Enabling Compilers to exploit Heterogeneous Computing

Philip Ginsbach

Mike O'Boyle, Björn Franke, Adam Lopez and Chris Ryder

EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC Engineering and Physical Sciences Research Council

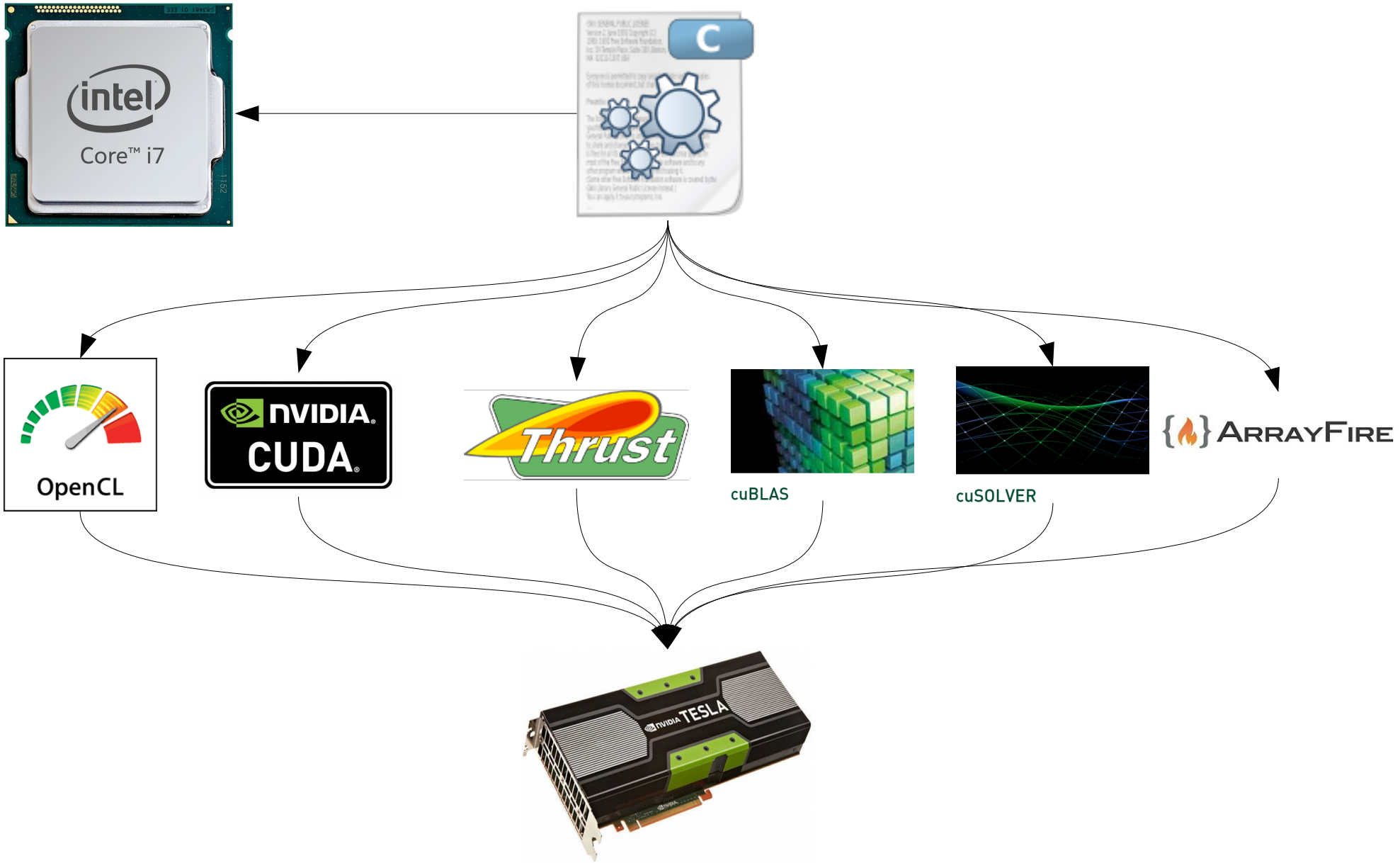
ARM

The University of Edinburgh informatics

Heterogeneous Computing

- Using structurally different processor cores
- Examples:
  - GPGPU (General Purpose GPU)
  - ARM big.LITTLE (slow CPU + fast CPU)
- Challenge: difficult to program

