PENCIL: Towards a Platform-Neutral Compute Intermediate Language for DSLs

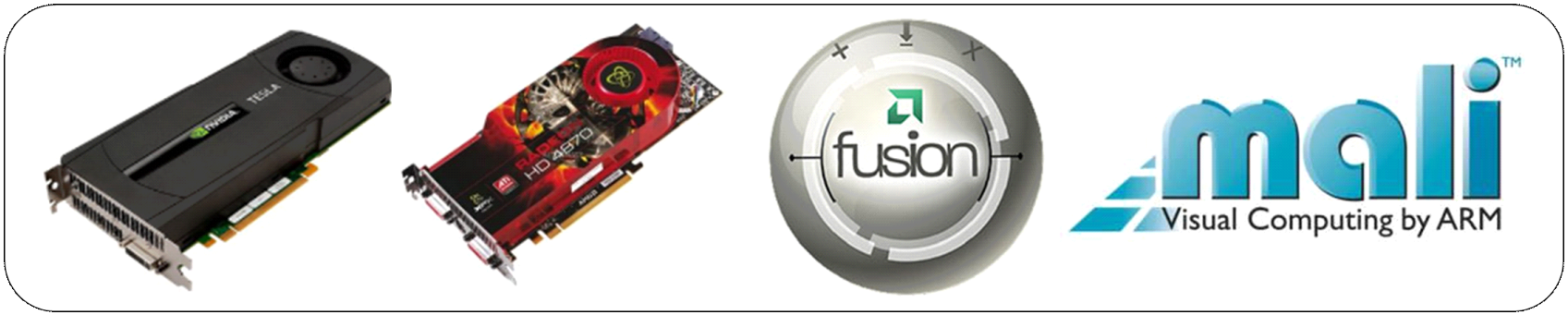
Riyadh Baghdadi A. Cohen S. Guelton S. Verdoolaege
J. Inoue T. Grosser G. Kouveli A. Kravets
A. Lokhmotov C. Nugteren F. Waters A. Donaldson

ARM
Imperial College
ENS/INRIA, Paris
-
Riyadh.Baghdadi@inria.fr

November 16, 2012

Accelerators (mainly GPUs)

- GPU architectures are becoming popular accelerators
- GPUs: high performance and low energy consumption



Problems:

- Highly optimized code is *hard* to write
- Optimization for *diverse* platforms
- Maintenance of *multiple* sources

→ DSLs are being used to target accelerators

DSLs enable more optimization opportunities

- Problem: general purpose languages are not optimization-friendly
  - no semantic information about the algorithm
  - expressive $\rightarrow$ ambiguity disables optimizations (e.g., aliasing).
- But compiling DSLs directly into OpenCL or CUDA is not advisable.
- Solution: target an appropriate intermediate language (IL) and benefit from the optimization framework

