# Adopt a Polyhedral Compiler!

## IMPACT 2013 Workshop

Albert Cohen

INRIA and École Normale Supérieure, Paris
http://www.di.ens.fr/ParkasTeam.html

## People Have Great Expectations

- Accelerating legacy code for ever
- Simplifying compiler construction and library generation
- Peak performance at the touch of a button
- Proving program transformations
- Code generation for heterogeneous architectures
- High-level circuit synthesis
- Publishing great papers
- [ Name your own dream project here ]

# There Were, and Will Be Times in the Wilderness...

# But the World Will Eventually Turn Polyhedral!

Courtesy `www.progonos.com/furuti`

# Lost Memories in the Not-Yet-Polyhedral World

DDR3-2133 SDRAM

Latency: $\mathbf{10.3}$ ns

Memory bandwidth: $\mathbf{17.6}$ GB/s

4-core 2GHz ARM Cortex A15

Compute bandwidth: $\mathbf{2} \times \mathbf{4}$ threads $\times\ \mathbf{1}$ NEON unit $\times\ \mathbf{16}$ bytes $\times\ \mathbf{2}$ GHz $= \mathbf{1024}$ GB/s

8-core 3GHz AMD Opteron

Compute bandwidth: $\mathbf{2} \times \mathbf{8}$ threads $\times\ \mathbf{2}$ SSE units $\times\ \mathbf{16}$ bytes $\times\ \mathbf{3}$ GHz $= \mathbf{1536}$ GB/s

Memory bandwidth: $\mathbf{17.6}$ GB/s

256-core 400MHz Kalray MPPA

Compute bandwidth: $\mathbf{2} \times \mathbf{256}$ threads $\times\ \mathbf{2}$ words $\times\ \mathbf{4}$ bytes $\times\ \mathbf{400}$ MHz $= \mathbf{1638.4}$ GB/s

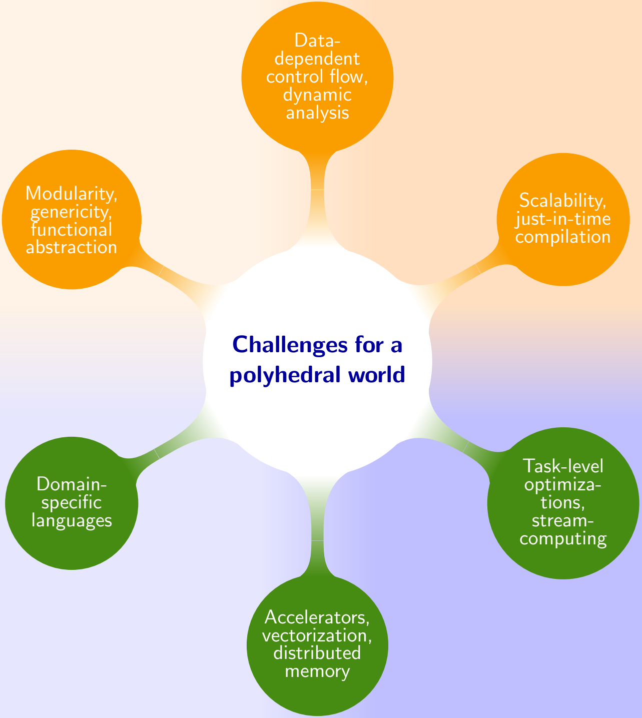
1536-core 1.006GHz NVIDIA Kepler

Compute bandwidth: $\mathbf{2} \times \mathbf{1536}$ threads $\times\ \mathbf{1}$ float $\times\ \mathbf{4}$ bytes $\times\ \mathbf{1.006}$ GHz $= \mathbf{12361.6}$ GB/s

Memory bandwidth: $\mathbf{190}$ GB/s

# Many Candidates for Adoption

- **What are the essential semantic requirements for source programs?**
- **Should programmers care**
  About parallelism?
  About the memory and power walls?
  **Which programmers?**
- **What role for the software stack?**
  Compilers
  Runtime systems
  Libraries, library generators
  Auto-tuning, dynamic optimization
  Operating system, virtual machine monitor
- **What role for the polyhedral tools?**


Challenges for a polyhedral world
Data-dependent control flow, dynamic analysis
Scalability, just-in-time compilation
Task-level optimizations, stream-computing
Accelerators, vectorization, distributed memory
Domain-specific languages
Modularity, genericity, functional abstraction