CPU + GPU


intel
Core™ i7
C
OpenCL
NVIDIA
CUDA
Thrust
cuBLAS
cuSOLVER
{} ArrayFire
NVIDIA TESLA

GPU-Accelerated Libraries

GPU-Accelerated libraries provide highly-optimized algorithms and functions you can incorporate into your applications, with minimal changes to your existing code. Many support drop-in compatibility to replace industry standard CPU-only libraries such as MKL, IPP, FFTW and widely-used libraries. Some also feature automatic multi-GPU performance scaling.

AmgX

A simple path to accelerated core solvers, providing up to 10x acceleration in the computationally intense linear solver portion of simulations, and is very well suited for implicit unstructured methods.

cuDNN

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks, it is designed to be integrated into higher-level machine learning frameworks.

cuFFT

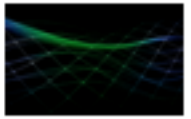
NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster, without having to develop your own custom GPU FFT implementation.

IndeX Framework

NVIDIA IndeX Framework is a real-time scalable visualization plug-in for ParaView.

cuRAND

The CUDA Random Number Generation library performs high quality GPU-accelerated random number generation (RNG) over 8x faster than typical CPU only code.



CUDA Math Library

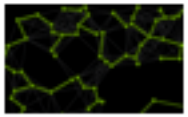
An industry proven, highly accurate collection of standard mathematical functions, providing high performance on NVIDIA GPUs.

Thrust

A powerful, open source library of parallel algorithms and data structures. Perform GPU-accelerated sort, scan, transform, and reductions with just a few lines of code.

NVBIO

A GPU-accelerated C++ framework for High-Throughput Sequence Analysis for both short and long read alignment.



nvGRAPH

nvGRAPH Analytics Library is a GPU-accelerated graph analytics library.

GIE

NVIDIA GPU Inference Engine is a high performance neural network inference library for deep learning applications

NPP

NVIDIA Performance Primitives is a GPU accelerated library with a very large collection of 1000's of image processing primitives and signal processing primitives.

FFmpeg

FFmpeg is a popular open-sour multi-media framework with a library of plugins that can be applied to various parts of the audio and video processing pipelines.

NVIDIA VIDEO CODEC SDK

Accelerate video compression with the NVIDIA Video Codec SDK. This SDK includes documentation and code samples that illustrate how to use NVIDIA's NVENC and NVDEC hardware in GPUs to accelerate encode, decode, and transcode of H.264 and HEVC video formats.

HiPLAR

HiPLAR (High Performance Linear Algebra in R) delivers high performance linear algebra (LA) routines for the R platform for statistical computing using the latest software libraries for heterogeneous architectures.

OpenCV

OpenCV is the leading open source library for computer vision, image processing and machine learning, and now features GPU acceleration for real-time operation.

Geometry Performance Primitives(GPP)

GPP is a computational geometry engine that is optimized for GPU acceleration, and can be used in advanced Graphical Information Systems (GIS), Electronic Design Automation (EDA), computer vision, and motion planning solutions.

CHOLMOD

GPU-accelerated CHOLMOD is part of the SuiteSparse linear algebra package by Prof. Tim Davis. SuiteSparse is used extensively throughout industry and academia.

CULA Tools

GPU-accelerated linear algebra library by EM Photonics, that utilizes CUDA to dramatically improve the computation speed of sophisticated mathematics.

MAGMA

A collection of next gen linear algebra routines. Designed for heterogeneous GPU-based architectures. Supports current LAPACK and BLAS standards.

IMSL Fortran Numerical Library

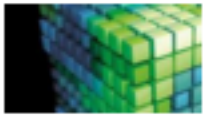
Developed by RogueWave, a comprehensive set of mathematical and statistical functions that offloads work to GPUs.

aralution

library for sparse iterative wthods with special focus on ulti-core and accelerator chnology such as GPUs.

