


PENCIL

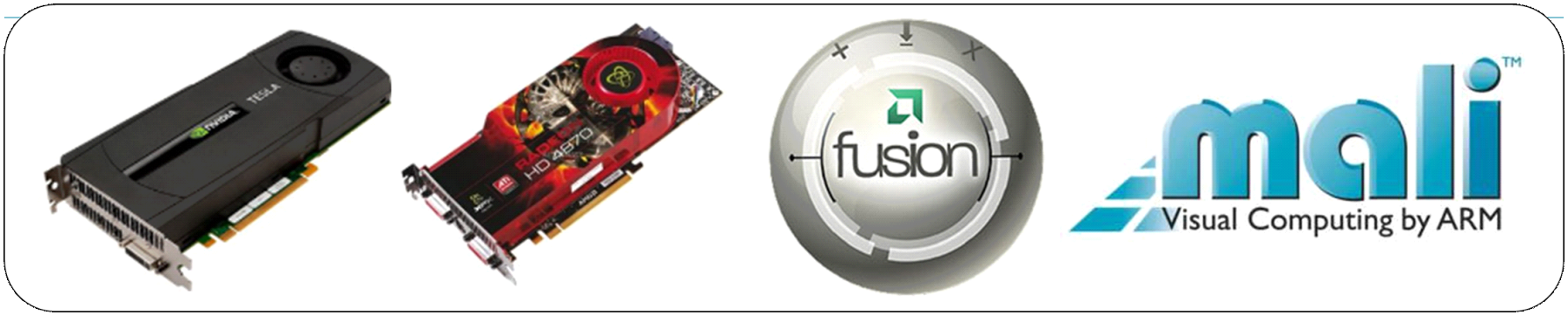
A Platform-Neutral Compute Intermediate Language for DSL Compilers

Riyadh Baghdadi[1,2] Javed Absar[3] Ulysse Beaugnon[1,2]
Adam Betts[4] Albert Cohen[1,2] Róbert Dávid[5]
Alastair Donaldson[4] Tobias Grosser[1,2] Sven Van Haastregt[3]
Elnar Hajiyev[5] Jeroen Ketema[4] Alexey Kravets[3]
Michael Kruse[1] Anton Lokhmotov[3] Chandan Reddy[1]
Sven Verdoolaege[2,6]

[1]École Normale Supérieure, [2]INRIA, [3]ARM, [4]Imperial College London, [5]RealEyes, [6]KU Leuven

9th February 2016

Obligatory Motivational Slide About Accelerators



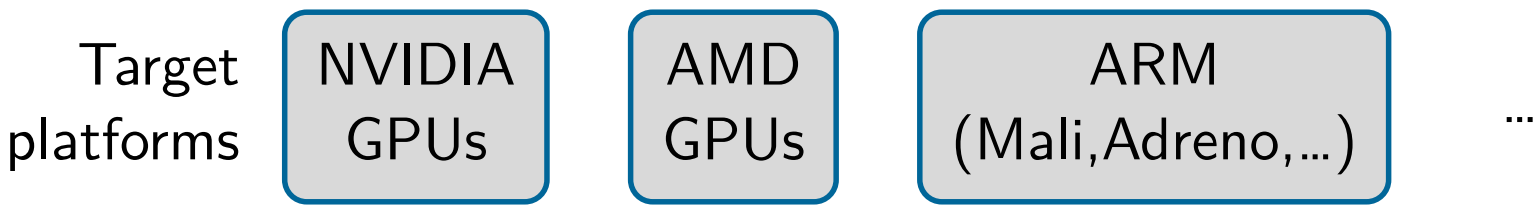
Advantages

- Impressive raw performance
- Massive parallelism
- Low energy consumption per operation

Problems

- Highly optimized code is hard to write
- Non-portable performance
- Maintaining multiple sources for different architectures not easy

Compilers



Compilers

Domain languages

VOBLA

```
function gemm(alpha: Value,
              in A: SparseIterable<Value>[m][k],
              in B: Value[k][n],
              beta: Value,
              out C: Value[m][n])
{
    Cij *= beta forall _, _, Cij in C.sparse;

    C[i][j] += alpha*Ail*B[l][j]
        for i, l, Ail in A.sparse, j in 0:n-1;
}
```

Target platforms

NVIDIA GPUs

AMD GPUs

ARM (Mali,Adreno,...)

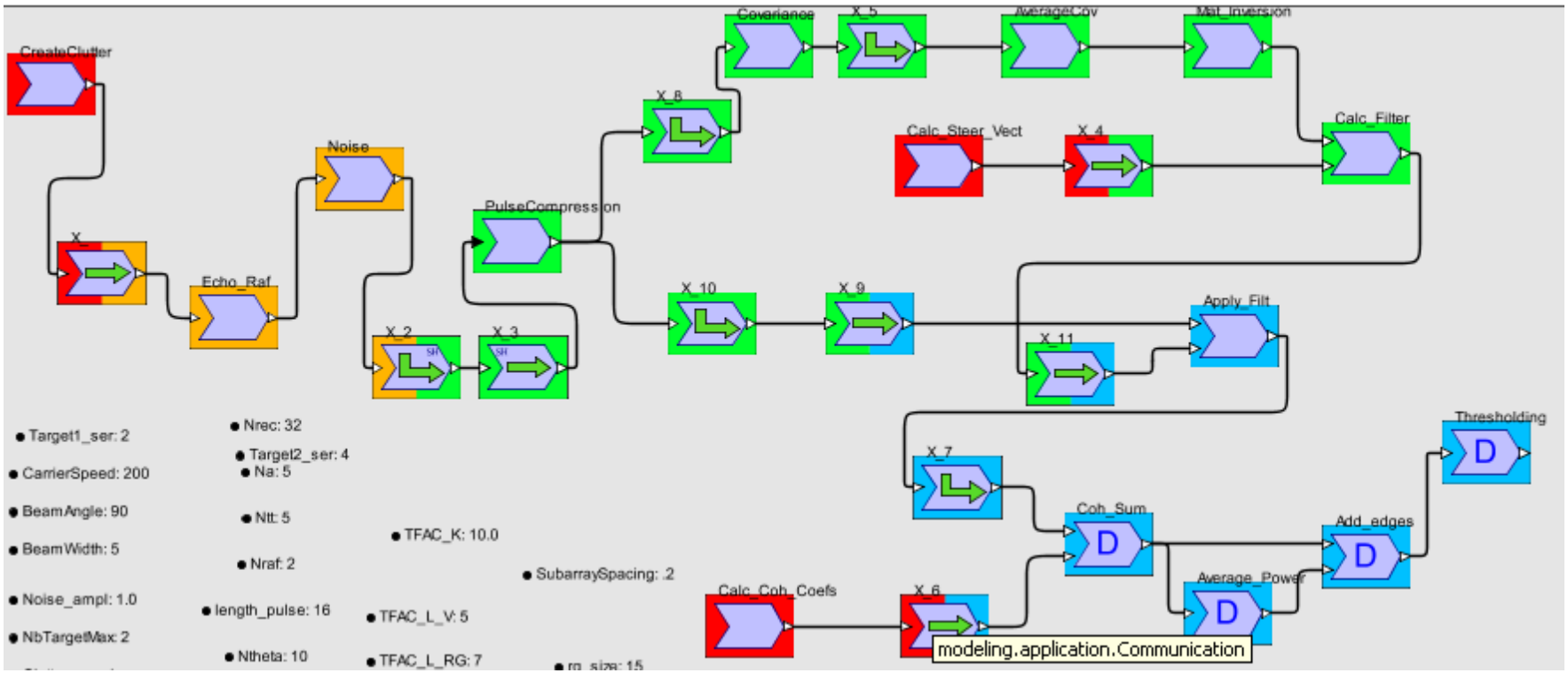
...