# State-of-the-Art Tool: PPCG – Polyhedral Parallel Code Generator

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

- Input: C
- Output:
  - OpenMP
  - CUDA
  - OpenCL (soon)

# State-of-the-Art Tool: PPCG – Polyhedral Parallel Code Generator

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

- Input: C
- Output:
  - OpenMP
  - CUDA
  - OpenCL (soon)

Steps:

- Extract polyhedral model from source code (`pet`,`isl`)
- Dependence analysis (`isl`)
- Scheduling (`isl`)
  - Expose parallelism and tiling opportunities
  - Separate schedule into parts mapped on host and GPU
  - perform tiling, mapping outer parallel dimensions to blocks and inner parallel dimensions to threads
- Memory management (`isl`)
  - Add transfers of data to/from GPU (`isl`)
  - Detect array reference groups
  - Allocate groups to registers and shared memory
- Generate AST (`isl`)

# PPCG Example – Input

Source code:

```
void matmul(int M, int N, int K,
           float A[static const restrict M][K],
           float B[static const restrict K][N],
           float C[static const restrict M][N])
{
  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
}
```

Options:

```
--ctx="[M,N,K] -> { : M = N = K = 256 }"
--sizes="{ kernel[i] -> tile[16,16,16];
           kernel[i] -> block[8,16] }"
```

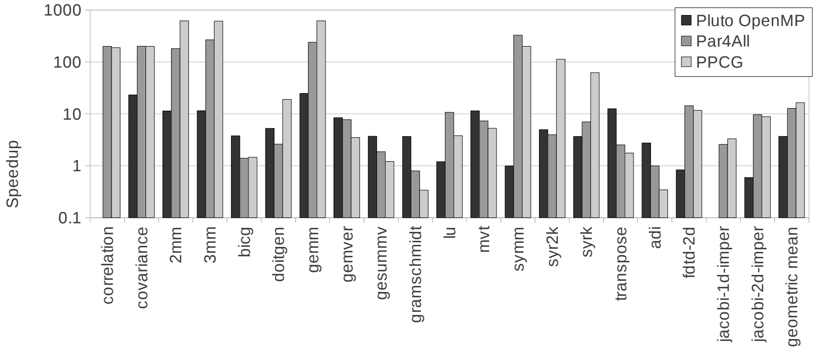
# PPCG Example – Output

Kernel code: (host code not shown)

```
int b0 = blockIdx.y, b1 = blockIdx.x;
int t0 = threadIdx.y, t1 = threadIdx.x;
__shared__ float s_A[16][16];
__shared__ float s_B[16][16];
float p_C[2][1];

p_C[0][0] = C[(16 * b0 + t0) * (256) + 16 * b1 + t1];
p_C[1][0] = C[(16 * b0 + t0 + 8) * (256) + 16 * b1 + t1];
for (int g9 = 0; g9 <= 240; g9 += 16) {
  for (int c0 = t0; c0 <= 15; c0 += 8)
    s_B[c0][t1] = B[(g9 + c0) * (256) + 16 * b1 + t1];
  for (int c0 = t0; c0 <= 15; c0 += 8)
    s_A[c0][t1] = A[(16 * b0 + c0) * (256) + t1 + g9];
  __syncthreads();
  if (g9 == 0) {
    p_C[0][0] = (0);
    p_C[1][0] = (0);
  }
  for (int c2 = 0; c2 <= 15; c2 += 1) {
    p_C[0][0] = (p_C[0][0] + (s_A[t0][c2] * s_B[c2][t1]));
    p_C[1][0] = (p_C[1][0] + (s_A[t0 + 8][c2] * s_B[c2][t1]));
  }
  __syncthreads();
}
C[(16 * b0 + t0) * (256) + 16 * b1 + t1] = p_C[0][0];
C[(16 * b0 + t0 + 8) * (256) + 16 * b1 + t1] = p_C[1][0];
```

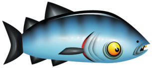
# PPCG Results




- Benchmarks: PolyBench 3.1
- Platform: Tesla M2070
- Baseline: sequential CPU execution `gcc -Ofast`