Triton Ocean SDK

Triton provides real-time visual simulation of the ocean and bodies of water for games, simulation, and training applications.

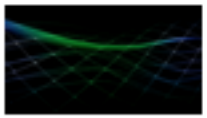


cuBLAS

NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS.

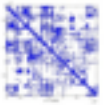
ArrayFire

Comprehensive, open source GPU function library. Includes functions for math, signal and image processing, statistics, and many more. Interfaces for C, C++, Java, R and Fortran.



cuSOLVER

A collection of dense and sparse direct solvers which deliver significant acceleration for Computer Vision, CFD, Computational Chemistry, and Linear Optimization applications



cuSPARSE

NVIDIA CUDA Sparse (cuSPARSE) Matrix library provides a collection of basic linear algebra subroutines used for sparse matrices that delivers over 8x performance boost.

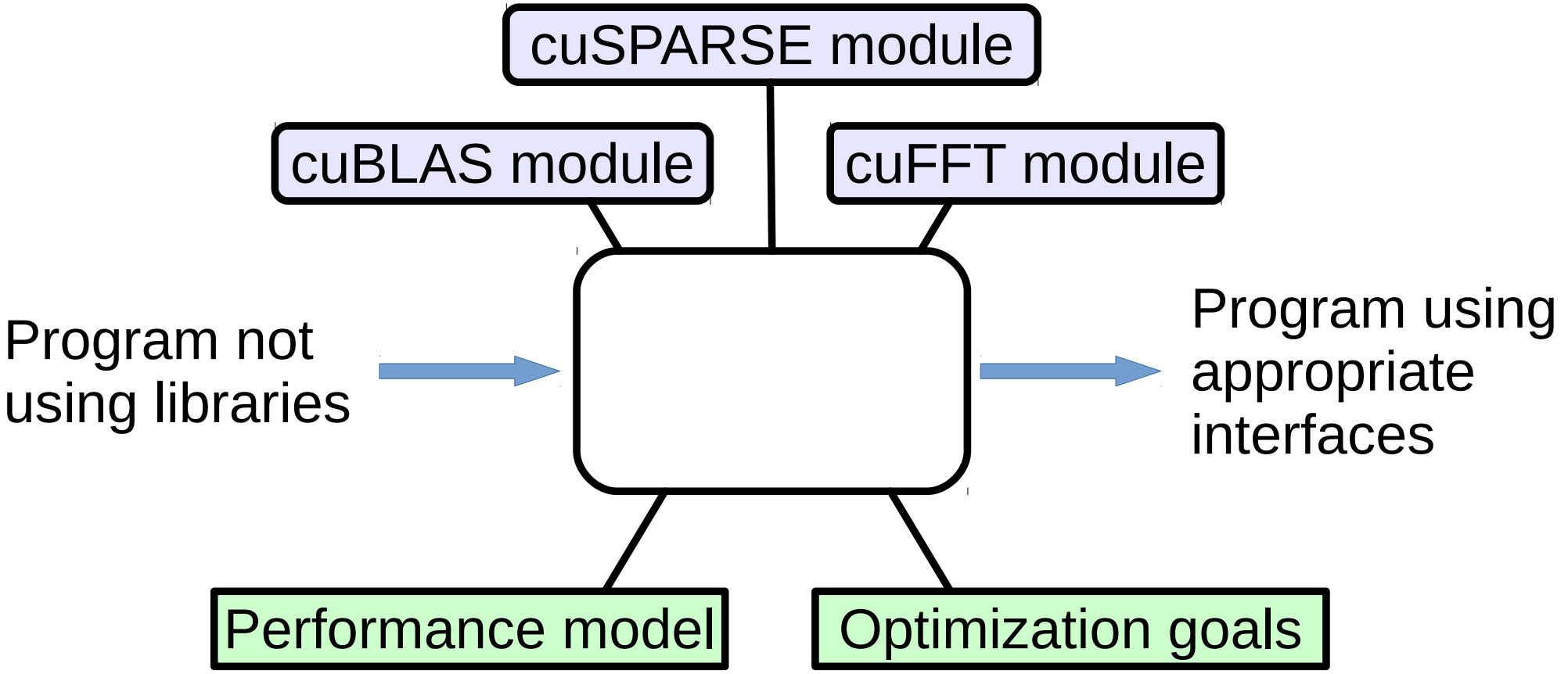
Heterogeneous Computing

- Many different interfaces
- Writing fast code for heterogeneous hardware

  = using the right combination of interfaces
- **Can we build a compiler that selects appropriate library interfaces fo us?**