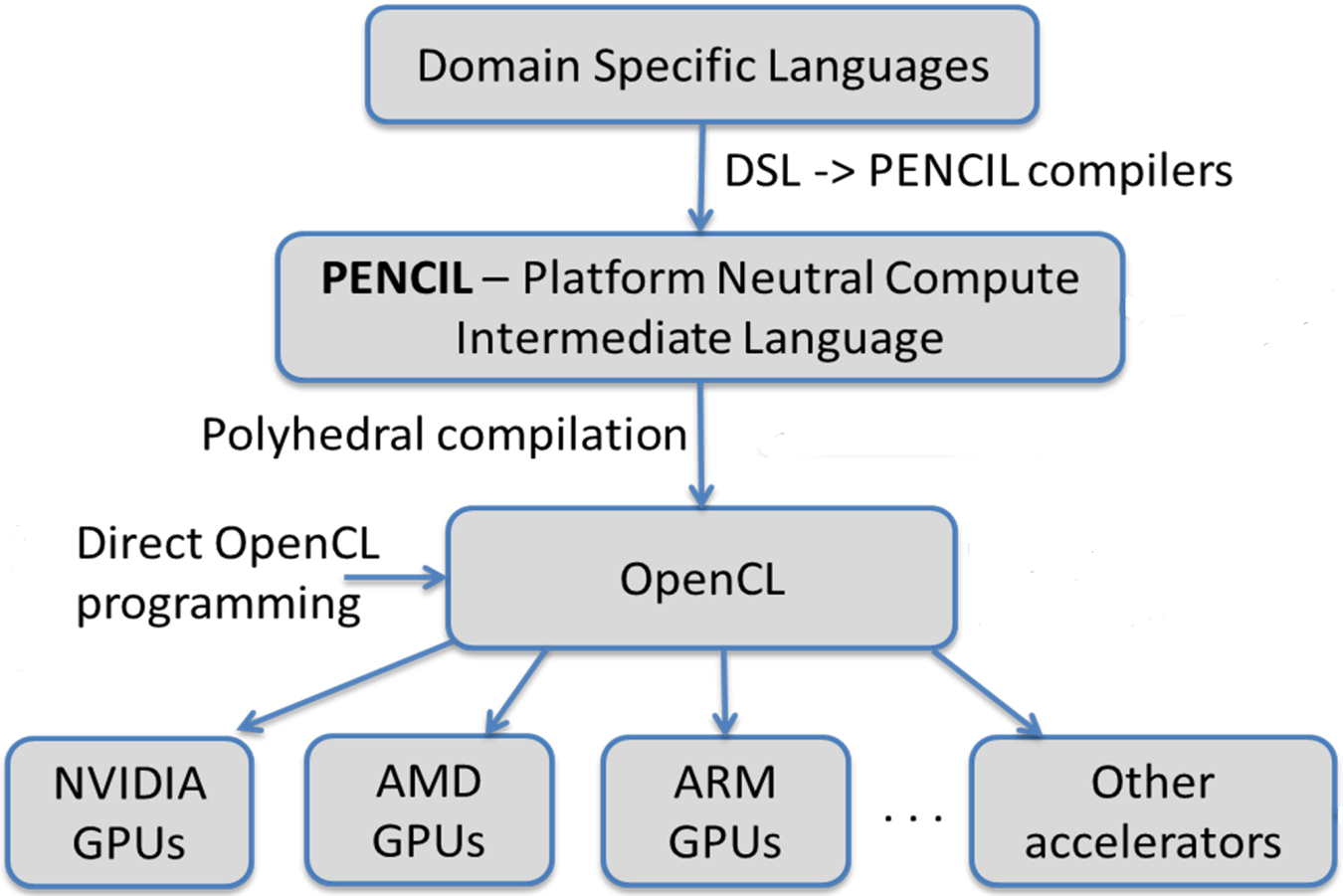
PENCIL: a Platform-Neutral Compute Intermediate Language for DSLs

- An intermediate language for DSL compilers
- C based intermediate language
- A set of coding rules, language extensions and directives on top of C

Design goals

- Unlock the power of optimization frameworks by
  - keeping a maximum of information expressed by the DSL
  - eliminating ambiguity for optimizers
- Users: Code generators + expert developers

How to use PENCIL



More about PENCIL

- An equivalent LLVM IR will be provided
- Platform neutral
- Only computation intensive code regions need to be PENCIL-compliant
- PENCIL does not compete with other DSL ILs such as Delite IR; they are complementary
- The runtime system schedules the kernels

Platform-Neutral Compute Intermediate Language

- Coding rules
- Extensions
- Directives

Platform-Neutral Compute Intermediate Language

- Coding rules
  - no pointers aside from function arguments
  - pointer arguments should be declared with `restrict const`
  - no recursion
  - no unstructured control flow (no `gotos`)
- Extensions
- Directives

Platform-Neutral Compute Intermediate Language

- Coding rules
  - no pointers aside from function arguments
  - pointer arguments should be declared with `restrict const`
  - no recursion
  - no unstructured control flow (no `gotos`)
- Extensions
  - access summary functions
    - describe access pattern of a function if automatic analysis cannot be performed (no source or not PENCIL compliant) or if the results are too inaccurate
    - information used in calling function
- Directives

Platform-Neutral Compute Intermediate Language

- Coding rules
  - no pointers aside from function arguments
  - pointer arguments should be declared with `restrict const`
  - no recursion
  - no unstructured control flow (no `gotos`)
- Extensions
  - access summary functions
    - describe access pattern of a function if automatic analysis cannot be performed (no source or not PENCIL compliant) or if the results are too inaccurate
    - information used in calling function
- Directives
  - `#pragma pencil independent` $[(l_1, \ldots, l_n)]$

    listed statements (all if unspecified) do not carry any dependences across the loop following the directive

Example of PENCIL code

```
int function(int A[const restrict 100][100],
             int C[const restrict 100][100]) {
  #pragma pencil independent
  for (int k = 0; k < N; k++)
    for (int j = 0; j < N; j++)
      A[k][t[j]] = foo(C);
}
```