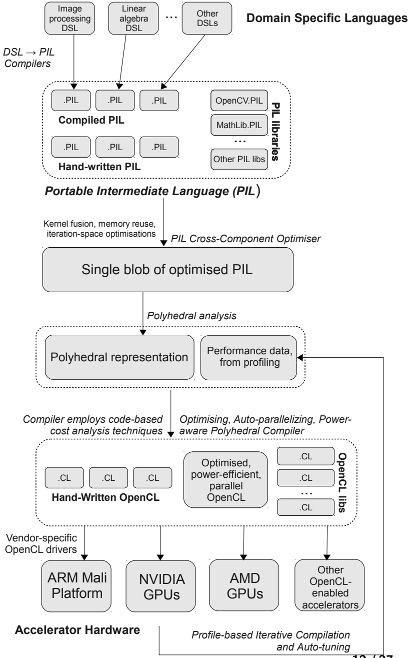
Attend Carlos Juega's talk on Wednesday morning!

# CARP EU Project



w/ ARM, RealEyes, Rightware, Monoidics, Imperial College, RWTH Aachen, U. Twente

- Compiler construction for DSLs: support for parallelization, vectorization, loop transformation...
- Reconcile advanced loop nest optimizations and software engineering practices

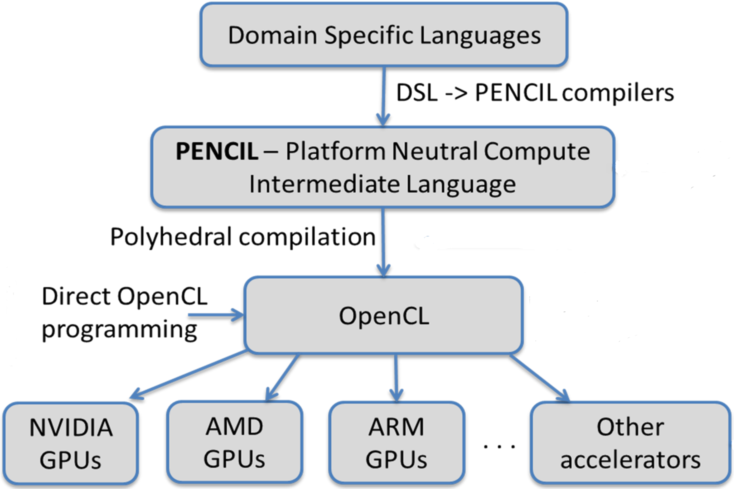


# DSLs to the Rescue

- Problem: general purpose languages are not optimization-friendly
  - much static semantics is lost
  - much domain information is lost
  - high expressiveness $\rightarrow$ ambiguitis disable optimizations (e.g., pointer aliasing)
- Some DSLs are designed primarily for abstraction and productivity
  $\rightarrow$ we are interested in the performance-focused DSLs

- But compiling DSLs directly into OpenCL or CUDA is not advisable
- Approach: target an appropriate intermediate language (IL) and leverage a generic optimization framework

# Zooming in on Pencil



# Pencil: a Platform-Neutral Compute Intermediate Language

## An intermediate language for DSL compilers

- C-based intermediate language
- Code regions specifically marked as PENCIL-compliant
- Sequential, platform neutral
- A set of coding rules, language extensions and directives
- Planning for an LLVM IR version of PENCIL
- Complementary objectives to DSL intermediate languages such as Delite IR

## 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

# Platform-Neutral Compute Intermediate Languages

- Coding rules for PENCIL functions
- Language extensions (C11-compatible)
- Directives

# Platform-Neutral Compute Intermediate Languages

- Coding rules for PENCIL functions
  - cannot be recursive
  - no `goto`s
  - no pointers
  - array arguments should be declared with `static const restrict` inferred through automatic versioning
  - dedicated types and builtins for dynamic analysis (work in progress)

- Language extensions (C11-compatible)

- Directives

# Platform-Neutral Compute Intermediate Languages

- Coding rules for PENCIL functions
  - cannot be recursive
  - no `gotos`
  - no pointers
  - array arguments should be declared with `static const restrict` inferred through automatic versioning
  - dedicated types and builtins for dynamic analysis (work in progress)
- Language extensions (C11-compatible)
  - access summary functions
    - describe access pattern of a function if static analysis cannot be performed (no source or not PENCIL compliant) or if the results are too inaccurate
    - modular interprocedural information used in the caller through "polyhedral inlining"
- Directives