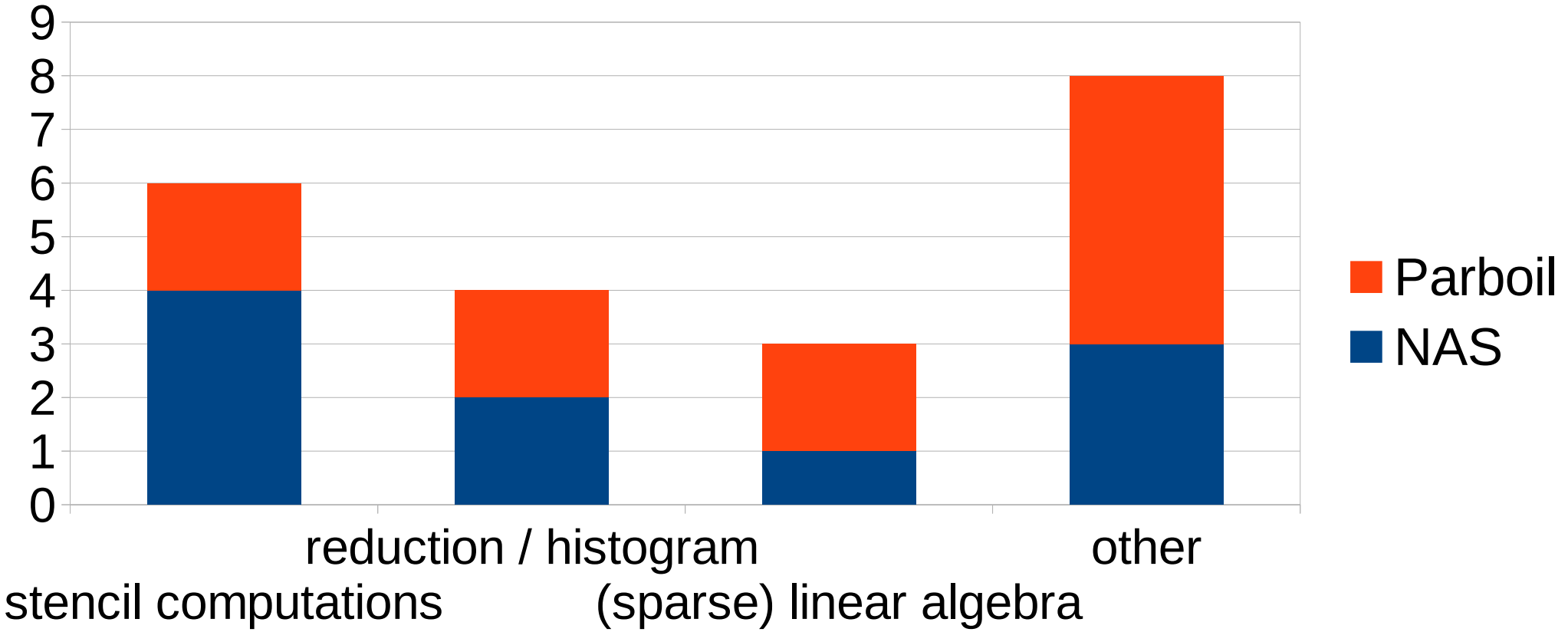
Research Objective

- Idea: compiler recognises functionality and replaces it with library function calls



