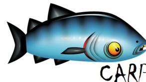


Pencil

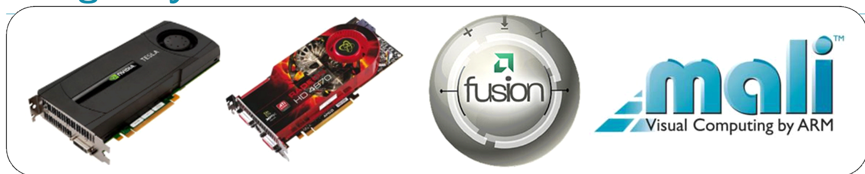
A Platform-Neutral Compute Intermediate Language for DSL Compilers

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

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

20th January 2015

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

Target platforms

- NVIDIA GPUs
- AMD GPUs
- ARM (Mali, Adreno, ...)
- ...

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

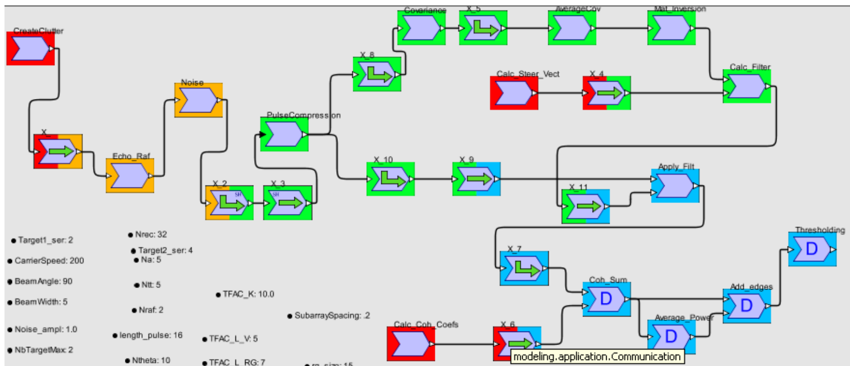
AMD GPUs

ARM (Mali, Adreno, ...)

...

Compilers

Domain languages | VOBLA | SpearDE



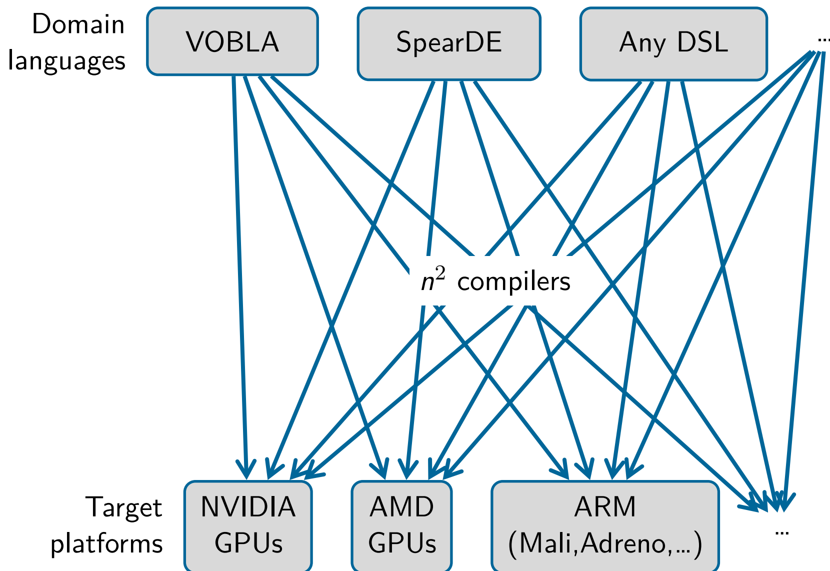
Target platforms | NVIDIA GPUs | AMD GPUs | ARM (Mali, Adreno, ...) | ...

