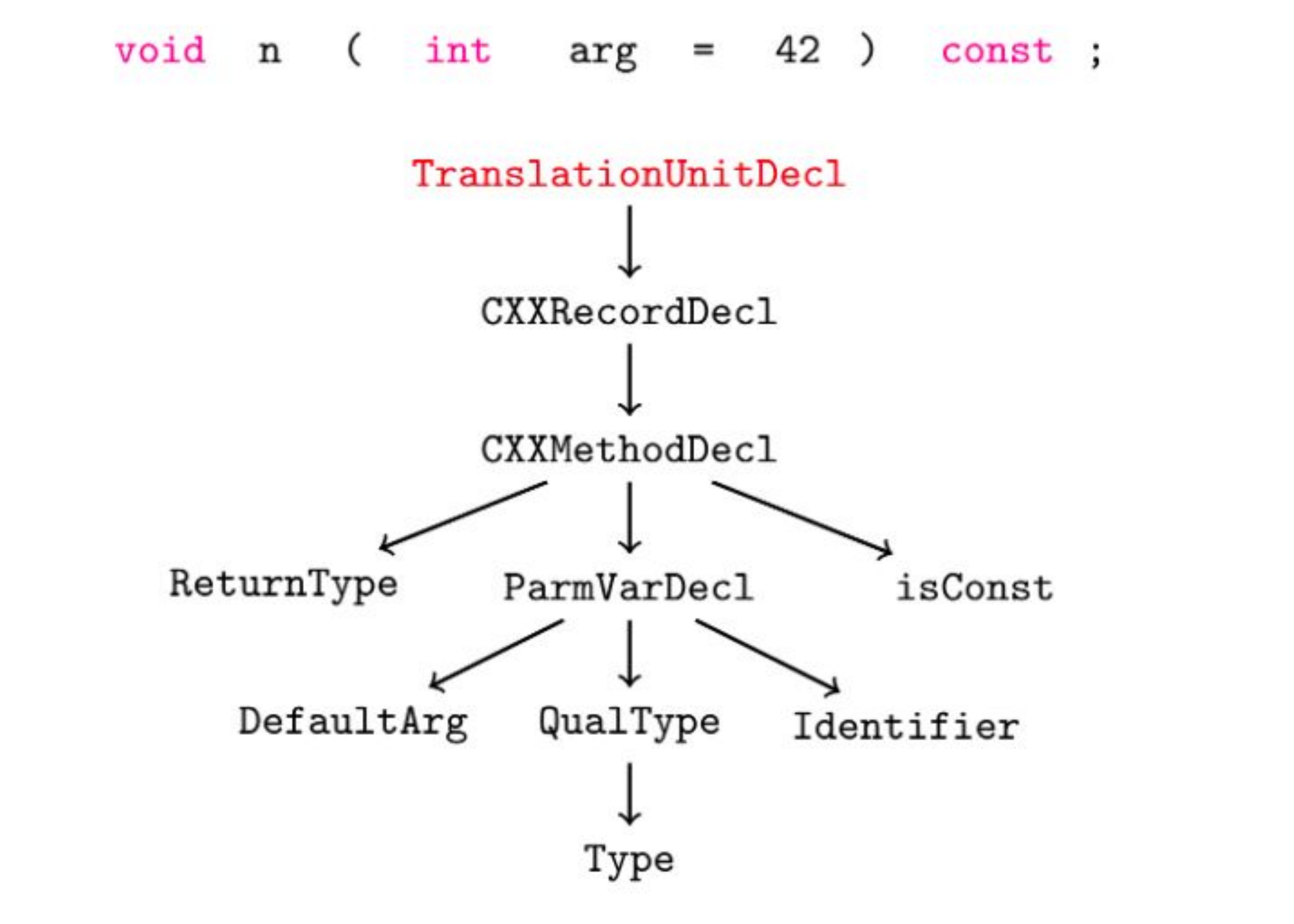


void n (int arg = 42) const ;
CXXMethodDecl
ReturnType
ParmVarDecl
isConst
DefaultArg
QualType
Identifier

Clang AST


void n (int arg = 42) const ;
CXXMethodDecl
ReturnType
ParmVarDecl
isConst
DefaultArg
QualType
Identifier
Type