Benchmark Study

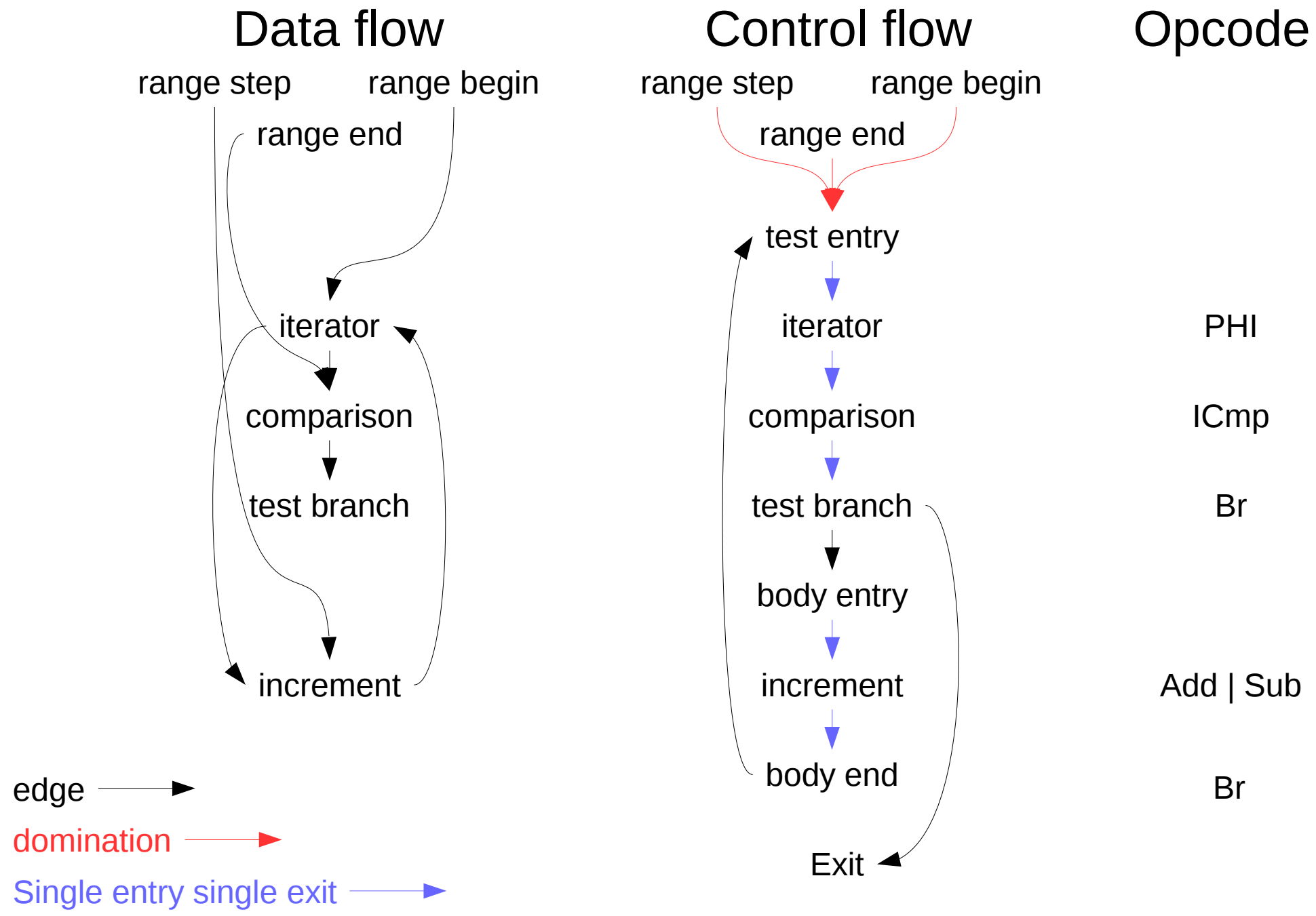
- What functionality do we need to recognise?
- 3 computational patterns cover 60% of benchmark bottlenecks