Compilers

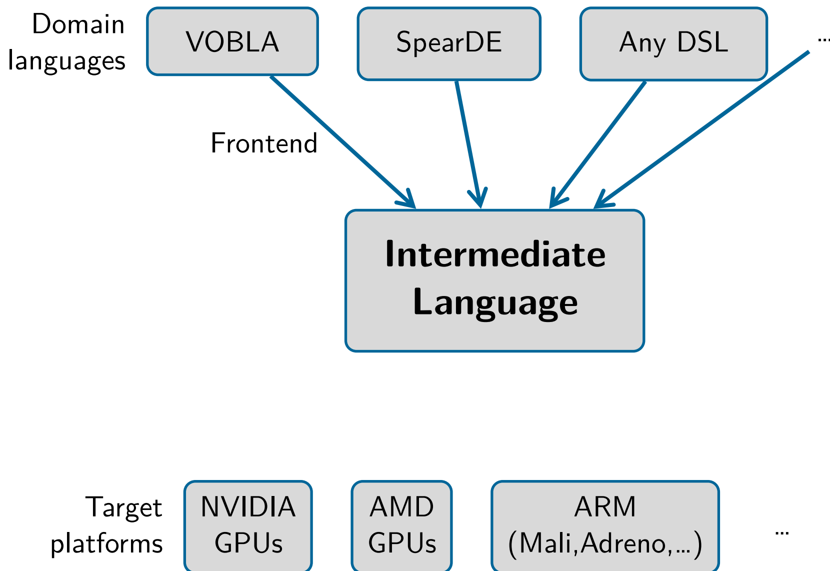
Domain languages: VOBLA | SpearDE | Any DSL | ...

Target platforms: NVIDIA GPUs | AMD GPUs | ARM (Mali,Adreno,...) | ...