Example of PENCIL code

```
void foo_summary(int C[const restrict n][n]) {
     USE(C);
}

void foo(int C[const restrict n][n])
     ACCESS(foo_summary(C));

int function(int A[const restrict 100][100],
             int C[const restrict 100][100]) {
  #pragma pencil independent
  for (int k = 0; k < N; k++)
    for (int j = 0; j < N; j++)
      A[k][t[j]] = foo(C);
}
```

PPCG, an Example of an Optimization Framework

PPCG (`http://freecode.com/projects/ppcg`)

- Input: C (PENCIL to be implemented)
- Output:
  - CUDA
  - OpenMP and OpenCL (soon)

PPCG, an Example of an Optimization Framework

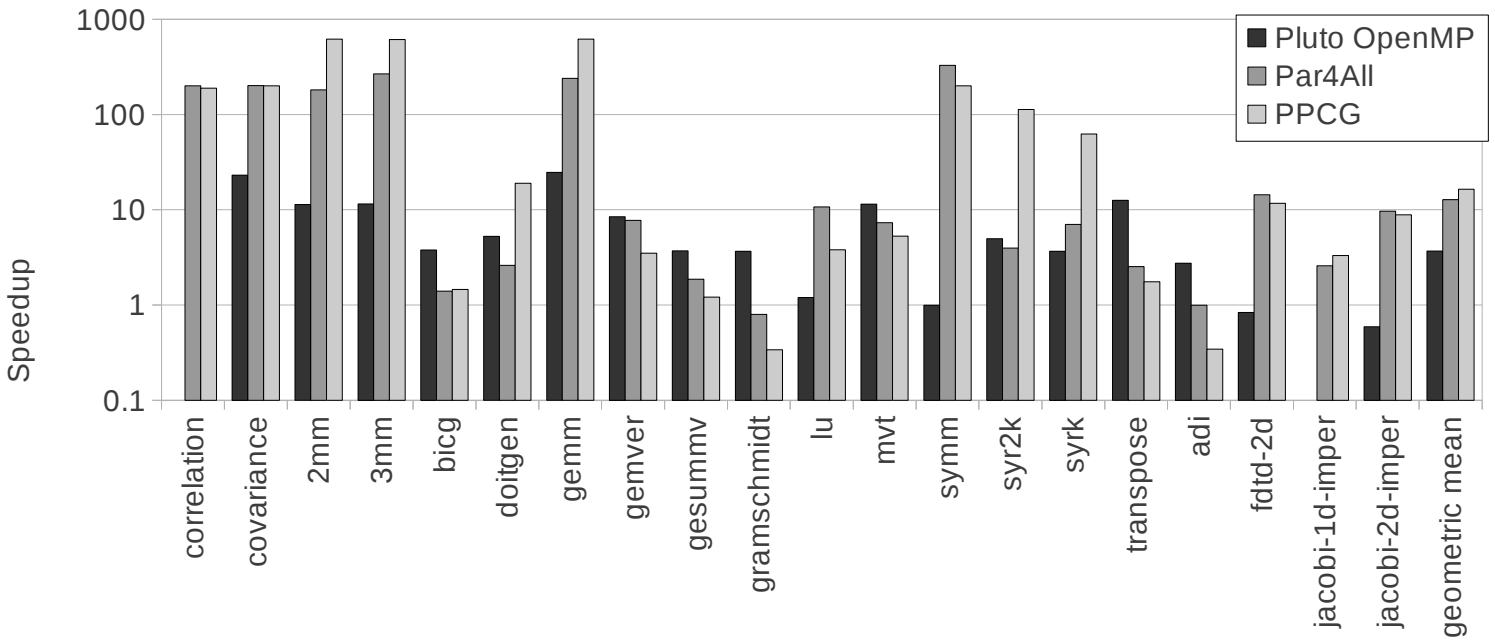
PPCG (`http://freecode.com/projects/ppcg`)

- Input: C (PENCIL to be implemented)
- Output:
  - CUDA
  - OpenMP and OpenCL (soon)

Steps:

- Extract polyhedral model from PENCIL code
- Dependence analysis
- Scheduling
  - Expose parallelism and tiling opportunities
  - Separate schedule into parts mapped on host and GPU
- Memory management
  - Add transfers of data to/from GPU
- Generate AST

PPCG Results



- Experiments performed by Juan Carlos Juega
- Benchmarks: PolyBench
- Platform: Tesla M2070
- Baseline: sequential CPU execution

Summary

- Work in progress: PENCIL an IL for DSLs
  - C based
  - no pointers
  - summary access functions
  - independent pragma
- Provide an optimization framework (polyhedral optimization)