Autotuning for High Performance Computing

Pedro Bruel

phrb@ime.usp.br

March 27, 2017

Instituto de Matemática e Estatística
Universidade de São Paulo

About

Pedro Bruel
phrb@ime.usp.br

Alfredo Goldman
gold@ime.usp.br

Sai Rahul Chalamalasetti
sairahul.chalamalasetti@hpe.com

Dejan Milojicic
dejan.milojicic@hpe.com

Outline

1. Autotuning
2. Autotuning GPU Compiler Parameters
3. Autotuning High-Level Synthesis for FPGAs
4. Next Steps: Expensive-to-Evaluate Functions

Slides

Slides are hosted at GitHub:

- `github.com/phrb/`

Autotuning: Optimization as a Search Problem

Casting program optimization as a search problem:

Search Spaces:

- Algorithm Selections
- Program Configurations
- ...

Search Objectives:

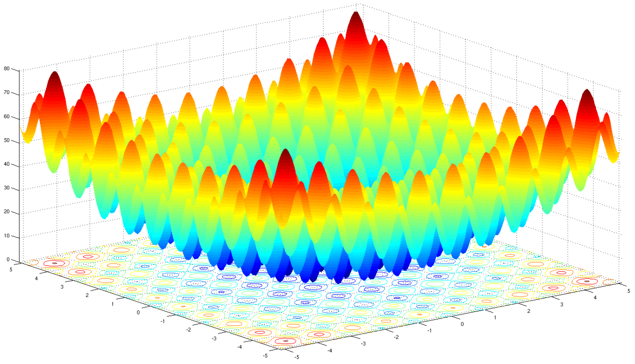
- Minimize execution time
- Maximize usage of resources
- ...

Autotuning: Optimization as a Search Problem

Domain	Search Space	Search Objective
LLVM Compiler	Ordering and Parameters of Optimizations	Size, time, memory, ...
Genetic Algorithms	Configuration Parameters	Time to find solution
Sorting	Size to switch to Insertion Sort	Execution time
Machine Learning	Model selection and parameters	Model accuracy
DSP Algorithms	Algorithm Parameters	Accuracy, execution time, ...

Table 1: Sample of autotuning domains, possible search spaces and objectives

The search spaces created by program optimization problems can be difficult to explore



Rastrigin function, with global minimum $f(0,0) = 0$

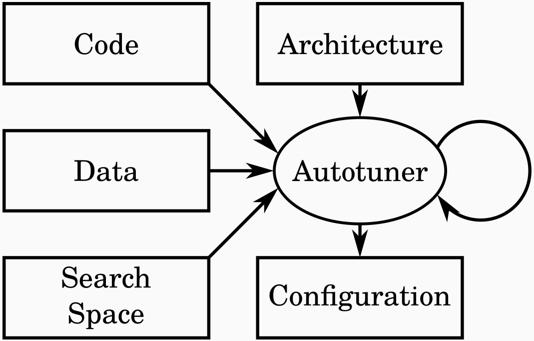
Search Spaces & Techniques

System	Domain	Search Technique
ATLAS	Dense Linear Algebra	Exhaustive
Insieme	Compiler	Genetic Algorithm
SPIRAL	DSP Algorithms	Pareto Active Learning
Active Harmony	Runtime	Nelder-Mead

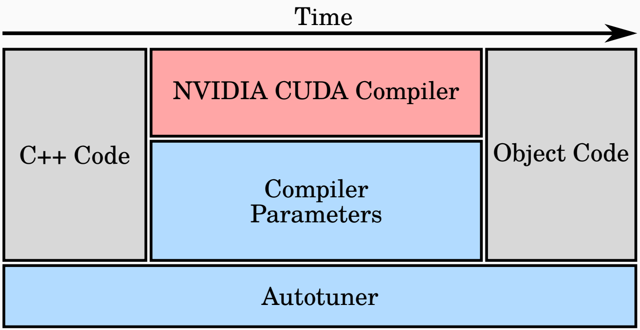
Table 2: Sample of autotuning systems, their domains and techniques

- Different problem domains generate different search spaces
- No single solution for all domains
- Search techniques can be composed: OpenTuner
- Independent searches can be parallelized and distributed