Compilers

Domain languages

VOBLA

SpearDE



Target platforms

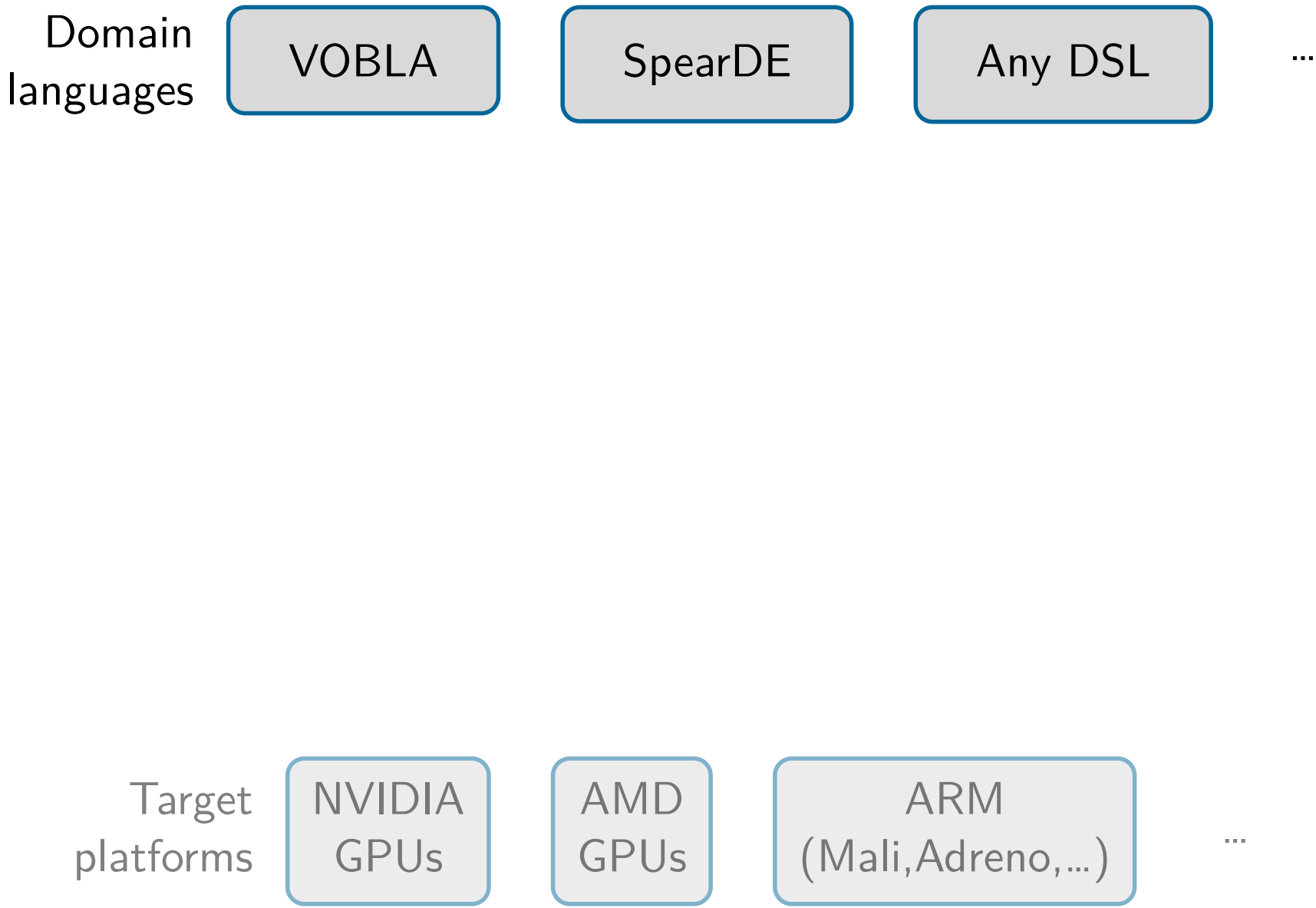
NVIDIA GPUs

AMD GPUs

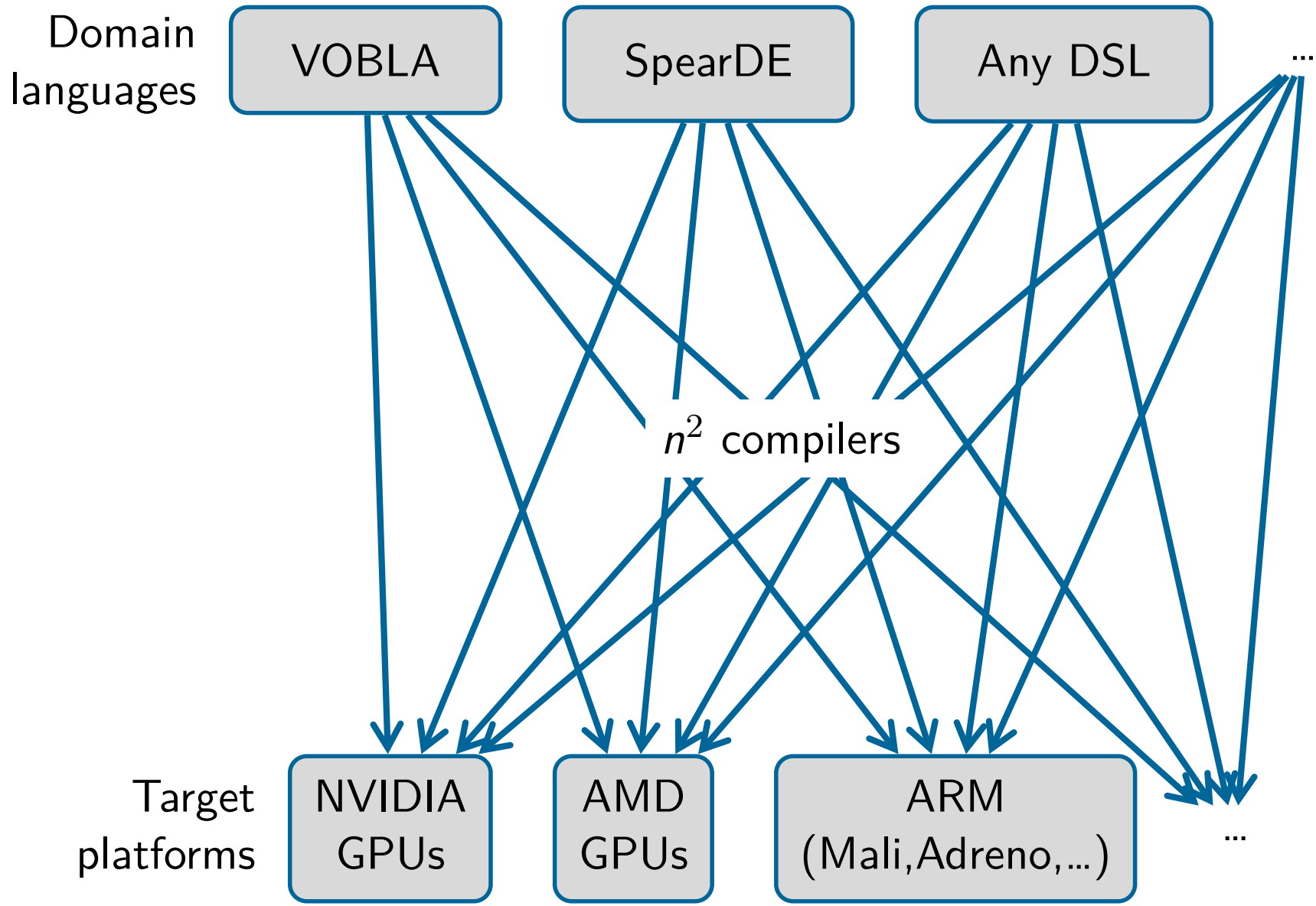
ARM (Mali,Adreno,...)

...