# Platform-Neutral Compute Intermediate Languages

- Coding rules for PENCIL functions
  - cannot be recursive
  - no `goto`s
  - no pointers
  - array arguments should be declared with `static const restrict` inferred through automatic versioning
  - dedicated types and builtins for dynamic analysis (work in progress)

- Language extensions (C11-compatible)
  - access summary functions
    - describe access pattern of a function if static analysis cannot be performed (no source or not PENCIL compliant) or if the results are too inaccurate
    - modular interprocedural information used in the caller through "polyhedral inlining"

- 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[static const restrict 100][100],
             int C[static 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[static const restrict n][n]) {
  for (int i=0; i<n; i++)
    USE(C[i]); // marks row i of C as being read
}

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

int function(int A[static const restrict 100][100],
             int C[static 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);
}
```

# Modularity, Genericity, Functional Abstraction, and DSLs

- Short-term
  - Functional abstraction $\longrightarrow$ inlining
  - Genericity $\longrightarrow$ specialization, partial evaluation
  - Modularity $\longrightarrow$ staged programs: write program generators
  - ... a roadmap for a DSL compiler builder
    cf. NumPy, pythran, C++ template metaprogramming (TaskGraph library, RapidMind/ArBB), Delite (Scala), Halide, OP2, MetaOCaml experiments...

- Long-term
  - Support function-level fusion, vectorization, tiling
    cf. Kennedy's Telescoping Languages
  - On-demand function cloning rather than inlining

# What Else Do You Want From a Polyhedral Compiler?

# What Else Do You Want From a Polyhedral Compiler?

**Complex transformations**

**Scalability**

**Just-in-time and split compilation**

**Auto-tuning**

# What Else Do You Want From a Polyhedral Compiler?

**Complex transformations**

**Scalability**

**Just-in-time and split compilation**

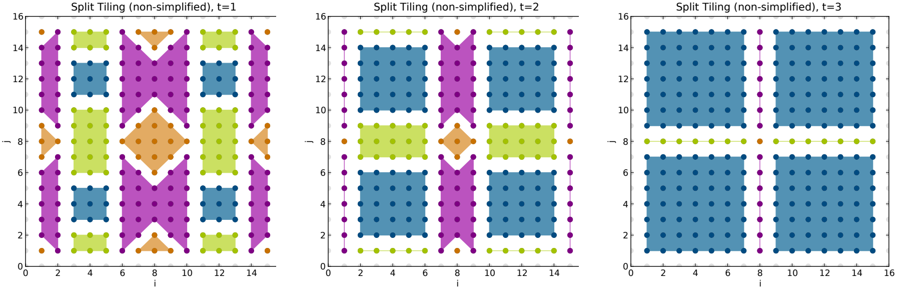
**Auto-tuning**

**Dynamic analysis, optimistic transformations**

**Adaptation and optimization of parallel code**

# Complex Transformations

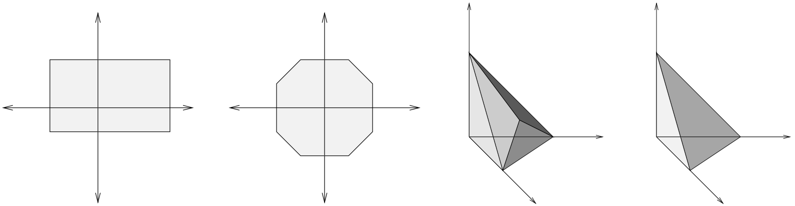