Clang AST



```
void n ( int arg = 42 ) const ;
```

TranslationUnitDecl

CXXRecordDecl

CXXMethodDecl

ReturnType

ParmVarDecl

isConst

DefaultArg

QualType

Identifier

Type

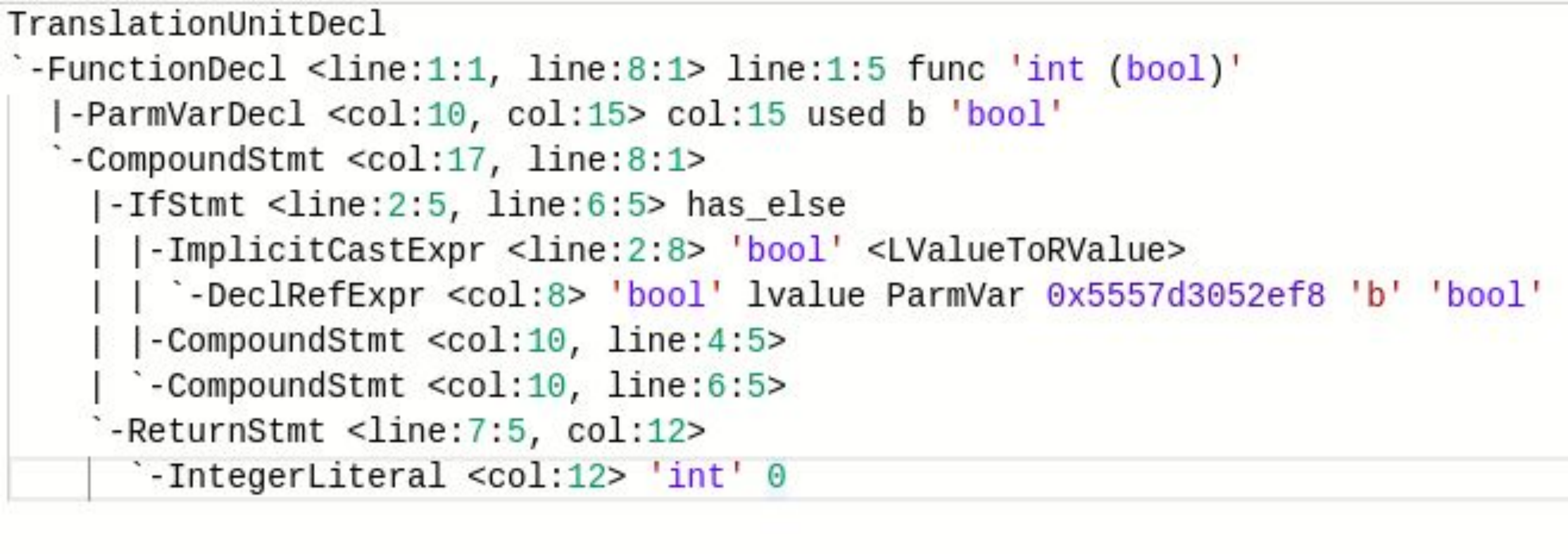
Clang AST

Clang AST

```
int func(bool b){
    if(b){
        // true block
    }else{
        // false block
    }
    return 0;
}
```

```
clang++ -Xclang -ast-dump -fsyntax-only
source.cpp
```

Clang AST



```
TranslationUnitDecl
`-FunctionDecl <line:1:1, line:8:1> line:1:5 func 'int (bool)'
  |-ParmVarDecl <col:10, col:15> col:15 used b 'bool'
  `-CompoundStmt <col:17, line:8:1>
    |-IfStmt <line:2:5, line:6:5> has_else
    | |-ImplicitCastExpr <line:2:8> 'bool' <LValueToRValue>
    | | `-DeclRefExpr <col:8> 'bool' lvalue ParmVar 0x5557d3052ef8 'b' 'bool'
    | |-CompoundStmt <col:10, line:4:5>
    | `-CompoundStmt <col:10, line:6:5>
    `-ReturnStmt <line:7:5, col:12>
      `-IntegerLiteral <col:12> 'int' 0
```

Clang AST

- Clang provides a **rich AST representation**
- It allows developers to **query information** about every token in the code
- Preserves **source information**
- This enables developers to build great tools like **linter, static analyzers, formatting tools, source translators etc.**

Clang AST

- Clang provides access to its AST through the following interfaces
  - **libClang** - A stable C interface, also has python interface
  - **libTooling** - A less stable C++ interface (recommended for complex tools)

What do I use?