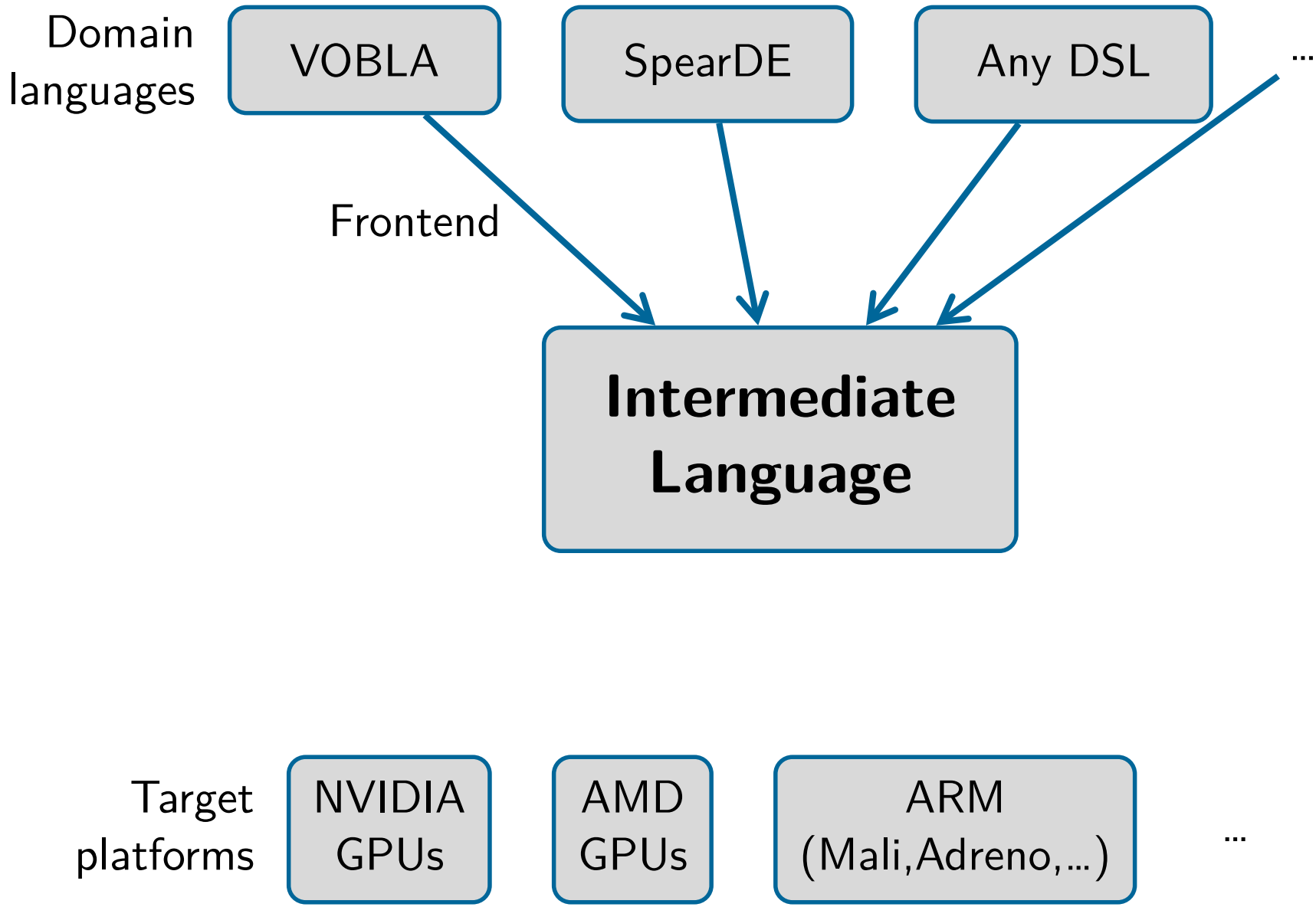
Compilers


Domain languages
VOBLA
SpearDE
Any DSL
...
Target platforms
NVIDIA GPUs
AMD GPUs
ARM (Mali,Adreno,...)
...

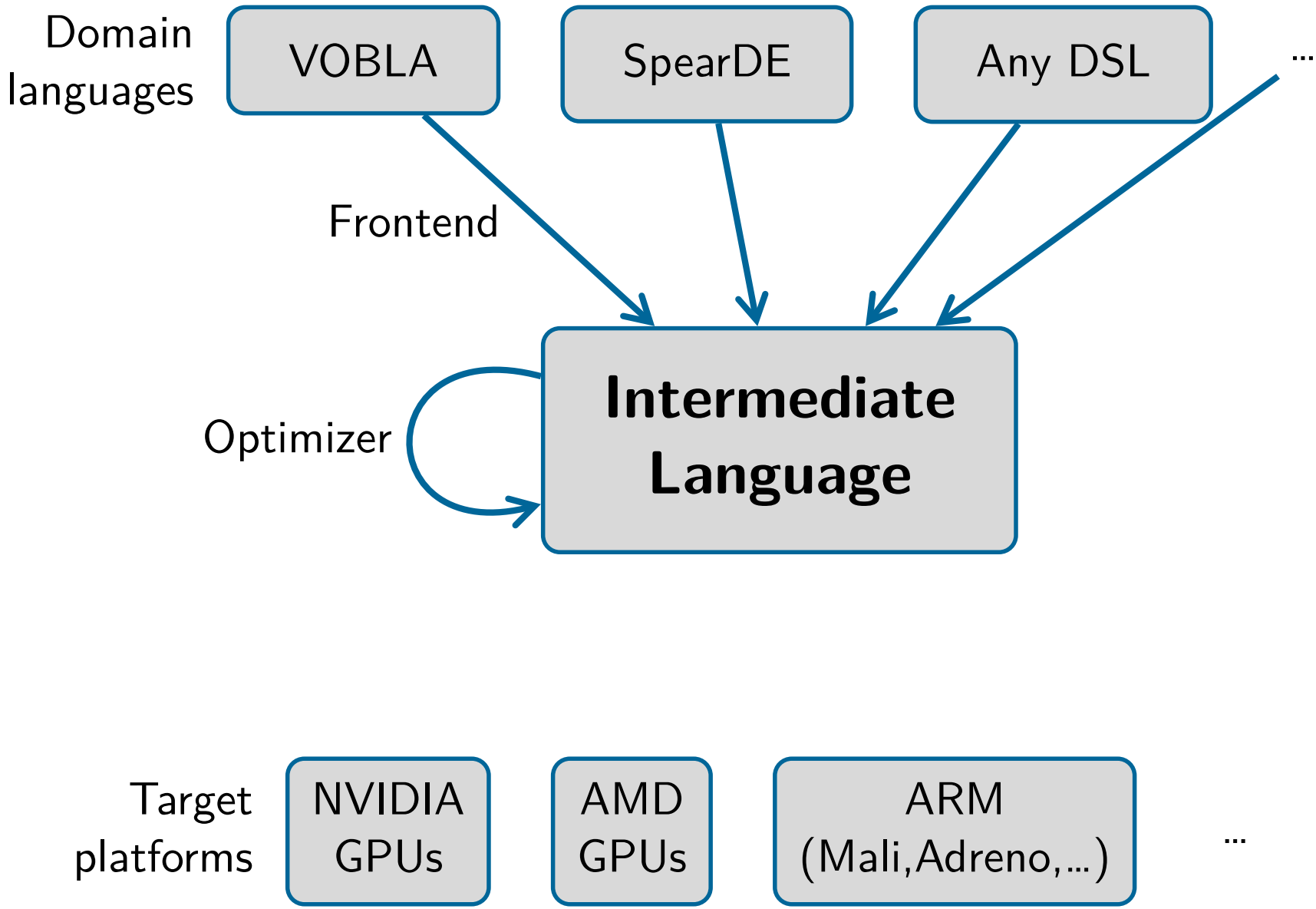
Compilers


Domain languages
VOBLA
SpearDE
Any DSL
...
n^2 compilers
Target platforms
NVIDIA GPUs
AMD GPUs
ARM (Mali,Adreno,...)
...