E.g., split tiling, diamond tiling, overlapped tiling...




More complex?

- Instancewise code generation options
- Scripting affine transformations
- [ Your crazy idea here ]

# Scalability: Sub-Polyhedral Approximations



Interval

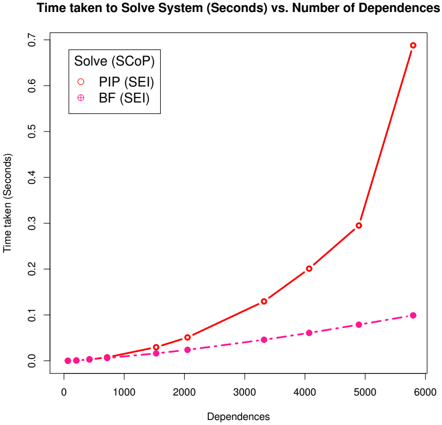
| Interval | Octagon (UTVPI) | TVPI | Convex Polyhedra |
|---|---|---|---|
| $a \leqslant x_i \leqslant b$ | $\pm x_i \pm x_j \leqslant c$ | $ax_i + bx_j \leqslant c$ | $\sum a_i x_i \leqslant c$ |

$$\xrightarrow{\textbf{Precision}}$$
$$\text{Intervals} \subset \text{Octagons (UTVPI)} \subset \text{TVPI} \subset \text{Polyhedra}$$
$$\xrightarrow[\textbf{Cost}]{}$$

# Scalability: Sub-Polyhedral Approximations

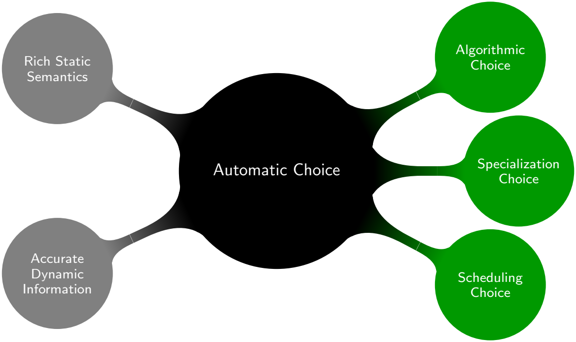
- Replace linear programming (Simplex) with Bellman Ford
  $\mathcal{O}(mn^3) \rightsquigarrow \mathcal{O}(mn)$
- Applicable to dependence analysis, code generation:
  over-approximation
- Applicable to affine scheduling:
  under-approximation
- Preserving feasibility of polyhedra is a tough challenge for some affine scheduling problems

(Unrolled) Gauss-Seidel benchmark
Automatic parallelization with PLuTo



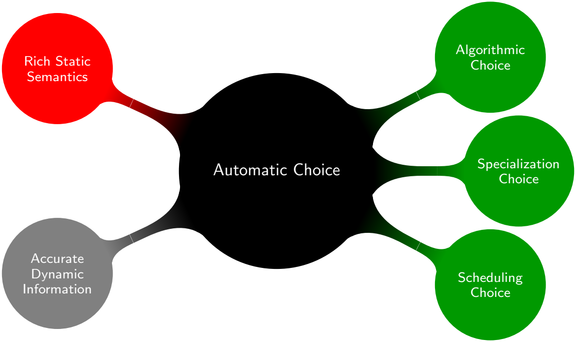
# Just-in-Time and Split Compilation

Tools could do a lot better, if provided with enough choice and precise information



# Just-in-Time and Split Compilation

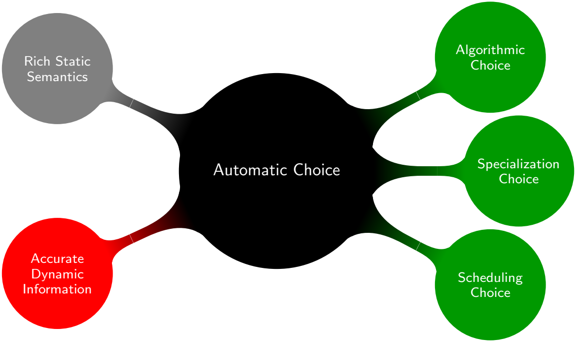
Tools could do a lot better, if provided with enough choice and precise information




Importance of static, non-functional semantics

# Just-in-Time and Split Compilation

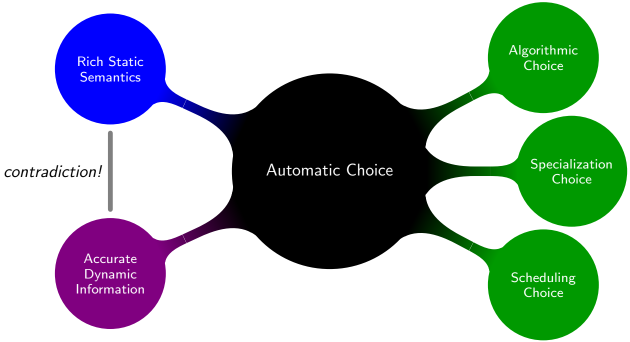
Tools could do a lot better, if provided with enough choice and precise information




Importance of delaying choice until information is available

# Just-in-Time and Split Compilation

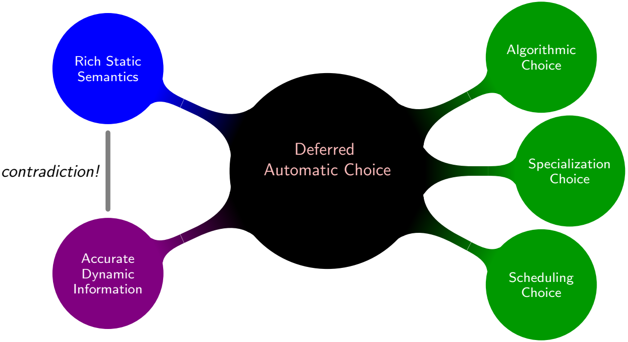
Tools could do a lot better, if provided with enough choice and precise information




Contradiction: *accurate information is only available after the most important choices have already been made*

# Just-in-Time and Split Compilation

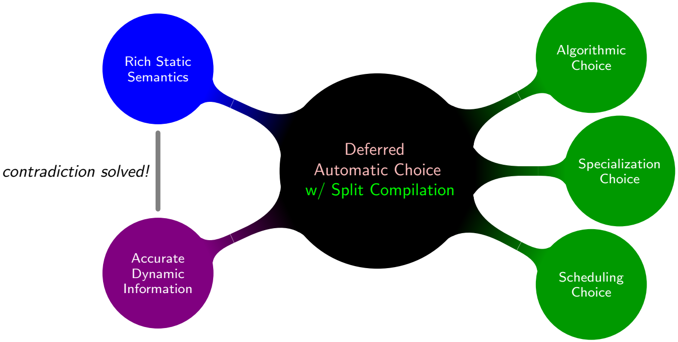
Tools could do a lot better, if provided with enough choice and precise information




Deferred compilation enables Just-in-Time (JIT) optimization when accurate information is available, but loses much of the static semantics carrying choice opportunities