Algorithm Detection

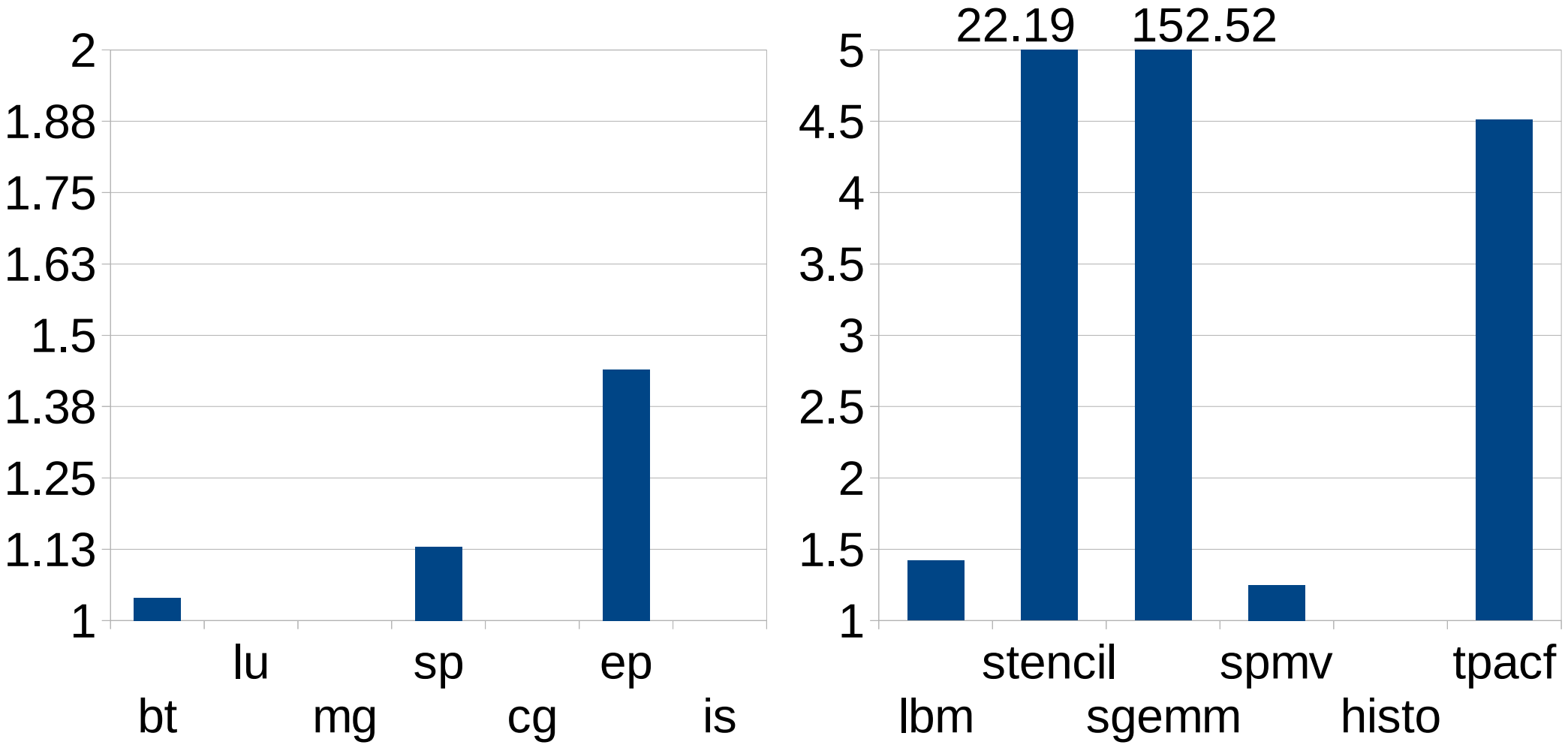
- Compiler needs to recognise linear algebra, reductions, stencil computations
- “Pattern matching” on control flow and data flow
- Formalize algorithmic patterns as constraints on CFG, DFG