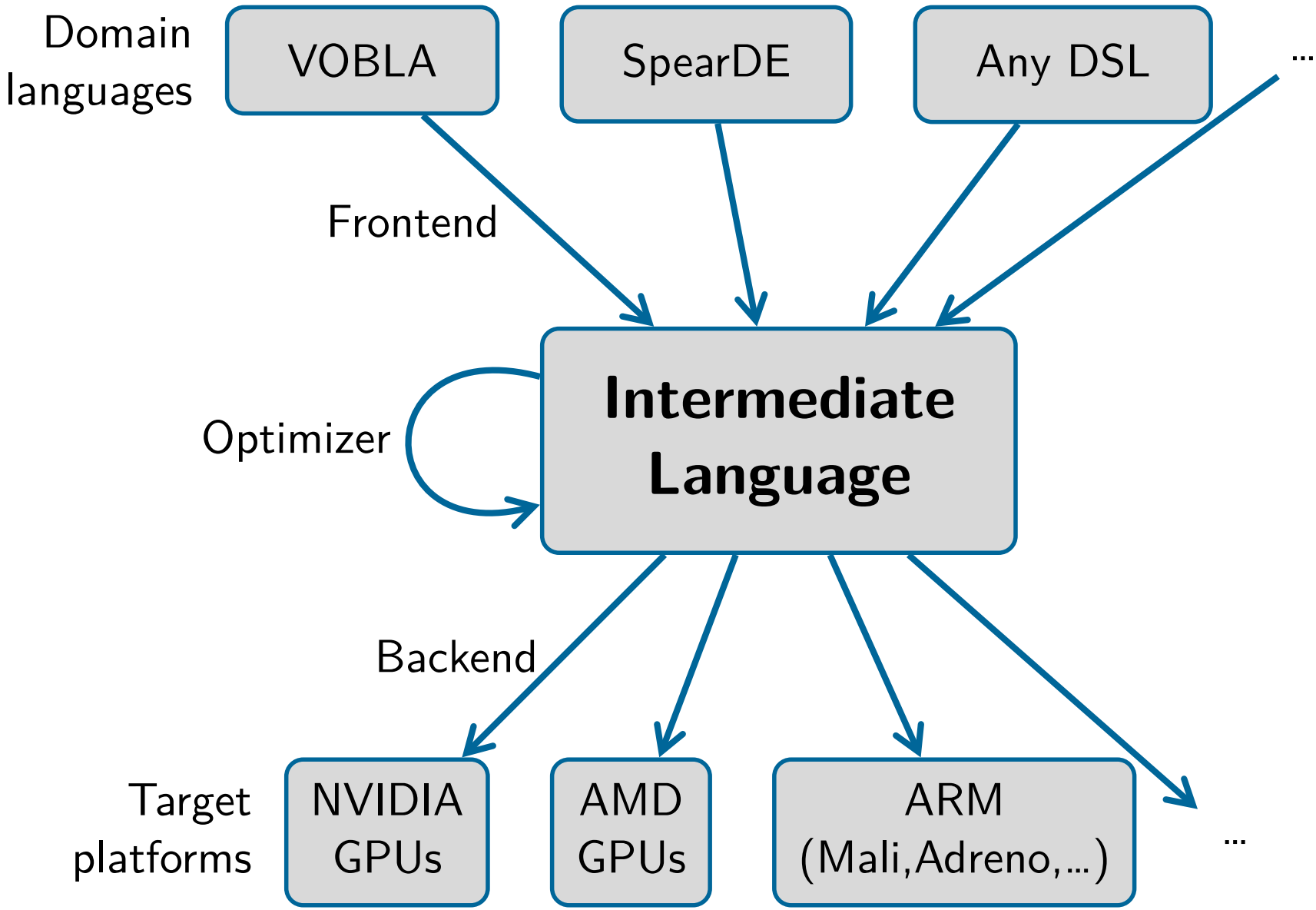
Compilers


Domain languages
VOBLA
SpearDE
Any DSL
...
Frontend
Intermediate Language
Target platforms
NVIDIA GPUs
AMD GPUs
ARM (Mali,Adreno,...)
...

Compilers


Domain languages
VOBLA
SpearDE
Any DSL
...
Frontend
Optimizer
Intermediate Language
Target platforms
NVIDIA GPUs
AMD GPUs
ARM (Mali,Adreno,...)
...

Compilers


Domain languages
VOBLA
SpearDE
Any DSL
...
Frontend
Optimizer
Intermediate Language
Backend
Target platforms
NVIDIA GPUs
AMD GPUs
ARM (Mali,Adreno,...)
...

Overview

Ideas

Ideas

- Intermediate language
  Any frontend language possible

Ideas

- Intermediate language
  Any frontend language possible
- Polyhedral compiler friendly
  Powerful static analysis and optimizations

Ideas

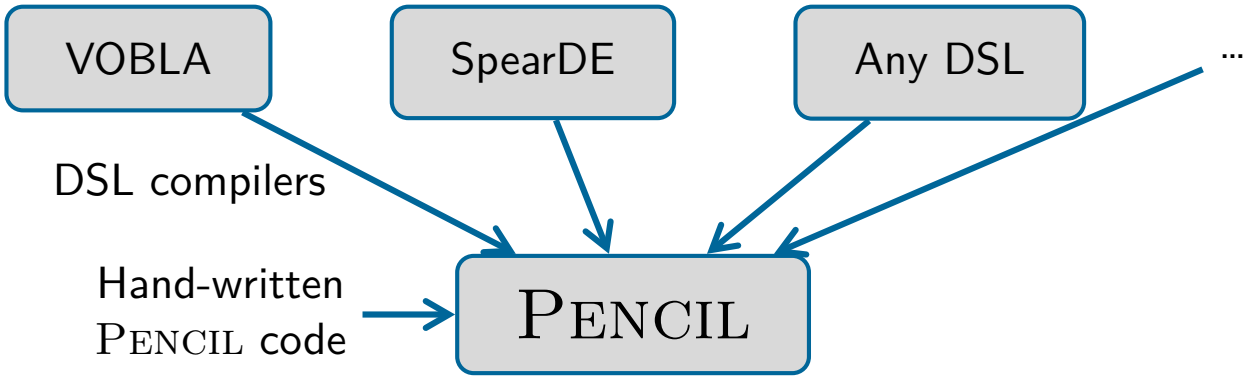
- Intermediate language
  Any frontend language possible
- Polyhedral compiler friendly
  Powerful static analysis and optimizations
- Meta-level specification
  Code properties not derivable from code itself (helps pessimistic optimizers)

Ideas

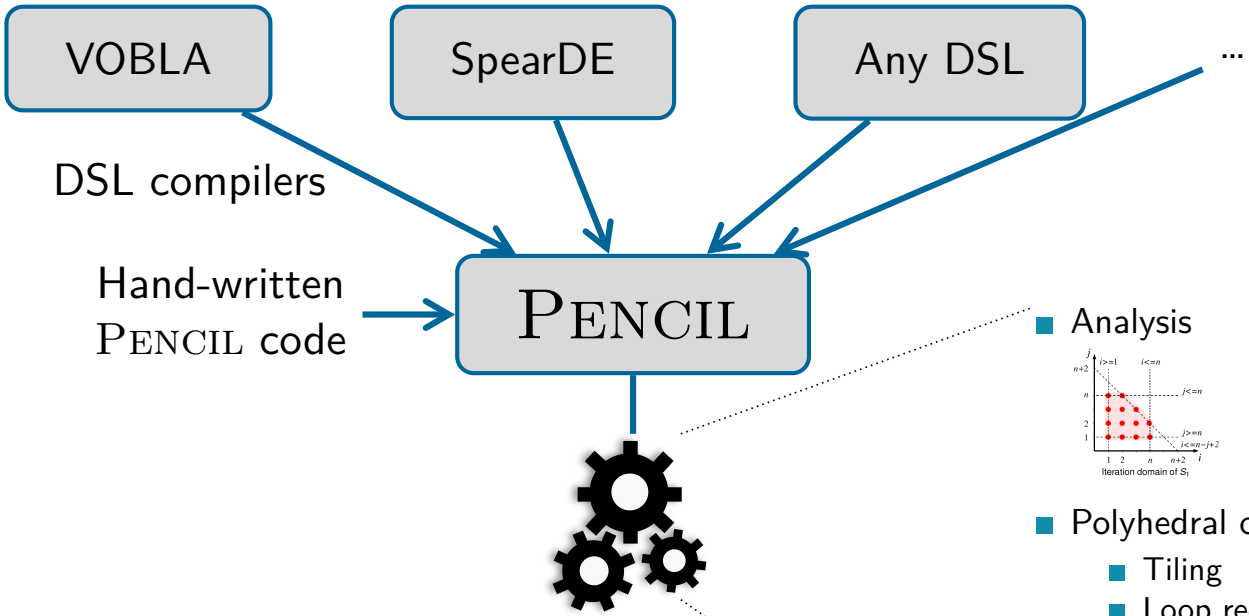
- Intermediate language
  Any frontend language possible
- Polyhedral compiler friendly
  Powerful static analysis and optimizations
- Meta-level specification
  Code properties not derivable from code itself (helps pessimistic optimizers)
- OpenCL output
  Multiple target platforms