# Just-in-Time and Split Compilation

Tools could do a lot better, if provided with enough choice and precise information



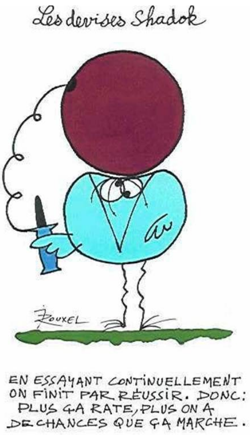

Contradiction solved with split compilation: *optimizations split over coordinated, offline and online compilation steps, communicating through rich intermediate languages*

# Auto-Tuning, Iterative Optimization, Machine Learning Compilation

Even with rich static semantics and accurate information, the compiler is left with a huge space of optimization and specialization opportunities

# Auto-Tuning, Iterative Optimization, Machine Learning Compilation

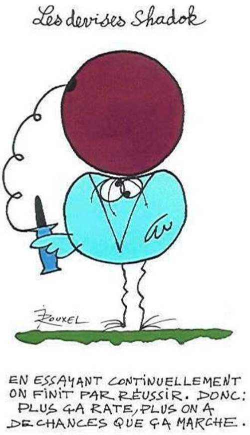
Even with rich static semantics and accurate information, the compiler is left with a huge space of optimization and specialization opportunities




*"By continuously trying, we finally succeed.*
*Therefore: the more it fails, the more it has chances to work."*

# Auto-Tuning, Iterative Optimization, Machine Learning Compilation

Even with rich static semantics and accurate information, the compiler is left with a huge space of optimization and specialization opportunities




*"By continuously trying, we finally succeed.*
*Therefore: the more it fails, the more it has chances to work."*

Principle of iterative, feedback-directed optimization

- Can be embedded transparently in a virtual execution environment

Machine learning techniques for split compilation

- Offline training, feeding online predictive models with target- and application-specific weights
- Leveraging static features in deferred compilation steps

# Adopt a Polyhedral Compiler!

**It is happening now**

- Many blockers have been lifted: better tools, more effective heuristics, better performance, more incentive to reengineer the compilers, more performance to gain, more market impact...
- The expectations are high, much work is awaiting us
- Convince industry to (really) invest into robust platforms, and address open issues, or let's build the software company that will do it

**Software**

"Production-quality" integer Set Library: `http://freshmeat.net/projects/isl`

$\longrightarrow$ barvinok, iscc, pet, ppcg, PLuTo, PoCC, Polly (LLVM), Graphite (GCC)