The ARES High-level Intermediate Representation

Nick Moss, Kei Davis, Pat McCormick

11/14/16
About ARES

- HLIR is part of the ARES project (Abstract Representations for the Extreme-Scale Stack)
- Inter-operable tools and approaches for programming next-generation architectures
- LANL (Pat McCormick) + ORNL (Jeffrey Vetter)
- Funded by: Advanced Scientific Computing Research, Office of Science, of the United States Department of Energy
HLIR Motivation

- LLVM remains a purely sequential representation while parallel programming is becoming increasingly ubiquitous with the end of Moore’s law.

- Parallel functionality is often achieved through libraries, but use of libraries alone lead to missed optimization opportunities and programmability challenges.
HLIR Motivation

- Essential information about high level structures of the program is lost such as loops and typically has to be reconstructed from IR.

- Approaches such as Clang’s OpenMP perform analysis/transformations in the front-end. It would be preferable to write such parallel transformations once in the backend for targeting by multiple frontends.
Adding parallel extensions directly to LLVM is a challenging problem; adds complexity and would be very disruptive to core features of LLVM, e.g. control flow graph analysis and other types of analysis.

IR alone is perhaps too low-level; sequential may not be the best form; we posit the need for an AST-like representation.
To address these concerns and needs we created HLIR (High Level Intermediate Representation)… to allow multiple frontends to target parallel functionality.

HLIR is an extension of LLVM, taking advantage of LLVM’s broad capabilities and infrastructure – HLIR can be viewed as a superset of LLVM.

HLIR = representation + code generation / runtime interface
Parallel Constructs

Our current implementation supports:

- tasks
- parallel for
- parallel reduce
- communication and synchronization building blocks
HLIR Features

- Extensibility
- Ease of use
- Human readable and in-memory representations
- Targetable by diverse types of frontends: C++, Fortran, … any language that uses LLVM for code generation
- Nested/hierarchical to represent AST-like structures such as parallel for loops
- Mutability and successive transformation
HLIR Flow

- C++ Frontend: Lex/Parse/Sema → CodeGen
- Fortran Frontend: Lex/Parse/Sema → CodeGen
- Charm++ Frontend: Lex/Parse/Sema → CodeGen

HLIR → LLVM IR + HLIR Intrinsics

HLIR Runtime Library → Executable → IR + Calls to HLIR Runtime → HLIR Pass
HLIR Module

- HLIR module – one-to-one correspondence with IR module

- HLIR module provides methods for creating parallel constructs, e.g. parallel for, contains top-level metadata

- For example, in the case of parallel for/reduce, HLIR sets up an outlined function and provides entry points for the body to be defined or induction or reduce variables retrieved
Sample HLIR Representation

```
body: <<<function:
define void @hlir.parallel_for.body(i32 %i){
  entry:
    %index.ptr = alloca i32
    store i32 %i, i32* %index.ptr %0
    %call = call i32 (i8*, ...)
    ... ret void
}>>>}
```
HLIR nodes

- Leaf nodes: symbol, string, floating, integer, IR Value, IR Type, arbitrary sequence of IR wrapped in function, etc.

- Dynamic/flexible, recursive nodes – heterogenous types – vector, symbol map

- Constructs: Parallel For/Reduce, Task, etc. Constructs have key-mapped values, e.g: body, induction var, return type, etc.
Data Dependencies & Capturing

- One of the important benefits/abstractions that HLIR provides is capturing of data dependencies and bundling them up into a struct so that can be queued along with a function ptr to outlined function to our thread pool.

- A front-end performing code generation can conveniently neglect that values appearing within, e.g. a task body were defined externally.
Simple prototype runtime – defines an ABI interface which is potentially swappable with a different runtime

Uses Argonne Argobots user-level threads => yield(), solves recursion problem, thread waits while occupying

Thread pool, depth, synch
Synchronization – barrier, “virtual” semaphores
Communication building blocks – channels, message handlers
Tasks Implementation

- Body specified by frontend => outline, data capturing

- Transform calls to task-marked functions into HLIR intrinsic for task launch

- The return value becomes a future, HLIR pass looks for uses of this value and turns them into runtime calls to yield/await this future.
Parallel For/Reduce Implementation

- Body specified by frontend => outline, data capturing, same machinery as task

- Nested parallel for/reduce loops add complexity, data dependencies are unwrapped, and re-queued at each level

- Parallel Reduce – divide and conquer – entire algorithm code generated
Clang-based Frontend

- We implemented a proof of concept frontend by extending Clang to target HLIR. Adds first-class support for tasks and parallel for/reduce.

- Only required <= approximately 100 lines of code added to Clang for each construct.
task int fib(int i){
    if(i <= 1){
        return i;
    }
    return fib(i-1) + fib(i-2);
}
Clang-based Frontend – Parallel For

```c
float A[SIZE];
for(auto i : Forall(0 , SIZE)){
    A[i] = i;
}
```
float sum = 0.0;
for(auto i : ReduceAll(0 , SIZE, sum)){
    sum += 1.0;
}
Related Work

- Open64
- Diderot
- GCC Gimple
- OpenARC
- OpenCL SPIR
- Scout, Kokkos Clang (LANL)
Future Work

- Additional parallel constructs, e.g. data layout and memory placement, data parallel, etc.

- Distributed functionality, integrate with communication infrastructure, tasks: data dependence

- Optimize execution/runtime, depth, priority, chunking parallel for, etc.

- Next phase: targeting/extending HLIR with OpenMP/OpenACC/pragma-based semantics
Conclusion

- Multiple frontends can target HLIR to benefit from a centrally optimized lowering and runtime system.

- Target at a high-level while attaining benefits of low-level code generation and optimization.

- Flexible and hierarchical, extensibility.
Questions?

- Thanks for your time!
- ARES HLIR can be found at: https://github.com/losalamos/ares
- Questions?