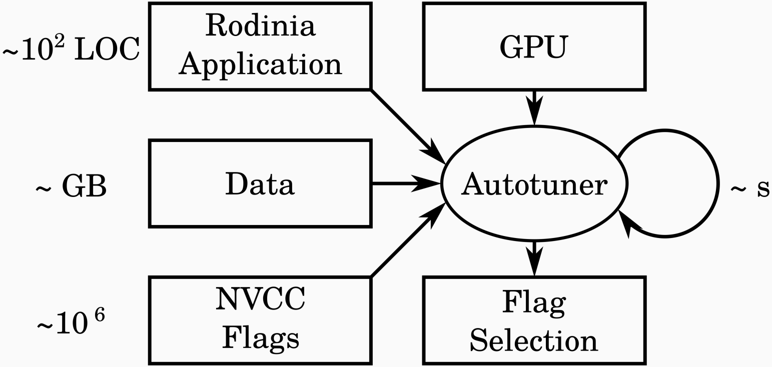
Autotuning: Abstract Model



NVIDIA CUDA Compiler: From CUDA C++ to Object Code

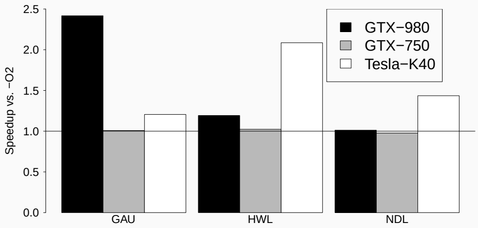


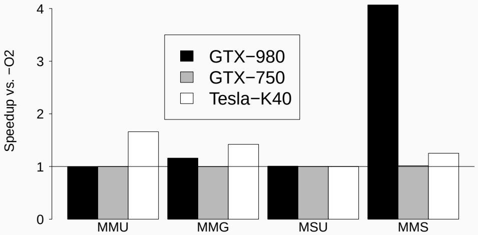
- We tuned applications from the Rodinia Benchmark Suite
- C++ $\rightarrow$ Object Code: takes seconds; up to 4x speedup
- We tuned the parameters of the NVIDIA CUDA Compiler (NVCC)



1h of tuning $\rightarrow \dfrac{3600s}{\sim s} \approx 10^3$ iterations

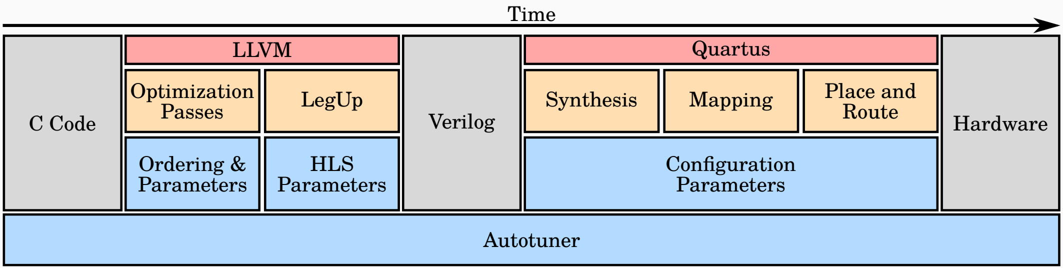
Most significative speedups for Rodinia applications, on the left, and matrix multiplication optimizations, on the right, after 1.5h of tuning:





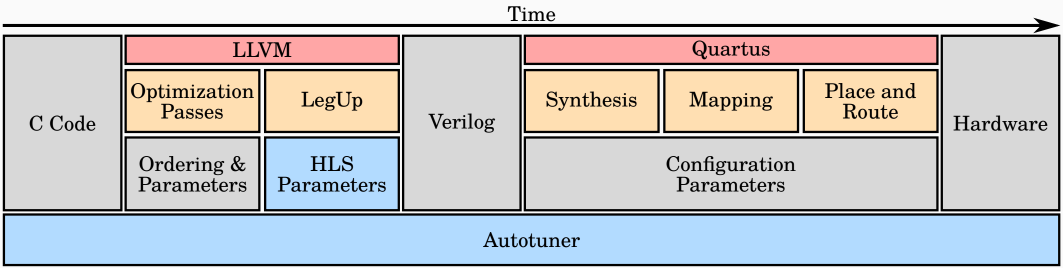
We found no globally good parameter selections for specific GPUs or applications

