Parallel and Distributed Extensions to the LLVM Compiler

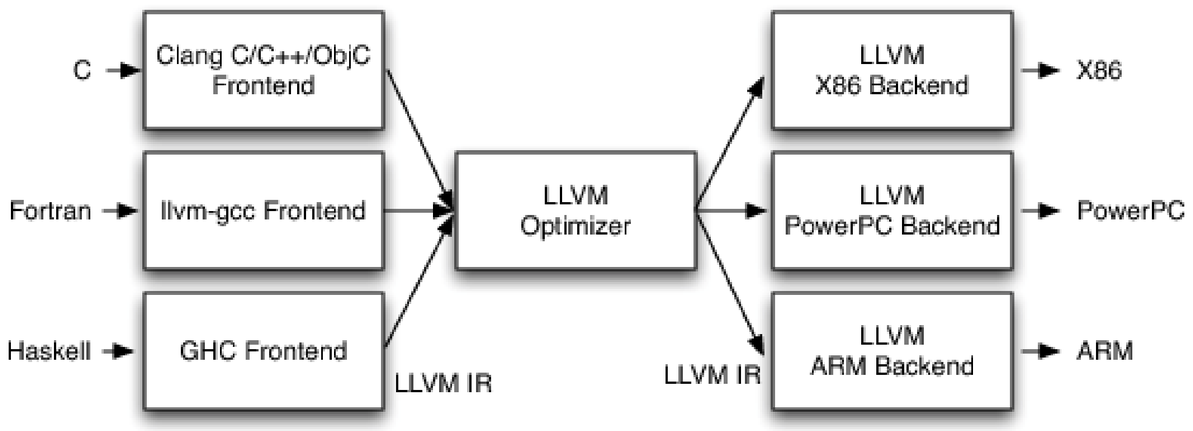
Lefteris Ioannidis, Shoaib Kamil, Saman Amarasinghe

MIT EECS Undergraduate Research and Innovation Scholar

MIT EECS ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

SuperUROP

The LLVM Compiler - Overview

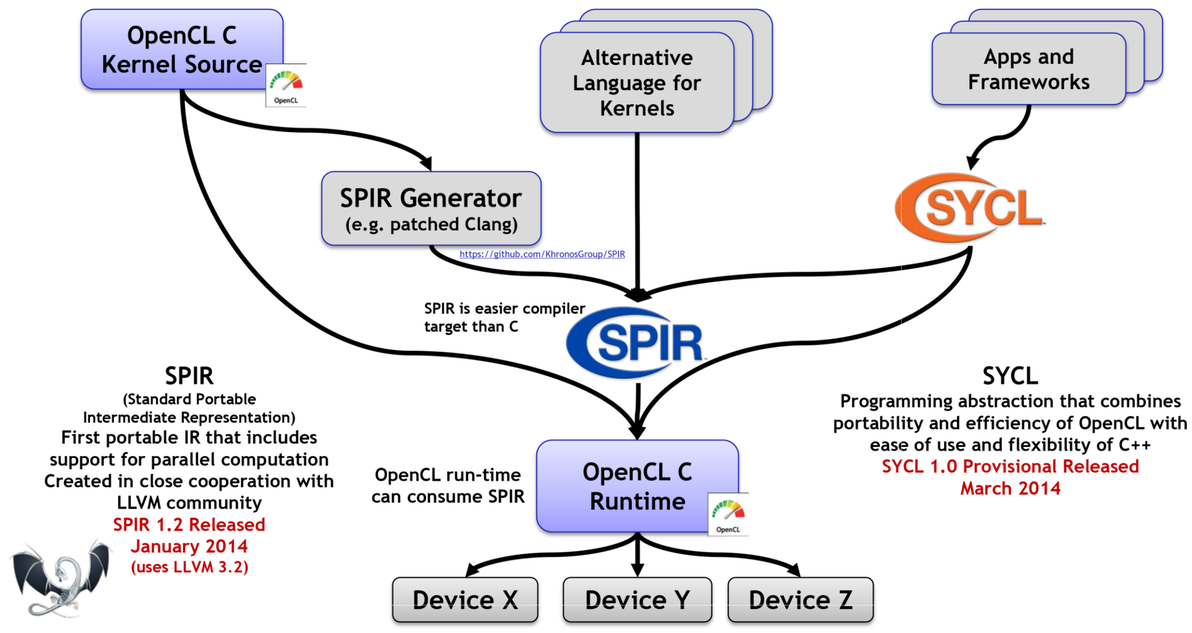


Objective

Propagate parallel interfaces exposed in the frontend (Clang) to the LLVM Backend, by designing a parallel interface in the Intermediate Representation level (LLVM IR).