General Flow

General Flow

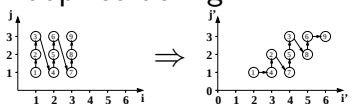


General Flow



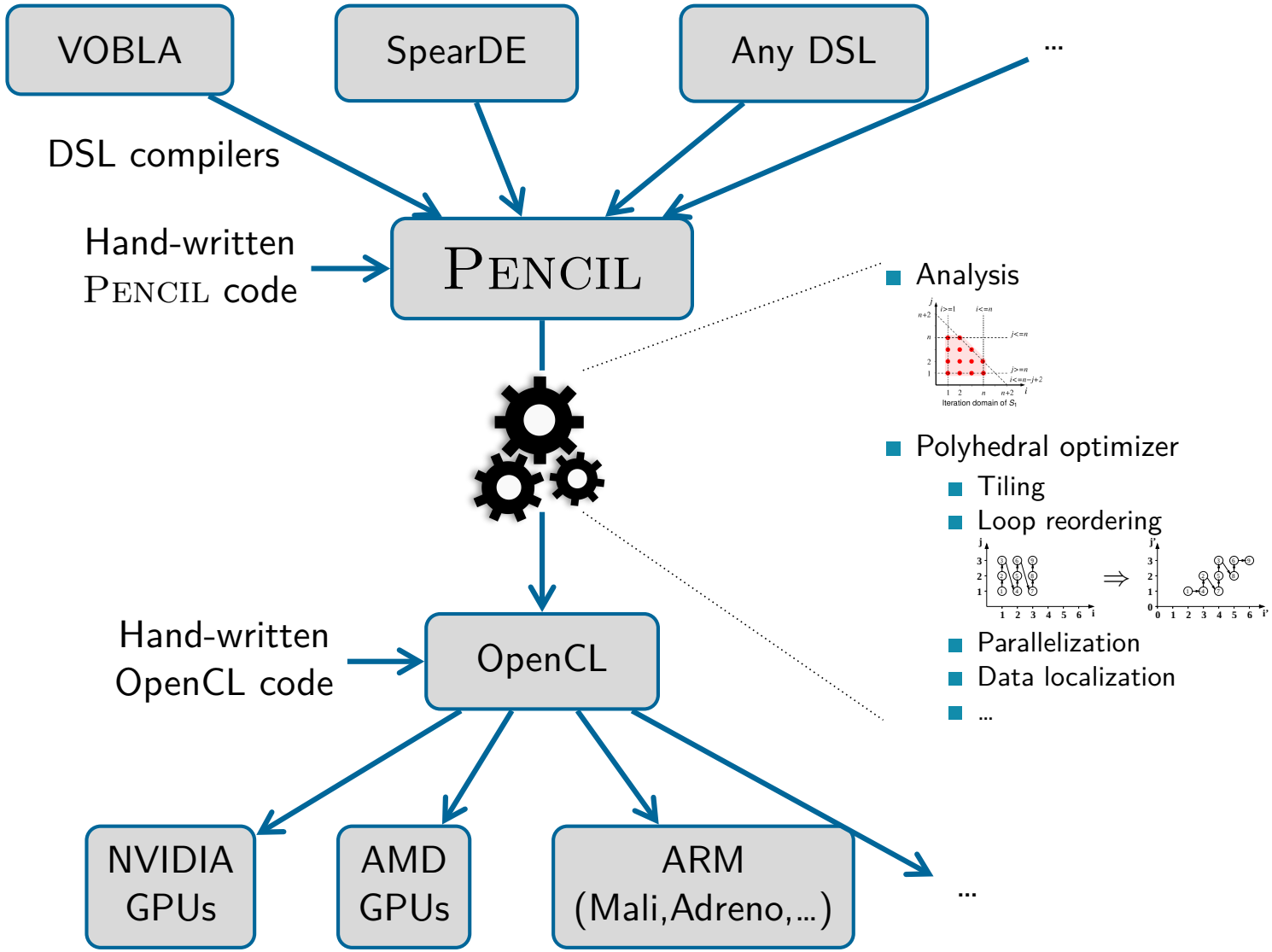
- Analysis

- Polyhedral optimizer
  - Tiling
  - Loop reordering

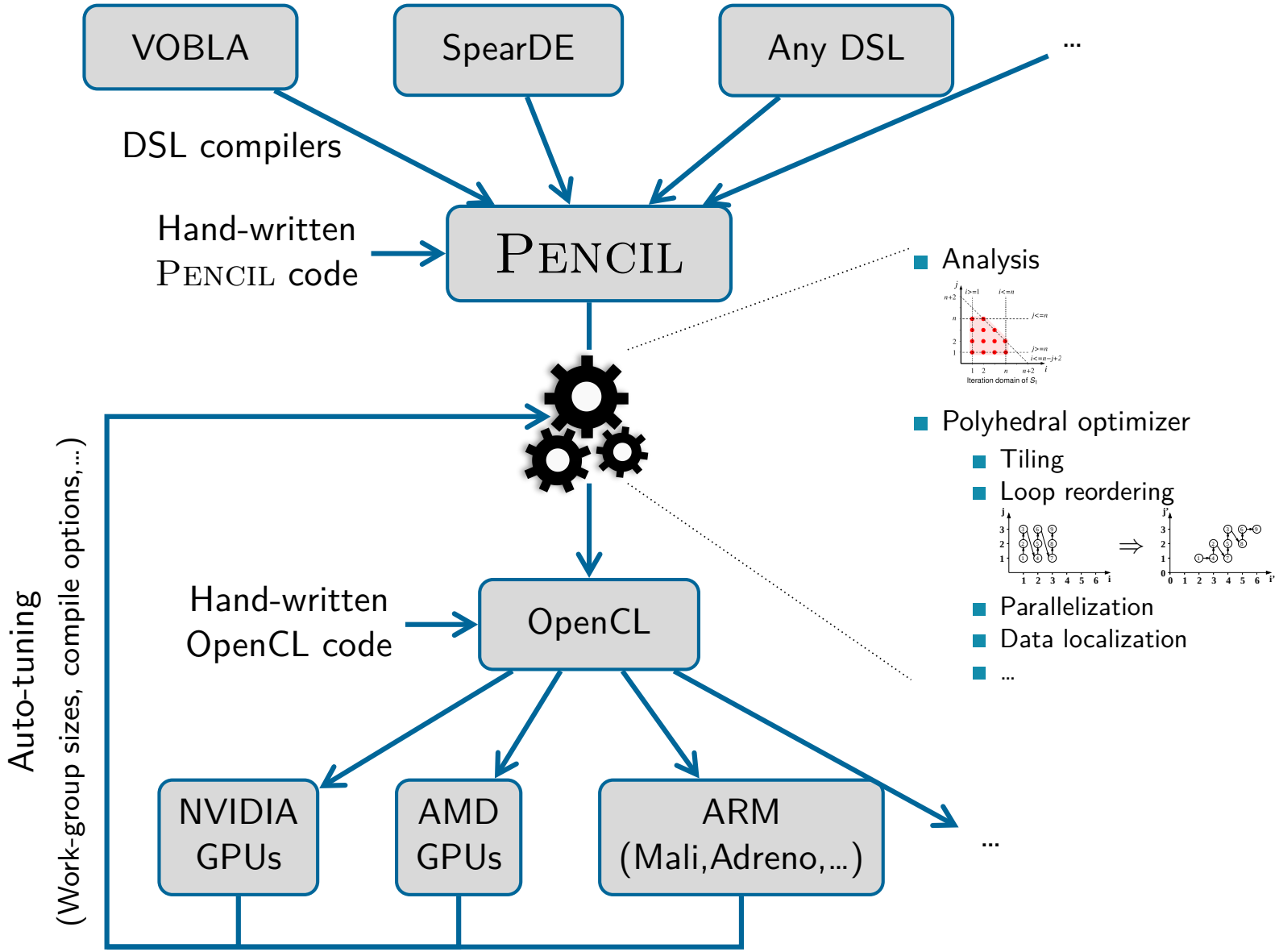


  - Parallelization
  - Data localization
  - ...

