LLVM-based dynamic dataflow compilation for heterogeneous targets

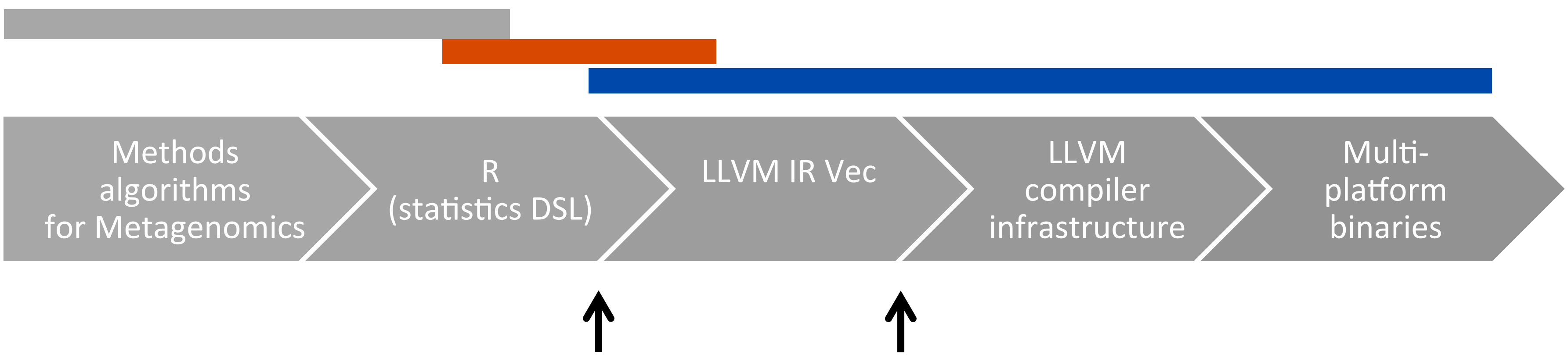
V. Ducrot, K. Juilly, S.Monot,
G. Bayle Des Courchamps — AS+ Groupe Eolen

T. Goubier — CEA List /DACLE /LCE

Benoit Da Mota — Anger University

Donnons de la suite à vos idées…

Context : the MACH Project



Front end R to IR Vec + Front end IR to LLVM

Heterogeneous HPC aware front end R to LLVM

Accelerating R on heterogeneous targets

- R: the dominant language for statistical analysis
  - Used by everyone, everywhere
  - Fast to use (easy scripting)
  - Slow to use (with large data sets)
- MACH: DSeLs for heterogeneous computing
  - R is a DSL (statistics)
  - R can be used to target accelerated heterogeneous computing
- R in MACH
  - Extract / Transform data parallelism in R scripts
    - In a R front-end
  - Specify it to target:
    - GPUs (Nvidia/AMD)
    - CPU accelerators (Intel MIC)

EOLEN CONSEIL ET INGÉNIERIE · FROM RESEARCH TO INDUSTRY cea tech · MACH · ITEA 2 INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Compilation + runtime tool chain

Complex system

- Task management
- Non trivial algorithmic
- Multi-target implementation

Toolchain to simplify programming

- Automated task extraction from the code
- Automated insertion of runtime control function
- Constraints on data structure to simplify analysis and give better performance

EOLEN CONSEIL ET INGÉNIERIE

FROM RESEARCH TO INDUSTRY cea tech

MACH

ITEA 2 INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Three stage compilation system

- Frontend
  - Goes from R to middle-end IR
- Middle end
  - Split for multi-target management
  - Re-express code as standard LLVM adapted to target
- Backend
  - Standard LLVM passes and backend
  - A specific pass to insert runtime management calls

Dataflow runtime

- Parallelism is expressed as task and data dependency
  - Easy to generate parallelism from the compiler
- Execution is out-of-order with sequential consistency guaranties
  - Efficient
  - Hard to debug
- Natural auto-tuning application
- Memory needs to be managed