- LibTooling
- There are different approaches to build the tool
  - **Clang Plugin** - shared libraries
  - **Clang Tool** - standalone tool

Current Status of the Tool

- The tool works for simple examples
- Partial support for control flow and synchronization statements
- Initial support for Structs and Enums

Future Directions

- The tool is still experimental and evolving
- It works for specific examples
- Improve **Struct support**
- Support for corner cases **__syncthreads are inside for loop**
- Full support for device function **specifically __syncthreads**
- Customizing Translation Unit

Where you can find my Tool?

The Tool

- https://github.com/schwarzschild-radius/spmdfy

Experiments

- https://github.com/schwarzschild-radius/CUDA_to_ISPC/tree/master/Experiments

Where you can find my Me?

- Github - schwarzschild-radius
- MatterMost - 983efeed2c0c62899bc0
- Skype - pradeepisro49
- Mail - pradeepisro49@gmail.com

We have time?

Bonus Slides!!!

SPMDfy Pipeline

```
SpmdfyAction::newASTConsumer() { ...
   // match kernel function
  ADD_MATCHER(
       functionDecl(hasAttr(clang::attr::CUDAGlobal),
                         isDefinition())
                        .bind("cudaKernelFunction"), this);
   // match device function
   ADD_MATCHER(functionDecl(hasAttr(clang::attr::CUDADevice),
               isExpansionInMainFile(), isDefinition(),
               unless(mat::cxxMethodDecl()),
               .bind("cudaDeviceFunction"), this); ... }
```

1. AST Matchers

```
run(const mat::MatchFinder::MatchResult &result) { ...
  if (cudaKernelFunction(result))
       return;
   if (cudaDeviceFunction(result))
       return;
... }
```

2. Matcher Callback

```
bool SpmdfyAction::cudaKernelFunction(
   const mat::MatchFinder::MatchResult &result) { ...
   ...
   clang::Stmt *body = kernel_function->getBody();
   if (body) {
       m_stmt_visitor->Visit(body);
   }
   metadata["context"] = m_stmt_visitor->getContext();
   metadata["body"] = m_stmt_visitor->getFunctionBody();
   ... }
```

3. Matcher Callback

```
bool SpmdfyAction::cudaKernelFunction(
   const mat::MatchFinder::MatchResult &result) { ...
   ...
   clang::Stmt *body = kernel_function->getBody();
   if (body) {
       m_stmt_visitor->Visit(body);
   }
   metadata["context"] = m_stmt_visitor->getContext();
   metadata["body"] = m_stmt_visitor->getFunctionBody();
   ... }
```

3. Matcher Callback

```
SpmdfyStmtVisitor::VisitCompoundStmt(clang::CompoundStmt
*cpmd_stmt) { ...
   for (auto stmt = cpmd_stmt->body_begin(); stmt !=
              cpmd_stmt->body_end(); stmt++) {
       if (Visit(*stmt))
           continue;
    ... }
```

4. Visitor

```
clang::Stmt* SpmdfyStmtVisitor::
            VisitCallExpr(clang::CallExpr*call_expr) { ...
    if (callee_name == "__syncthreads") {
        m_block++;
        return call_expr;
    } else if (callee_name == "printf") {
        return call_expr;
    }
    ...
}
```

5. Visitor

Example - Use Case

Allen Example

Allen Example

- Saxpy
- Saxpy Header
- Handler
- Sequence
- Visitor

Question?

References

- https://ispc.github.io/ispc.html
- https://github.com/ispc/ispc/
- https://clang.llvm.org/docs/LibASTMatchersReference.html
- https://www.cs.virginia.edu/~cr4bd/3330/F2018/simdref.html
- https://clang.llvm.org/docs/IntroductionToTheClangAST.html
- https://clang.llvm.org/doxygen/