General Flow



General Flow



Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts
  - C99 VLA syntax for array parameters

```
void foo(int * const restrict A)
```

$\Rightarrow$

```
void foo(int n, int A[static const restrict n])
```

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts
  - C99 VLA syntax for array parameters

    ```
    void foo(int * const restrict A)
    ```

    $\Rightarrow$

    ```
    void foo(int n, int A[static const restrict n])
    ```
- Optimization hints

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts
  - C99 VLA syntax for array parameters

    ```
    void foo(int * const restrict A)
    ```

    $\Rightarrow$

    ```
    void foo(int n, int A[static const restrict n])
    ```
- Optimization hints
  - `__pencil_kill`
    Invalidate data currently helt by an array

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts
  - C99 VLA syntax for array parameters

    ```
    void foo(int * const restrict A)
    ```

    $\Rightarrow$

    ```
    void foo(int n, int A[static const restrict n])
    ```
- Optimization hints
  - `__pencil_kill`
    Invalidate data currently helt by an array
  - `#pragma pencil independent`
    The result does not depend on the execution order of iterations

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts
  - C99 VLA syntax for array parameters

    ```
    void foo(int * const restrict A)
    ```

    $\Rightarrow$

    ```
    void foo(int n, int A[static const restrict n])
    ```
- Optimization hints
  - `__pencil_kill`
    Invalidate data currently helt by an array
  - `#pragma pencil independent`
    The result does not depend on the execution order of iterations
  - `__pencil_assume`
    Make assumption violations undefined behviour

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - Context-free grammar
  - No `GOTO`s, no recursion
  - Pointers generally forbidden
    only allowed in decay-to-pointer with subscripts
  - C99 VLA syntax for array parameters

    ```
    void foo(int * const restrict A)
    ```

    $\Rightarrow$

    ```
    void foo(int n, int A[static const restrict n])
    ```
- Optimization hints
  - `__pencil_kill`
    Invalidate data currently helt by an array
  - `#pragma pencil independent`
    The result does not depend on the execution order of iterations
  - `__pencil_assume`
    Make assumption violations undefined behviour
  - Summary functions
    Describe the memory access pattern of a function

General Syntax

Example: Finite Impulse Response

```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4           float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```

General Syntax

Example: Finite Impulse Response

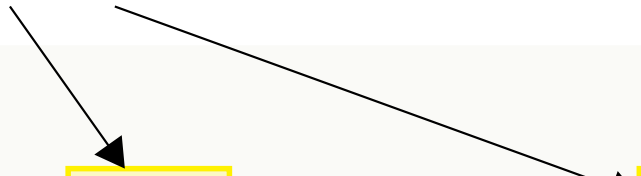
Declarations & plain C compatibility

```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4           float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```

General Syntax

Example: Finite Impulse Response

At least `n` elements (and non-null if $n > 0$)

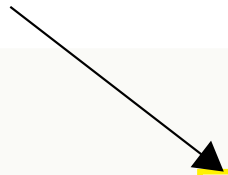


```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4          float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```

General Syntax

Example: Finite Impulse Response

Pointer `in` does not change in function body



```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4           float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```

General Syntax

Example: Finite Impulse Response

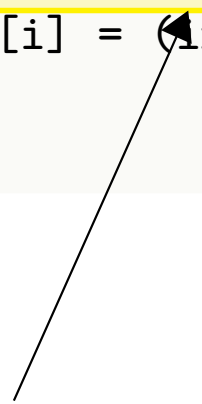
No aliasing

```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4           float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```

General Syntax

Example: Finite Impulse Response

```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4           float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```

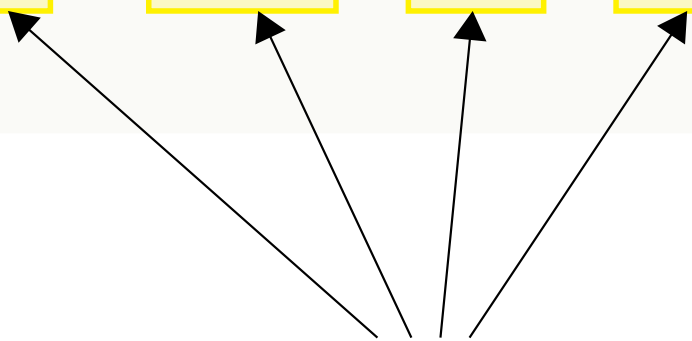


Canonical `for`-loop

General Syntax

Example: Finite Impulse Response

```
1 #include <pencil.h>
2
3 void fir(int n, float in[static const restrict n],
4           float out[static const restrict n])
5 {
6     for (int i = 1; i < n-1; ++i) {
7         out[i] = (in[i-1] + in[i] + in[i+1])/3;
8     }
9 }
```



Array subscripts (syntactically different from `*(out + i)`)

Kill Statement

Example: Finite Impulse Response

```
1  void doublefir_inplace(int n, float in[static const restrict n],
2                         float tmp[static const restrict n],
3                         float out[static const restrict n])
4  {
5
6
7
8      for (int i = 1; i < n-1; ++i) {
9          tmp[i] = (in[i-1] + in[i] + in[i+1])/3;
10     }
11     tmp[0] = (in[0] + in[1])/2;
12     tmp[n-1] = (in[n-2] + in[n-1])/2;
13
14     for (int i = 1; i < n-1; ++i) {
15         out[i] = (tmp[i-1] + tmp[i] + tmp[i+1])/3;
16     }
17
18
19 }
```

Kill Statement

Example: Finite Impulse Response

```
1  void doublefir_inplace(int n, float in[static const restrict n],
2                         float tmp[static const restrict n],
3                         float out[static const restrict n])
4  {
5
6
7
8      for (int i = 1; i < n-1; ++i) {
9          tmp[i] = (in[i-1] + in[i] + in[i+1])/3;
10     }
11     tmp[0] = (in[0] + in[1])/2;
12     tmp[n-1] = (in[n-2] + in[n-1])/2;
13
14     for (int i = 1; i < n-1; ++i) {
15         out[i] = (tmp[i-1] + tmp[i] + tmp[i+1])/3;
16     }
17
18     __pencil_kill(tmp); // Avoid copy to host
19 }
```

Kill Statement

Example: Finite Impulse Response

```
1  void doublefir_inplace(int n, float in[static const restrict n],
2                         float tmp[static const restrict n],
3                         float out[static const restrict n])
4  {
5      __pencil_kill(tmp); // Avoid copy to device
6      __pencil_kill(out); // Avoid copy to device
7
8      for (int i = 1; i < n-1; ++i) {
9          tmp[i] = (in[i-1] + in[i] + in[i+1])/3;
10     }
11     tmp[0] = (in[0] + in[1])/2;
12     tmp[n-1] = (in[n-2] + in[n-1])/2;
13
14     for (int i = 1; i < n-1; ++i) {
15         out[i] = (tmp[i-1] + tmp[i] + tmp[i+1])/3;
16     }
17
18     __pencil_kill(tmp); // Avoid copy to host
19 }
```

Independent Directive

Example: Basic Histogram

```
1  void basic_histogram(int n, int m,
2                       unsigned char image[static const restrict n][m],
3                       int hist[static const restrict 256]) {
4
5      for (int i=0; i<N; i++)
6
7          for (int j=0; j<M; j++) {
8              int idx = image[i][j];
9              atomic_add(hist[idx], 1);
10         }
11 }
```

Independent Directive

Example: Basic Histogram

```
1  void basic_histogram(int n, int m,
2                       unsigned char image[static const restrict n][m],
3                       int hist[static const restrict 256]) {
4  #pragma pencil independent
5      for (int i=0; i<N; i++)
6  #pragma pencil independent
7          for (int j=0; j<M; j++) {
8              int idx = image[i][j];
9              atomic_add(hist[idx], 1);
10         }
11 }
```

Assume Statement

Example: Matrix-Vector Multiplication

```
1  void gemv(int n, int m, float mat[static const restrict m][n],
2            float vec[static const restrict n],
3            float out[static const restrict m]) {
4
5
6
7      for (int i = 0; i < m; ++i) {
8          out[i] = 0;
9          for (int j = 1; j < n; ++j)
10             out[i] += mat[i][j] * vec[j];
11     }
12 }
```

Assume Statement

Example: Matrix-Vector Multiplication

```
1  void gemv(int n, int m, float mat[static const restrict m][n],
2            float vec[static const restrict n],
3            float out[static const restrict m]) {
4      __pencil_assume(n > 0 && m > 0);
5
6
7      for (int i = 0; i < m; ++i) {
8          out[i] = 0;
9          for (int j = 1; j < n; ++j)
10             out[i] += mat[i][j] * vec[j];
11     }
12 }
```

Assume Statement

Example: Matrix-Vector Multiplication

```
1  void gemv(int n, int m, float mat[static const restrict m][n],
2            float vec[static const restrict n],
3            float out[static const restrict m]) {
4      __pencil_assume(n > 0 && m > 0);
5
6
7      for (int i = 0; i < m; ++i) {
8          out[i] = 0;
9          for (int j = 1; j < n; ++j)
10             out[i] += mat[i][j] * vec[j];
11     }
12 }
```