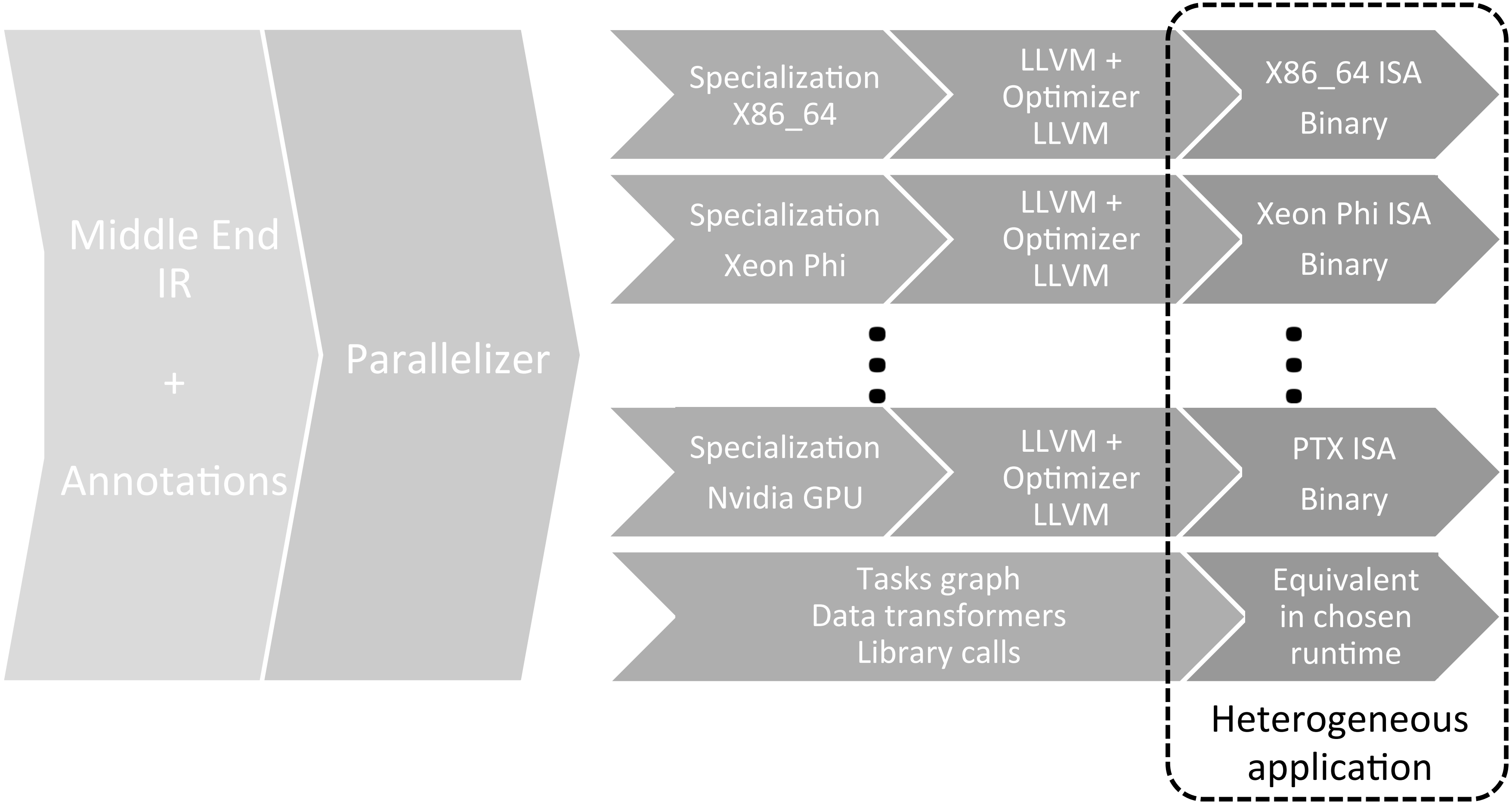
Managed Memory

Managed memory	Induced constraints
• A data driven execution model • Unified view on memory	• Referenced memory • No pointer arithmetic • No global • Library call must be wrapped (thread safety)

Runtime insertion at middle-end level

- Easier manipulation of multiple implementations
- Simplified frontend by removing most of the runtime knowledge from it
- Simplified way to add hardware specific analysis by leveraging LLVM infrastructure
- Target Runtime is currently starPU from Inria Bordeaux
    - http://starpu.gforge.inria.fr

Compilation Middle-end and Backend



Middle-end IR

- Build on top of the existing LLVM IR
  - Add support for arbitrary length vector
  - Add support for managed containers
  - Add intents markers on function(task) declarations
  - Add task declarations / submit marker
  - Add intrinsic vector operations

Middle-end IR
Arbitrary length vectors

- Arbitrary length vectors (ALV)
  - Marked as 0 length in IR
  - Managed data specifics load/store using them (effective size are derived from them at runtime)

```
%f0v = call <0 x float>(%nd_array_float_t*)* @ndarray.load.float(%nd_array_float_t * %f0)
call void @ndarray.store.float(%nd_array_float_t * %u1, <0 x float> %u1v)
```

  - *Masking* intrinsic

```
%mr = call {}* @llvm.mach.mask.activate.v0i1(<0 x i1> %alltrue)
 %merge2 = call <0 x i32> @llvm.mach.mask.merge.v0i32({}*%mr, <0 x i32> %r, <0 x i32> %alvizero)
call void @llvm.mach.mask.deactivate({}* %mr)
```

  - *Reduce / scan* intrinsic

```
%v3 = call <0 x float> @llvm.mach.alv.reduce.max.v0f32(<0 x float> %v2)
```

All classical vector operations are supported on ALV

EOLEN CONSEIL ET INGÉNIERIE · cea tech · MACH · ITEA 2

Middle-end IR
Managed data Containers

- ND-arrays
  - Python like ND-array as standard containers for tables
  - Views support
  - Manipulation functions for copy, extraction…
- Raw Data
  - Managed segment of memory without an attached layout
  - Task need using them cannot be written with arbirary length vector

All data containers provide also functions for accessing them outside the runtime.

Middle-end IR
Task Management

- Metadata for marking task call
- Metadata for expressing patterns on task implementation
  - ufunc
  - rfunc
  - scan
- Intents on managed data (read, write, scratch…)
  - Generated by analysis pass

IR specializing passes

- Task specializing
  - Architecture dependent rewriting of Middle-end IR to IR
  - Output standard LLVM IR adapted to a given target
- Workflow management
  - Takes the code with calls marked as task
  - Replace calls by task preparation and submission
- Multi-implementation management
  - Create initialization/finalization call to the runtime referencing each specialized implementation

Application and performance tuning

- The runtime supports multiple implementation for a given task on a given hardware
- Our pass generates multiple implementations
- The runtime chooses the best implementation according to the data sizes

Performance and results

- We have measured the execution time between benchmarks implemented in C and the same benchmarks implemented in middle-end IR

Code	GCC 4.9	icc 13	clang 3.6	IR version
Jacobi	28.71	31.38	41.9	29.72
Lattice Bolzmann	59.63	71.10	74.64	59.43

Conclusion

- We proposed an infrastructure to compile heterogeneous program on a dataflow runtime
- The middle-end IR enables us to compile for multiple target at reasonable performance
- Porting to a new target doesn't change the frontend