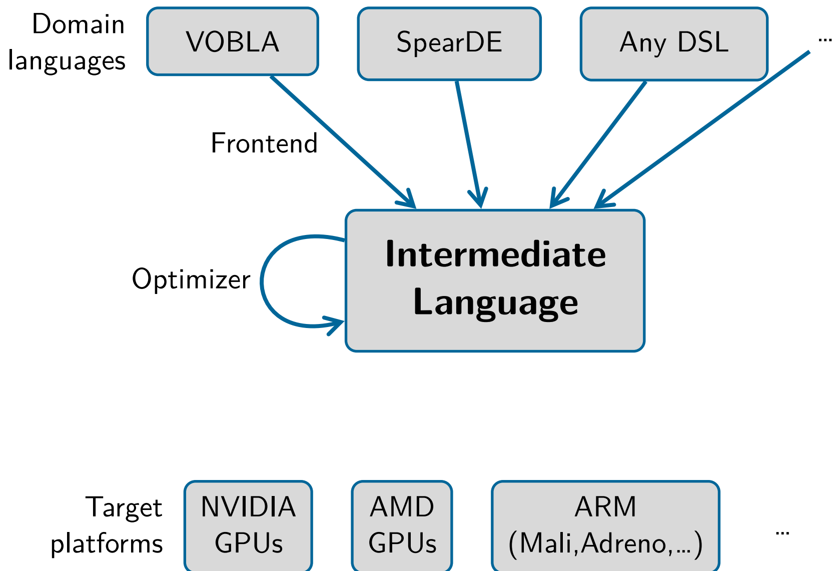
Compilers



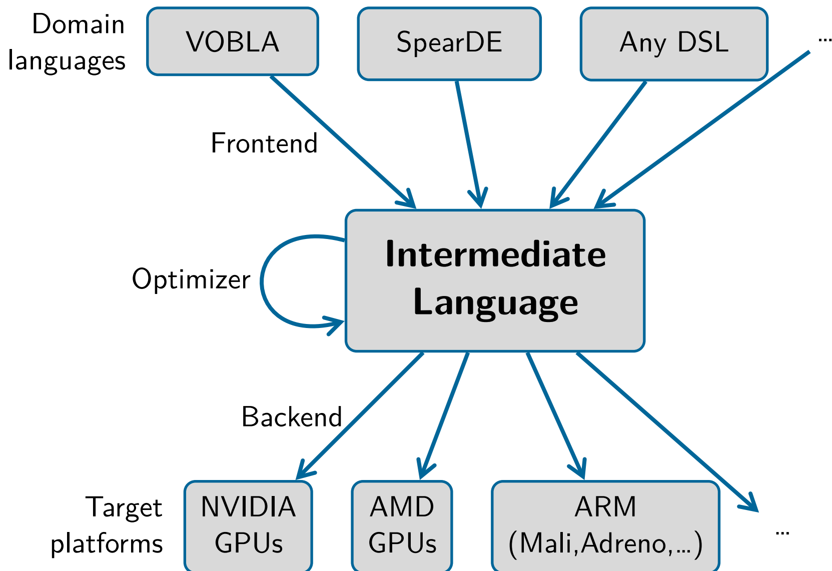
Compilers



Compilers



Compilers



Overview

1. Introduction
2. Approach
3. The Pencil Language
4. Toolchain Demonstration
5. Experiments
6. Discussion

Ideas

Ideas

- Intermediate language
  Any frontend language possible

Ideas

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

Ideas

- Intermediate language
  Any frontend language possible

- Polyhedral compiler friendly
  Powerful analysis and optimizations

- Meta-level specification
  Code properties not derivable from code itself (helps pessimistic optimizers)

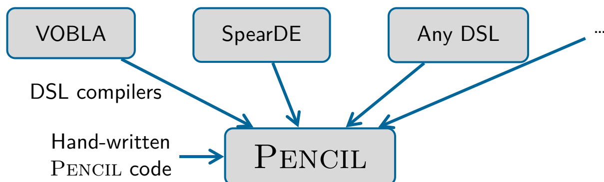
Ideas

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

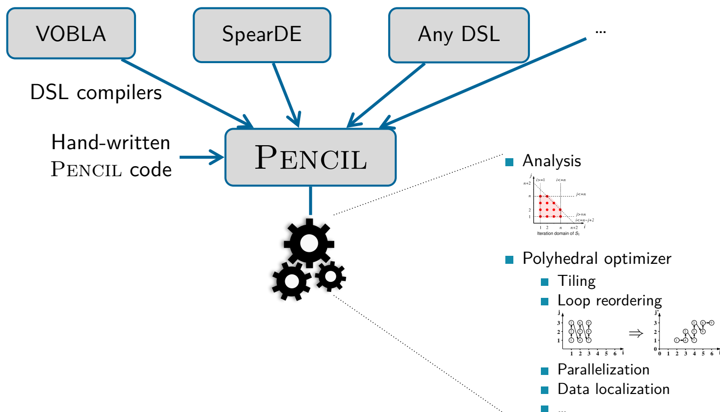
General Flow

VOBLA | SpearDE | Any DSL | ...