- Avoid code generation and checking for special cases

Assume Statement

Example: Matrix-Vector Multiplication

```
1  void gemv(int n, int m, float mat[static const restrict m][n],
2            float vec[static const restrict n],
3            float out[static const restrict m]) {
4      __pencil_assume(n > 0 && m > 0);
5      __pencil_assume(n <= 16);
6
7      for (int i = 0; i < m; ++i) {
8          out[i] = 0;
9          for (int j = 1; j < n; ++j)
10             out[i] += mat[i][j] * vec[j];
11     }
12 }
```

Assume Statement

Example: Matrix-Vector Multiplication

```
1  void gemv(int n, int m, float mat[static const restrict m][n],
2            float vec[static const restrict n],
3            float out[static const restrict m]) {
4      __pencil_assume(n > 0 && m > 0);
5      __pencil_assume(n <= 16);
6
7      for (int i = 0; i < m; ++i) {
8          out[i] = 0;
9          for (int j = 1; j < n; ++j)
10             out[i] += mat[i][j] * vec[j];
11     }
12 }
```

- Promote vec to local memory

Summary Function

```
1
2
3
4
5
6
7
8  void blackbox(int n, float in[static const restrict n],
9      float out[static const restrict n]);
10
11 void foo(int m, int n, float in[static const restrict n],
12         float out[static const restrict m][n]) {
13     for (int j=0; j<m; ++j)
14         blackbox(n, in, out[j]);
15 }
```

Summary Function

```
1  static void blackbox_summary(int n, float in[static const restrict n],
2                               float out[static const restrict n]) {
3      for (int i=0; i<n; ++i)
4          USE(in[i]);
5      for (int i=0; i<n; ++i)
6          DEF(out[i]);
7  }
8  void blackbox(int n, float in[static const restrict n],
9       float out[static const restrict n]) ACCESS(blackbox_summary);
10
11 void foo(int m, int n, float in[static const restrict n],
12          float out[static const restrict m][n]) {
13     for (int j=0; j<m; ++j)
14         blackbox(n, in, out[j]);
15 }
```

Summary Functions

Example: Fast Fourier Transformation

```
1  static void fftKernel32_summary(int n, int start, struct float2 a[static const restrict n]) {
2      for (int i=start; i<start+32; ++i)
3          USE(a[i]);
4      for (int i=start; i<start+32; ++i)
5          DEF(a[i]);
6  }
7
8  void fftKernel32(int n, int start, struct float2 a[static const restrict n])
9          ACCESS(fftKernel32_summary);
10
11 void fft64(struct float2 a[static const restrict 64]) {
12     for (int k = 0; k < 2; ++k) {
13         fftKernel32(64, k*32, a);
14     }
15     for (int k = 0; k < 32; ++k) {
16         float2 t = a[k];
17         float2 s = a[32+k];
18
19         a[k].x = s.x + cos(-2*PI*k/64)*t.x - sin(-2*PI*k/64)*t.y;
20         a[k].y = s.y + sin(-2*PI*k/64)*t.x + cos(-2*PI*k/64)*t.x;
21         a[32+k].x = s.x - cos(-2*PI*k/64)*t.x + sin(-2*PI*k/64)*t.y;
22         a[32+k].y = s.y - sin(-2*PI*k/64)*t.x - cos(-2*PI*k/64)*t.x;
23     }
24 }
```

Summary/Conclusion

- PENCIL language
  - Polyhedral-friendly subset of C99
  - `__pencil_kill(expr)`
  - `__pencil_assume(expr)`
  - `#pragma pencil independent`
  - Summary functions
- Versatile intermediate step
- Competitive performance using PPCG
- PPCG also compiles to CUDA and OpenMP

References

- PPCG
  http://freecode.com/projects/ppcg
- VOBLA
  Beaugnon et. al. *VOBLA: A vehicle for optimized basic linear algebra*. LCTES '14
- SpearDE
  Lenormand and Edelin. *An industrial perspective: A pragmatic high end signal processing design environment at Thales*. SAMOS '03

Compile and Install

Requirements:

- gcc/clang
- GNU Autotools
- clang-dev
- Python 3
- OpenCL/nvcc in system default paths

```
$ git clone https://github.com/Meinersbur/pencilcc.git --recursive
$ cd pencilcc
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Usage I

- Show command options

```
$ pencilcc -h
$ ppcg -h
```

Usage I

- Show command options

```
$ pencilcc -h
$ ppcg -h
```

- Basic for C99 with PENCIL-extensions

```
$ pencilcc source.c -o program
```

Usage I

- Show command options

```
$ pencilcc -h
$ ppcg -h
```

- Basic for C99 with PENCIL-extensions

```
$ pencilcc source.c -o program
```

- Basic for PENCIL-only files (`#pragma scop`/`endscop` not required)

```
$ pencilcc main.c source.pencil.c -o program
```

Usage I

- Show command options

```
$ pencilcc -h
$ ppcg -h
```

- Basic for C99 with PENCIL-extensions

```
$ pencilcc source.c -o program
```

- Basic for PENCIL-only files (`#pragma scop`/`endscop` not required)

```
$ pencilcc main.c source.pencil.c -o program
```

- Compile to CUDA

```
$ pencilcc source.c -o program --target=cuda
```

Usage I

- Show command options

```
$ pencilcc -h
$ ppcg -h
```

- Basic for C99 with PENCIL-extensions

```
$ pencilcc source.c -o program
```

- Basic for PENCIL-only files (`#pragma scop`/`endscop` not required)

```
$ pencilcc main.c source.pencil.c -o program
```

- Compile to CUDA

```
$ pencilcc source.c -o program --target=cuda
```

- Show what it does

```
$ pencilcc source.c -o program --show-commands
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

- Dump effective optimization parameters

```
$ pencilcc source.c -o program --dump-sizes
{kernel[0]->tile[32];kernel[0]->block[64];kernel[0]->grid[1024]}
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

- Dump effective optimization parameters

```
$ pencilcc source.c -o program --dump-sizes
{kernel[0]->tile[32];kernel[0]->block[64];kernel[0]->grid[1024]}
```

- Change optimization parameters

```
$ pencilcc source.c -o program \
"--sizes={kernel[i]->tile[32];kernel[i]->block[64];kernel[i]->grid[1024]}"
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

- Dump effective optimization parameters

```
$ pencilcc source.c -o program --dump-sizes
{kernel[0]->tile[32];kernel[0]->block[64];kernel[0]->grid[1024]}
```

- Change optimization parameters

```
$ pencilcc source.c -o program \
"--sizes={kernel[i]->tile[32];kernel[i]->block[64];kernel[i]->grid[1024]}"
```

- Add hand-written OpenCL code

```
$ pencilcc source.c -o program --opencl-include-file=/path/to/file.inc
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

- Dump effective optimization parameters

```
$ pencilcc source.c -o program --dump-sizes
{kernel[0]->tile[32];kernel[0]->block[64];kernel[0]->grid[1024]}
```

- Change optimization parameters

```
$ pencilcc source.c -o program \
"--sizes={kernel[i]->tile[32];kernel[i]->block[64];kernel[i]->grid[1024]}"
```

- Add hand-written OpenCL code

```
$ pencilcc source.c -o program --opencl-include-file=/path/to/file.inc
```

- When type-declarations conflict:

```
$ pencilcc source.c --opencl-include-file=file.inc --no-opencl-print-kernel-types
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

- Dump effective optimization parameters

```
$ pencilcc source.c -o program --dump-sizes
{kernel[0]->tile[32];kernel[0]->block[64];kernel[0]->grid[1024]}
```

- Change optimization parameters

```
$ pencilcc source.c -o program \
"--sizes={kernel[i]->tile[32];kernel[i]->block[64];kernel[i]->grid[1024]}"
```

- Add hand-written OpenCL code

```
$ pencilcc source.c -o program --opencl-include-file=/path/to/file.inc
```

- When type-declarations conflict:

```
$ pencilcc source.c --opencl-include-file=file.inc --no-opencl-print-kernel-types
```

- Don't use shared (OpenCL: `__local`) memory:

```
$ pencilcc source.c -o program --no-shared-memory
```

Usage II

- Set tile size

```
$ pencilcc source.c -o program -S 32
```

- Dump effective optimization parameters

```
$ pencilcc source.c -o program --dump-sizes
{kernel[0]->tile[32];kernel[0]->block[64];kernel[0]->grid[1024]}
```