Algorithm Detection



Results

- Speedup on 60% of selected benchmarks



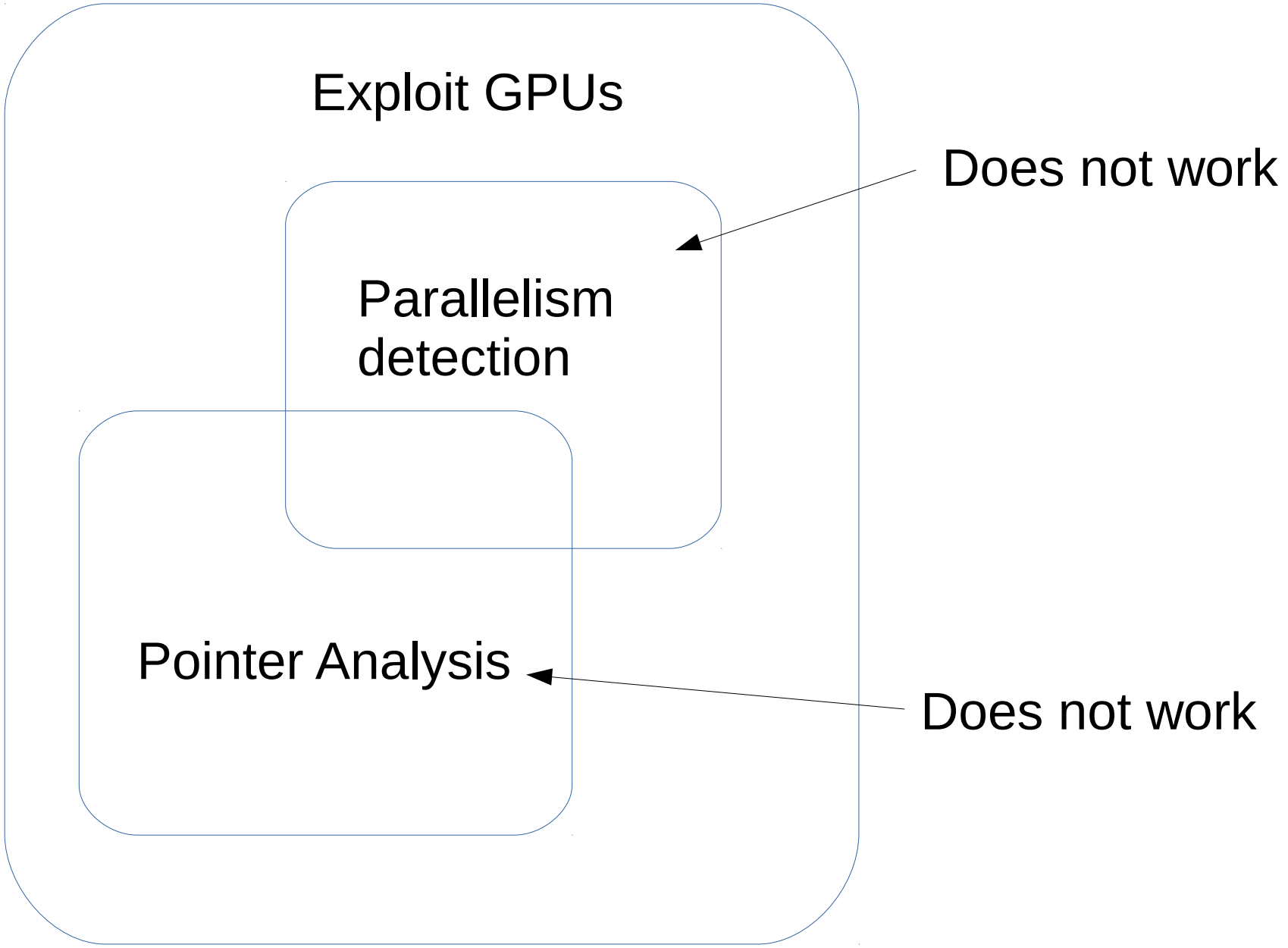
Outlook

- Finalise implementation of the 3 patterns
- Run on additional benchmarks
  - Did we “overfit”?
- Probabilistic models:
  - Choose between several potential implementations

Learn from failures

- Very hard problem
- Other people have tried
  - Example: automatic C→CUDA translation
  - Disappointing results
- Most keep away from compilers
  - libraries
  - domain specific languages

Exploit heterogeneous hardware



Case Study

```
struct Histogram { unsigned bin[256]; };
histogram char_hist(unsigned nc, unsigned char* c)
{
    unsigned  i;
    Histogram hist = empty_histogram();

    for(i = 0; i < nc; i++) {
        hist.bins[c[i]] ++;
    }
    return hist;
}
```

- Inherently parallel
  - Reduction operation on an array
- Huge problem for automatic tools
  - Parallelism is obscured by indirect access
  - Naive exploitation (atomic increment) is horrible

Case Study

- Easy example, already two major problems
- Compilers will not be able to figure this out!