High-Level Synthesis for FPGAs: From C to Hardware



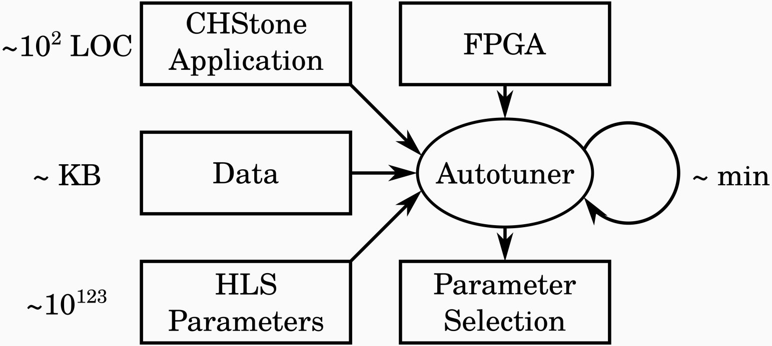
- We tuned applications from the CHStone Benchmark Suite
- C $\rightarrow$ Verilog: takes seconds; ~16% speedup
- Verilog $\rightarrow$ Hardware: takes minutes, hours; 10%-2x speedup
- We tuned C $\rightarrow$ Verilog, but had to pay the cost of Verilog $\rightarrow$ Hardware

High-Level Synthesis for FPGAs: From C to Hardware



- We tuned applications from the CHStone Benchmark Suite
- C $\rightarrow$ Verilog: takes seconds; ~16% speedup
- Verilog $\rightarrow$ Hardware: takes minutes, hours; 10%-2x speedup
- We tuned C $\rightarrow$ Verilog, but had to pay the cost of Verilog $\rightarrow$ Hardware

AUTOTUNING LEGUP PARAMETERS FOR CHSTONE



1h of tuning $\rightarrow \dfrac{60min}{\sim min} \approx$ 10 iterations

Autotuning LegUp Parameters for CHStone

Search Space:

- LegUp constraints that impact Verilog generation
- Read from a configuration file

Examples:

- `set_accelerator_function`
- `ENABLE_PATTERN_SHARING`

Source: `legup.eecg.utoronto.ca/docs/4.0/constraintsmanual.html#constraints` [Accessed on 15/09/16]

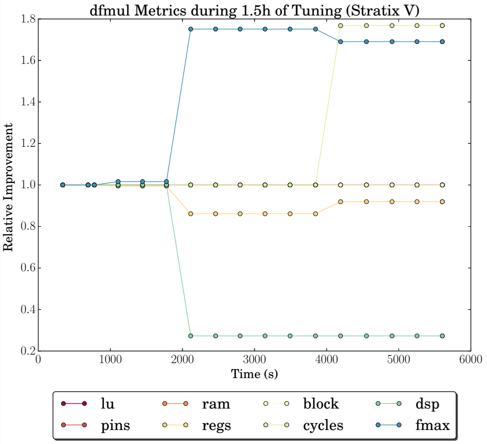
Calculating the fitness function:

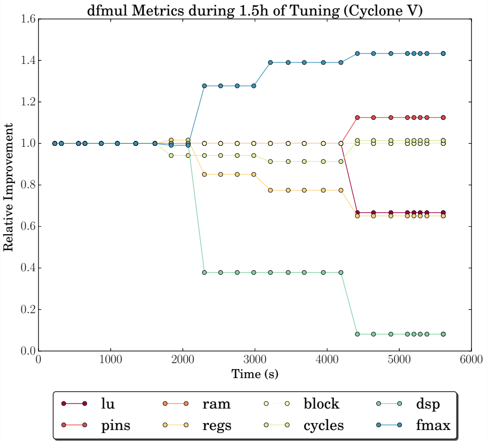
- M: the set of metrics
- W: the set of weights for each metric
- m_i^0: initial measured value for each metric
- $f(M, W)$: cost or fitness function, defined as

$$f(M, W) = \frac{\sum_{\substack{m_i \in M \\ w_i \in W}} w_i \left(\frac{m_i}{m_i^0} \right)}{\sum_{w_i \in W} w_i}$$

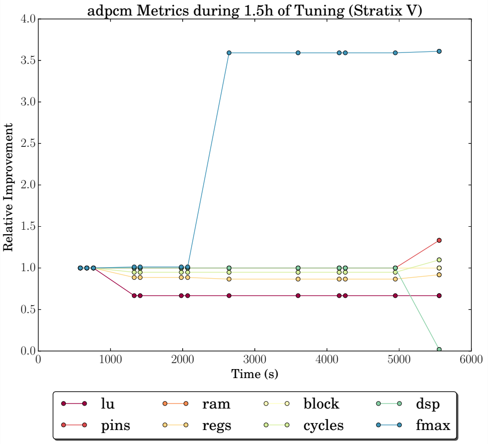
- Naive weights: $w_i = 1, \ \forall w_i \in W$