General Flow

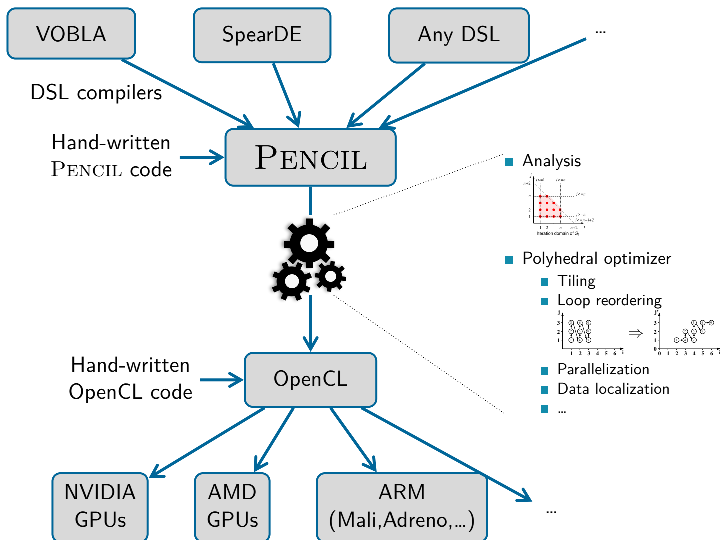


General Flow

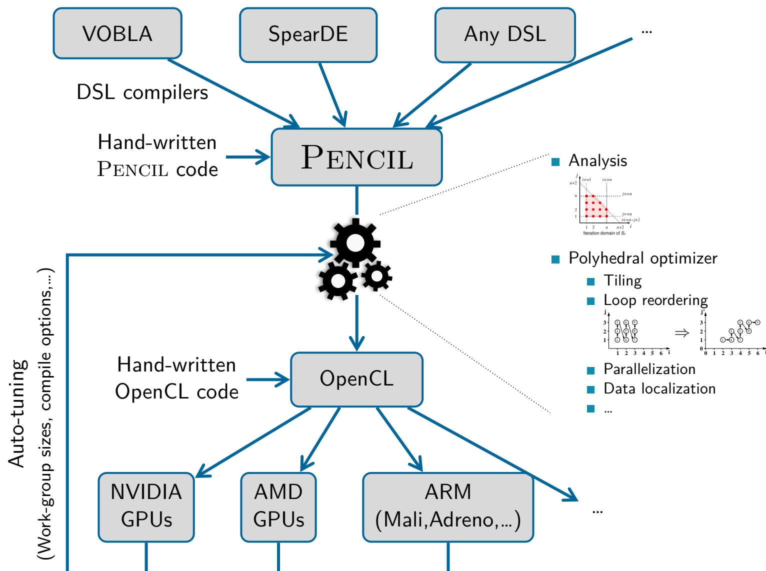


- Analysis
- Polyhedral optimizer
  - Tiling
  - Loop reordering
  - Parallelization
  - Data localization
  - ...

General Flow



General Flow



- Analysis
- Polyhedral optimizer
  - Tiling
  - Loop reordering
  - Parallelization
  - Data localization
  - ...

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99

Pencil Language Properties

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

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - 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
  - 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
  - No GOTOs, 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)
    ```
    ⇒
    ```
    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
  - 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)
    ```
    ⇒
    ```
    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
  - 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 violations undefined behviour

Pencil Language Properties

- Polyhedral-friendly subset of (GNU) C99
  - 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)
    ```

    ⇒

    ```
    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 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