:(

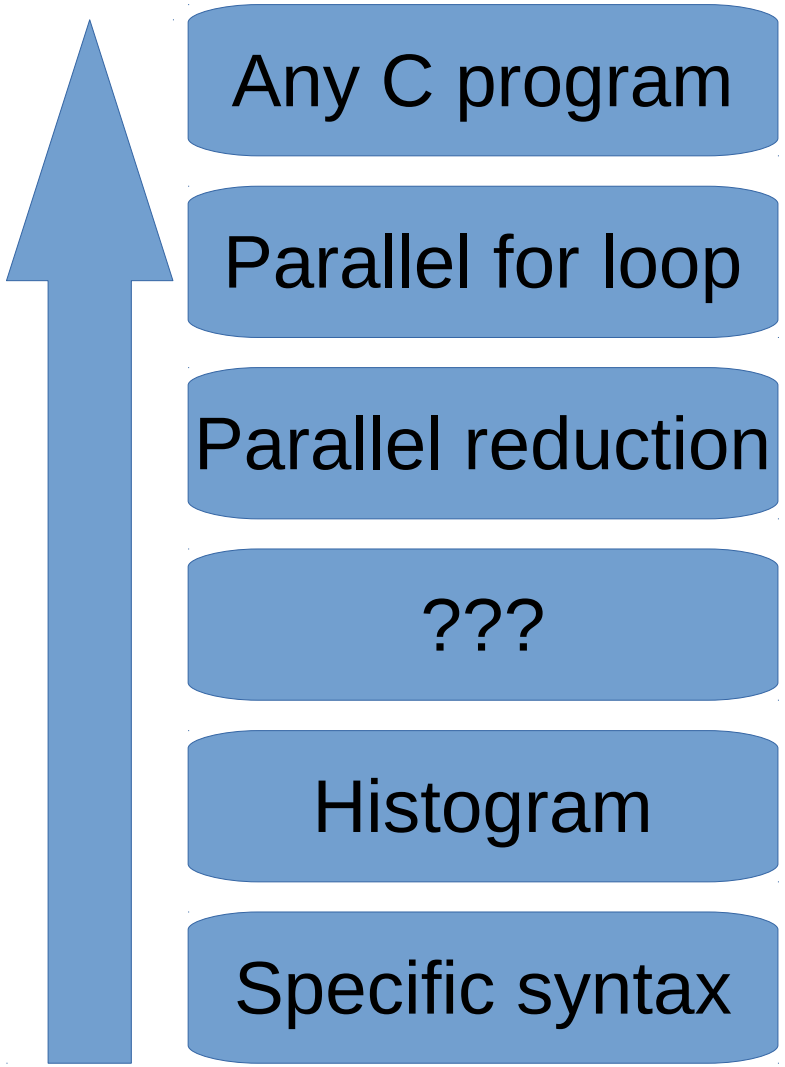
… but wait!

- We know the best solution already
- Compilers just need to apply it

A brute force approach

- We know good histogram implementations
  - Add optimization pass just for histograms
- Pros:
  - Histograms will be fast
- Negatives:
  - Nothing else will be
  - Code coverage almost zero

Bottom-up approach



Generalize to computational pattern

Don't work on source code level

The histogram pattern

```
for(i = 0; i < nc; i++) {
    hist.bins[c[i]] ++;
}

for(i = 0; i < nc; i++) {
    bin = &hist.bins[c[i]];
   *bin = *bin + 1;
}
```

- Generalise c[i]
  - Pure function of i and c[i]
- Generalise *bin + 1
  - Commutative function of *bin, i, c[i]

Work flow

- Identify bottleneck computation
  - Standard benchmark suites
- Find best implementation by hand
  - Established library, domain specific language
- Generalize the computation
  - Computational “pattern”
- Implement custom optimization pass
- repeat

Implementation

- Formalize patterns as constraints in LLVM IR
  - index is pure function of i and c[i]
    - → i and c[i] together dominate index in the DFG
- Pragmatic approach
  - False positives tolerable at this stage
- Working prototype