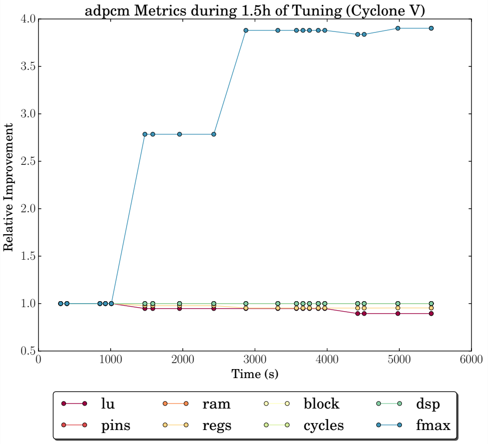
Relative variation for the individual metrics of the dfdiv CHStone application, during 1.5h of tuning:





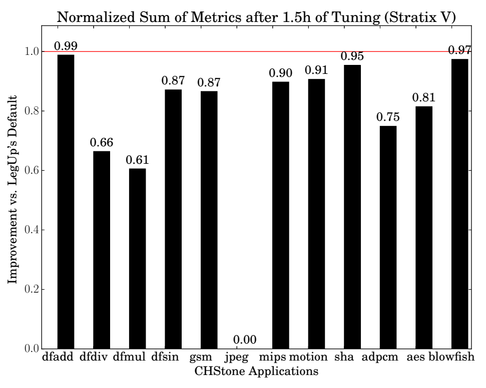
Relative variation for the individual metrics of the adpcm CHStone application, during 1.5h of tuning:

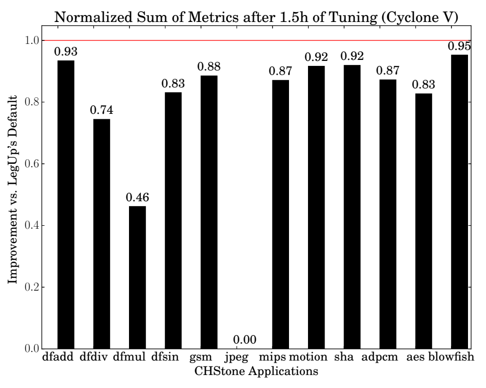




Results

Relative variation for the normalized sum of all metrics of all CHStone application, after 1.5h of tuning:



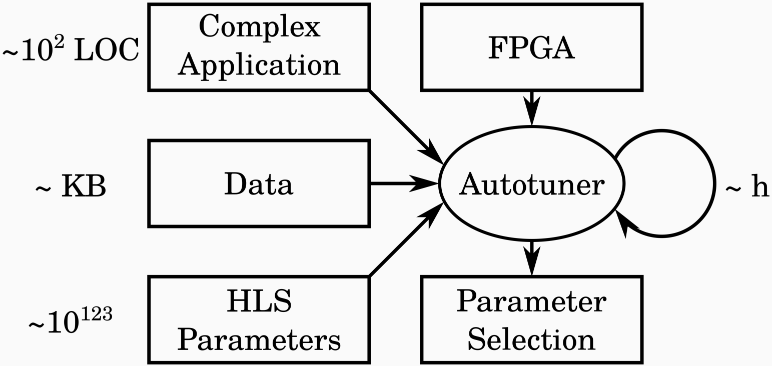


Results: Summary

Autotuning CHStone applications for the Stratix V DE5-Net and Cyclone V DE1-SoC:

- Different relative improvements depending on board and application
- From 1% up to 2x relative improvement on the Normalized Sum of Metrics
- Up to 4x relative increase in frequency
- Different relative improvements for each metric
- Next: Select weights to target specific metrics

Next Steps: Expensive-to-Evaluate Functions



- 1h of tuning $\rightarrow \dfrac{1h}{\sim h} \approx$ 1 iteration
- Iterations now take too long

Next Steps: Expensive-to-Evaluate Functions

Design of Experiments:

- With Arnaud Legrand and Jean-Marc Vincent
- Grenoble Université
- 2nd semester 2017

More future work:

- LLVM Passes for FPGAs
- Genetic Algorithm parameters
- Parallel and Distributed Autotuning

Next Steps: Parallel and Distributed Autotuning

StochasticSearch.jl

Domain-agnostic autotuning framework:

- Julia language
- High Performance, parallel and distributed computing
- Using High-level abstractions
- Expensive-to-evaluate functions
- `github.com/phrb/StochasticSearch.jl`

Autotuning for High Performance Computing

Pedro Bruel

phrb@ime.usp.br

March 27, 2017

Instituto de Matemática e Estatística
Universidade de São Paulo