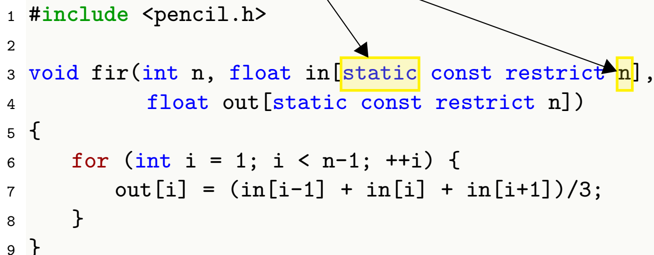
Example: Finite Impulse Response

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 }
```

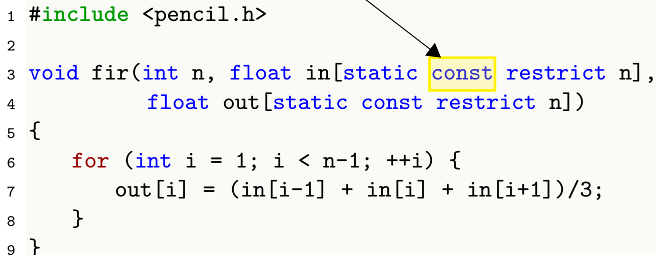
General Syntax

Example: Finite Impulse Response



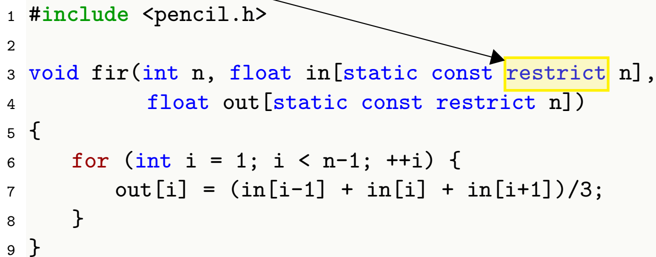
General Syntax

Example: Finite Impulse Response



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

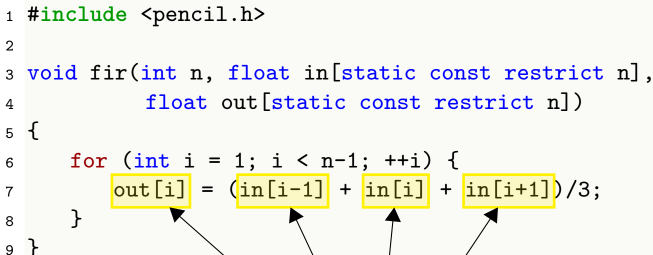
Example: Finite Impulse Response

```
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

```
void basic_histogram(uchar image[static const restrict N][M],
                     int hist[static const restrict K]) {

    for (int i=0; i<N; i++)

        for (int j=0; j<M; j++) {
            int idx = image[i][j];
            atomic_add(hist[idx], 1);
        }
}
```

Independent Directive

Example: Basic Histogram

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

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

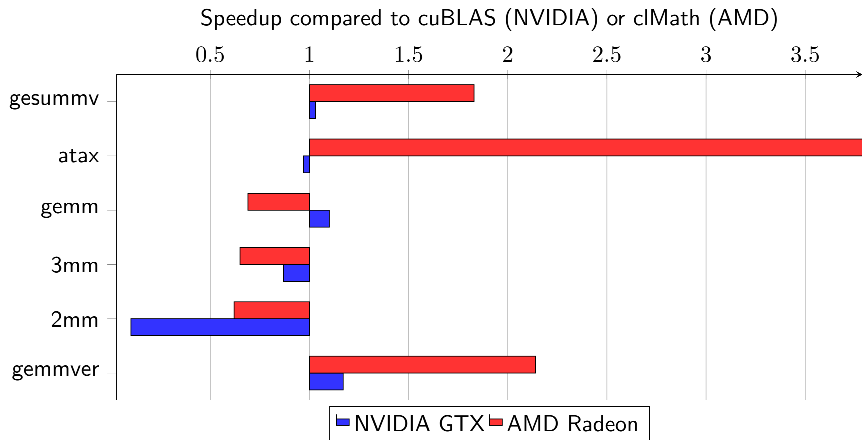
```
1  static void fftKernel32_summary(int n, int start, struct float2 a[static const restrict n]) {
2      for (int i=start; i<start+32; ++i)
3          USE(a[i]);
4      for (int i=start; i<start+32; ++i)
5          DEF(a[i]);
6  }
7
8  void fftKernel32(int n, int start, struct float2 a[static const 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 }
```