Previous Work

Khronos Group OpenCL SPIR and Runtime



Provides	A parallel intermediate representation for OpenCL
Lacks	Support for other Parallel Interfaces and Runtimes

Common Parallel Interfaces

POSIX/Pthreads

- `pthread_create()`
- `pthread_join()`
- `pthread_kill()`
- `pthread_mutex_lock()`
- `pthread_mutex_unlock()`
- `pthread_cond_wait()`
- `pthread_cond_signal()`
- `int pthread_barrier_wait()`

OpenMP

- `#pragma omp parallel`
- `#pragma omp for`
- `#pragma omp sections`
- `#pragma omp teams`
- `#pragma omp single`
- `#pragma omp task`
- `#pragma omp barrier`
- `#pragma omp atomic`

Intel Cilk

- `cilk_spawn()`
- `cilk_sync()`
- `cilk_for()`
- `CILK_C_REGISTER_REDUCER()`
- `CILK_C_UNREGISTER_REDUCER()`
- `vectors`

Legion

- `Task`
- `region<int>`
- `reads()`
- `writes()`
- `Partition<>`
- `array<>`

Challenge

Translate the variety of these interfaces to a single Parallel Intermediate Representation (PIR). This PIR has to be an extension of the LLVM IR and be compatible with the LLVM Backend.

SPIRE: Sequential to Parallel IR Extension

Language	Execution	Synchronization				Memory	
	Parallelism	Task creation	Task join	Point-to-point	Atomic section	Model	Data distribution
Cilk (MIT)	—	spawn	sync	—	cilk_lock	Shared	—
Chapel (Cray)	forall coforall cobegin	begin	—	sync	sync atomic	PGAS (Locales)	(on)
X10(IBM), Habanero-Java(Rice)	foreach	async future	finish	next force get	atomic isolated	PGAS (Places)	(at)
OpenMP	omp for omp sections	omp task omp section	omp taskwait omp barrier	—	omp critical omp atomic	Shared	private, shared...
OpenCL	EnqueueND-RangeKernel	EnqueueTask	Finish EnqueueBarrier	events	atom_add, ...	Distributed	ReadBuffer WriteBuffer
MPI	MPI_Init	MPI_spawn	MPI_Finalize MPI_Barrier	—	—	Distributed	MPI_Send MPI_Recv
SPIRE	**sequential, parallel**	**spawn**	**barrier**	**signal, wait**	**atomic**	Shared, Distributed	**send, recv**

Porting to LLVM

SPIRE can be ported to LLVM IR by adding a small set of attributes to standard LLVM objects.

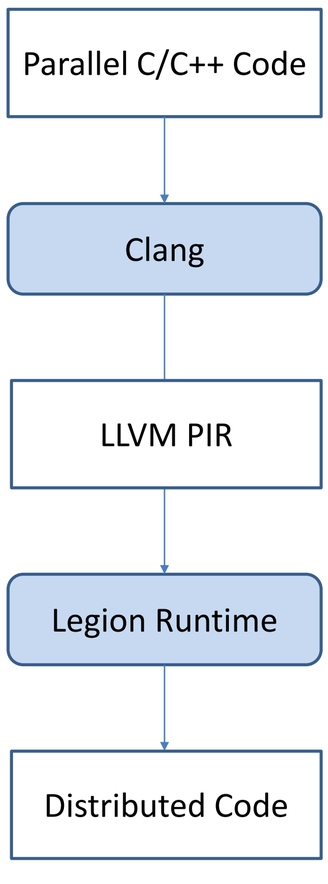
LLVM Parallel IR

This is our plan to port SPIRE to LLVM.

- An `execution` attribute is added to `function` and `block`: a parallel basic block sees all its instructions launched in parallel (in a fork/join manner), while all the blocks of a parallel function are seen as parallel tasks to be executed concurrently;
- A `synchronization` attribute is added to `instruction`; therefore, an instruction can be annotated with `spawn`, `barrier`, `single` and `atomic` synchronization attributes. When one wants to deal with a sequence of instructions, this sequence is first outlined in a function, to be called instead; this new call instruction is then annotated by the proper synchronization attribute, such as `spawn`, if the sequence must be considered as an asynchronous task. A similar technique is used for the other synchronization constructs `barrier`, `single` and `atomic`.
- As LLVM provides a set of intrinsic functions [13], SPIRE functions `newEvent`, `signal` and `wait` for handling point-to-point synchronization, and `send` and `recv` for handling data distribution, are added to this set.

Next Step: Distributed Runtime/Legion

Parallel to Distributed



Challenge

Propagate the Parallel Interfaces, through the LLVM PIR, into a distributed runtime like the Legion system developed at Stanford University.