- Change optimization parameters

```
$ pencilcc source.c -o program \
"--sizes={kernel[i]->tile[32];kernel[i]->block[64];kernel[i]->grid[1024]}"
```

- Add hand-written OpenCL code

```
$ pencilcc source.c -o program --opencl-include-file=/path/to/file.inc
```

- When type-declarations conflict:

```
$ pencilcc source.c --opencl-include-file=file.inc --no-opencl-print-kernel-types
```

- Don't use shared (OpenCL: `__local`) memory:

```
$ pencilcc source.c -o program --no-shared-memory
```

- Don't use private (thread-local) memory:

```
$ pencilcc source.c -o program --private-memory
```

Ideal Loop Nesting

```
1  void foo(int n, int m, float A[static const restrict m][n],
2                         float B[static const restrict m][n]) {
3  #pragma scop
4    for (int i = 0; i < m; ++i) {
5      for (int j = 0; j < n; ++j) {
6        A[i][j] = B[i][j];
7      }
8    }
9  #pragma endscop
10 }
```

DOs

DO use idiomatic for-loops

DOs

DO use idiomatic for-loops

DO use C99 VLAs

DOs

DO use idiomatic for-loops

DO use C99 VLAs

DO use `static const restrict` in declarations

DOs

DO use idiomatic for-loops

DO use C99 VLAs

DO use `static const restrict` in declarations

DO use `__pencil_assume` (lower/upper bound for each parameter)

DOs

DO use idiomatic for-loops

DO use C99 VLAs

DO use `static const restrict` in declarations

DO use `__pencil_assume` (lower/upper bound for each parameter)

DO use `__pencil_kill`

DOs

DO use idiomatic for-loops

DO use C99 VLAs

DO use `static const restrict` in declarations

DO use `__pencil_assume` (lower/upper bound for each parameter)

DO use `__pencil_kill`

DO declare local arrays right after `#pragma scop`

DONTs

AVOID while loops

DONTs

AVOID while loops

AVOID non-affine array subscripts

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONT alias

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONT alias

DONT exceed an array's dimensions's size

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONT alias

DONT exceed an array's dimensions's size

DONT declare structs/arrays variables inside loops (SROA them!)

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONT alias

DONT exceed an array's dimensions's size

DONT declare structs/arrays variables inside loops (SROA them!)

AVOID extra statements to compute subexpressions

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONT alias

DONT exceed an array's dimensions's size

DONT declare structs/arrays variables inside loops (SROA them!)

AVOID extra statements to compute subexpressions

AVOID reductions

DONTs

AVOID while loops

AVOID non-affine array subscripts

AVOID non-affine if-condition (Ternary operator no problem)

DONT alias

DONT exceed an array's dimensions's size

DONT declare structs/arrays variables inside loops (SROA them!)

AVOID extra statements to compute subexpressions

AVOID reductions

DONT write unusual code (Assign `n,m`; Shadowing; Assign multiple values in one statements; ...)

Use Multidimensional Arrays

```
1 #define INDEX(i,j) ((i)*n+(j))
2 void foo(int n, int m, float *A, float *B) {
3   for (int i = 0; i < m; ++i)
4     for (int j = 0; j < n; ++j)
5       A[INDEX(i,j)] = B[INDEX(i,j)];
6 }
```

$\Rightarrow$

```
1 void foo(int n, int m, float A[static const restrict m][n],
2                         float B[static const restrict m][n]) {
3   for (int i = 0; i < m; ++i)
4     for (int j = 0; j < n; ++j)
5       A[i][j] = B[i][j];
6 }
```

Use __pencil_assume

```
1 void foo(int m, float A[static const restrict m],
2               float B[static const restrict m]) {
3 #pragma scop
4   for (int i = 0; i < m; ++i)
5     A[i] = B[i];
6 #pragma endscop
7 }
```

⇒

```
1 void foo(int m, float A[static const restrict m],
2               float B[static const restrict m]) {
3 #pragma scop
4     __pencil_assume(m > 0);
5     __pencil_assume(m <= 128);
6   for (int i = 0; i < m; ++i)
7     A[i] = B[i];
8 #pragma endscop
9 }
```

Use __pencil_kill

```
1 void foo(int m, float A[static const restrict m],
2               float B[static const restrict m]) {
3 #pragma scop
4   for (int i = 0; i < m; ++i)
5     A[i] = B[i];
6 #pragma endscop
7 }
```

⇒

```
1 void foo(int m, float A[static const restrict m],
2               float B[static const restrict m]) {
3 #pragma scop
4   __pencil_kill(A);
5   for (int i = 0; i < m; ++i)
6     A[i] = B[i];
7 #pragma endscop
8 }
```

Declare Local Arrays within SCoP

```
1  void foo(int m, float A[static const restrict m],
2               float B[static const restrict m]) {
3    float tmp[m];
4  #pragma scop
5    for (int i = 0; i < m; ++i) {
6      tmp[i] = B[i];
7      A[i] = tmp[i];
8    }
9  #pragma endscop
10 }
```

$\Rightarrow$

```
1  void foo(int m, float A[static const restrict m],
2               float B[static const restrict m]) {
3  #pragma scop
4    float tmp[m];
5    for (int i = 0; i < m; ++i) {
6      tmp[i] = B[i];
7      A[i] = tmp[i];
8    }
9  #pragma endscop
```

No Iteration Over Pointers

```
1 for (float *p = A; p <= A+n; ++p)
2   *p = 0;
```

$\Rightarrow$

```
1 for (int i = 0; i < m; ++i)
2   A[i] = 0;
```

Avoid If-Conditions using Ternary Operator

```
1  for (int i = 0; i < m; ++i)
2    for (int j = 0; j < n; ++j)
3      if (x)
4        c = 5;
5      else
6        c = 6;
7      A[i][j] = c * B[i][j];
```

⇒

```
1  for (int i = 0; i < m; ++i) {
2    for (int j = 0; j < n; ++j) {
3      c = x ? 5 : 6;
4      A[i][j] = c * B[i][j];
5    }
```

Avoid Precomputing Common Subexpressions

```
1 for (int i = 0; i < m; ++i) {
2   c = i * i;
3   for (int j = 0; j < n; ++j)
4     A[i][j] = c * B[i][j];
5 }
```

$\Rightarrow$

```
1 for (int i = 0; i < m; ++i) {
2   for (int j = 0; j < n; ++j)
3     A[i][j] = (i*i) * B[i][j];
```

(except computing `c` is very complex)

Avoid Loop-Local Structs/Arrays I

```
1 for (int i = 0; i < m; ++i) {
2   for (int j = 0; j < n; ++j) {
3     struct float2 c = {1.0f, 0.0f};
4     A[i][j] = c.x * c.x + c.y * c.y;
5   }
6 }
```

$\Rightarrow$

```
1 for (int i = 0; i < m; ++i) {
2   for (int j = 0; j < n; ++j) {
3     float x = 1.0f;
4     float y = 0.0f;
5     A[i][j] = x * x + y * y;
6   }
7 }
```

Avoid Loop-Local Arrays II

```
1  void foo(int n, int m, float A[static const restrict m][n],
2                         float B[static const restrict m][n]) {
3    for (int i = 0; i < m; ++i) {
4      float tmp[n];
5      for (int j = 0; j < n; ++j) {
6        tmp[j] = B[i][j];
7        A[i][j] = tmp[j];
8      }
9    }
10 }
```

$\Rightarrow$

```
1  void foo(int n, int m, float A[static const restrict m][n],
2                         float B[static const restrict m][n]) {
3    float tmp[m][n];
4    for (int i = 0; i < m; ++i)
5      for (int j = 0; j < n; ++j) {
6        tmp[i][j] = B[i][j];
7        A[i][j] = tmp[i][j];
8      }
9  }
```

Help: PPCG did not parallelize my code!

- Add `#pragma pencil parallel`
- Contains a function call? Add a summary function
- Is there a locally declared struct/array variable?
- Non-affine array subscripts?
- Non-affine if-conditions?
- While loops?
- True serial dependency?

Help: PPCG takes too long!

- `-isl-schedule-max-coefficient=1`
- Use fewer statements (`a = b; ++a` ⇒ `a = b+1`)
- [Last resort] Translate code sections yourself in OpenCL/CUDA
- [If you feel experimental] Decorate code sections with tautological non-affine if-conditions (e.g. `if ((double)i*i>=0)`)