Demo

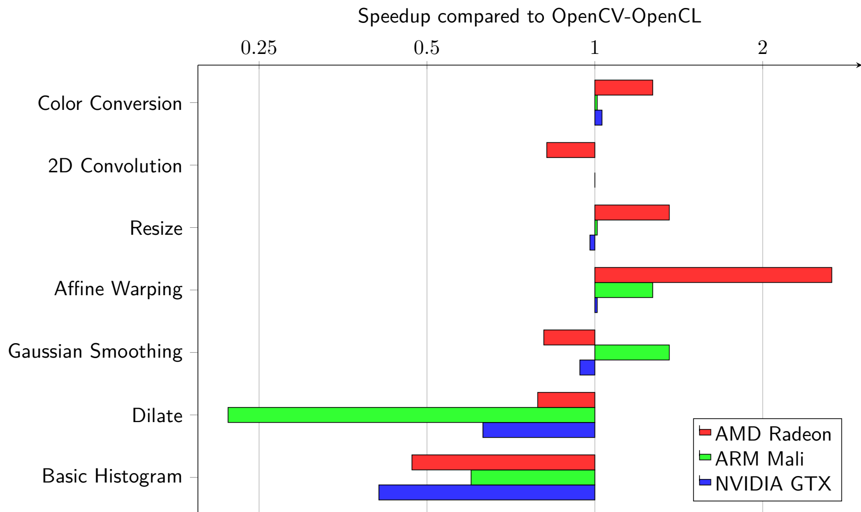
Experiments

- Compare OpenCL generated by PPCG to original
    - cuBLAS/clMath (Linear Algebra)
    - OpenCV (Image Processing)
    - Rodina, SHOC (OpenCL Benchmarks)
    - SPEAR-DE (Signal Processing)
- Platforms
    - AMD Radeon HD 5670 GPU
    - ARM Mali-T604 GPU
    - NVIDIA GTX470 GPU

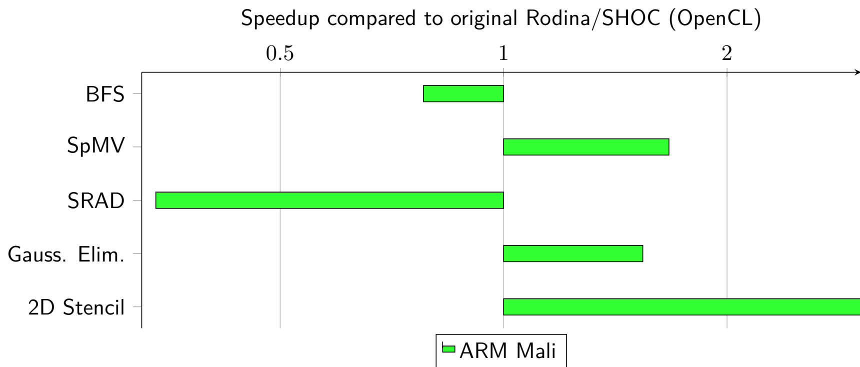
BLAS



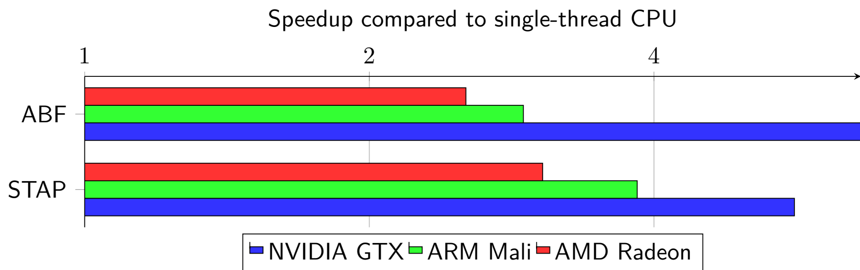
OpenCV



Rodina/SHOC



SPEAR-DE



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

That's all Folks!

Compiling Pencil

- Irregular array accesses (read/write)
  - Treated as possible access to the whole array dimension (may write)
  - Example:
    - `A[i] = B[foo(i)]`
      is treated as
    - `A[i] = B[*]`
- `__pencil_assume(expression)`
  - `expression` is a constraint on loop parameters
  - `expression` is added to the context (set of constraints on loop parameters)
  - This information (context) is used whenever needed
- `__pencil_kill(expression)`
  - Copy-in: Mark flow-dependencies to expression as “no source”
  - Copy-out: Remove output-dependencies from expression
- `independent` directive
  - Remove all loop carried dependences