Autotuning

Introduction to autotuning, overview of our research

Jiří Filipovič et al.
Institute of Computer Science
Masaryk University

Sep. 2021

Program development workflow

Programmer's questions

- which algorithm to use?
- how to implement the algorithm efficiently?
- how to set-up a compiler?

Program development workflow

Compiler's questions

- how to map variables to registers?
- which unrolling factor to use for a loop?
- which functions should be inlined?
- and many others...

Program development workflow

User's questions

- how many computing nodes and threads assign to the program?
- should accelerators be used?
- how to mix MPI and OpenMP threads?

Program development workflow

User's questions

- how many computing nodes and threads assign to the program?
- should accelerators be used?
- how to mix MPI and OpenMP threads?

A compiler works with **heuristics**, people usually too.

Tuning of the program

We can empirically tune these possibilities

- use different algorithm
- change source code optimizations
- use different compiler flags
- execute in a different number of threads
- etc.

Tuning of the program

A tuning allows us to outperform heuristics – we just test what works better.

- however, we have to invest more time into development
- there are non-linear (vertical) dependencies, so we cannot perform tuning steps in isolation
- the optimum usually **depends on hardware and input**

Autotuning

The tuning can be automated

- then we talk about **autotuning**

Autotuning

- in design time, we define the space of *tuning parameters*, which can be changed
- each tuning parameter defines some property of the tuned application
- a search method is used to traverse the space of tuning parameters efficiently
- performed according to some objective, usually performance

Taxonomy of Autotuning

Tuning scope

- what properties of the application are changed by autotuner
- e.g. compiler flags, number of threads, source code optimizations parameters

Tuning time

- off-line autotuning (performed once, e.g. after SW installation)
- dynamic autotuning (performed at runtime)

Developer involvement

- transparent, or requiring only minor assist from developer (e.g., compiler flags tuning)
- low-level, requiring an expert programmer to identify tuning opportunities (e.g. optimizations parameters tuning)

Our focus

Our focus

- source code optimizations parameters
- heterogeneous computing

We target several research questions:

- building framework for dynamic autotuning, which can be integrated into real-world applications
- efficient searching in autotuning spaces
- scheduling of autotuning, integration into task-based systems
- autotuning in higher-order languages

Autotuning framework

Kernel Tuning Toolkit (KTT)

- the source code in CUDA or OpenCL is changed during a tuning process
- the programmer defines how tuning parameters influence the code
- very powerful (source code may control nearly everything)
- implementation is difficult
  - requires recompilation
  - runtime checks of correctness/precision
  - non-trivial expression of tuning parameters
  - we have no implicit assumptions about tuning space
- offline and dynamic autotuning

KTT is a core framework used in other autotuning research.

Performance of our benchmarks library

Benchmark	2080Ti	1070	750	K20	Vega56
BiCG	88.3%	84.7%	81.7%	50.4%	75.6%
Coulomb 3D	91.8%	91.4%	84.3%	43.2%	65.3%
GEMM	79.8%	80.6%	91.1%	51.3%	96.3%
GEMM batched	86.8%	81.4%	90.0%	49.6%	86.0%
Transpose	87.1%	80.2%	86.3%	64.2%	86.1%
N-body	89.7%	86.6%	87.7%	40.6%	82.2%
Reduction	68.7%	87.5%	89.4%	64.1%	71.6%

Table: Performance of benchmarks autotuned for various hardware devices. The performance relative to the theoretical peak of devices.

Performance portability

	GPU→GPU		
Benchmark	avg±stdev	worst	failed
BiCG	89.0%±12.3%	57%	1
Convolution	79.4%±14.9%	55%	3
Coulomb 3D	95.8%±6.5%	67%	0
GEMM	83.6%±16.4%	31%	0
GEMM batched	85.4%±17%	37%	0
Hotspot	80.3%±17.5%	46%	3
Transpose	85.0%±21.9%	8%	3
N-body	78.8%±24.2%	2%	3
Reduction	88.4%±24%	12%	3
Fourier	74.5%±30%	31%	0

Table: Relative performance of benchmarks ported across GPU architectures without re-tuning.

Dynamic autotuning of Batched GEMM

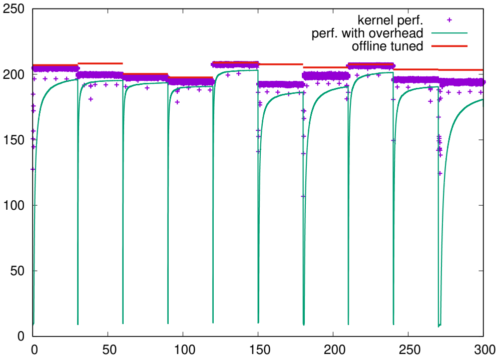

Figure: Batched GEMM on GeForce GTX 1070.

Tuning space searching

Tuning spaces are difficult to search

- non-linear dependencies, many dimensions, discrete space
- random search often superior to mathematical optimization or ML-based methods

Performance-counter based searcher

- ML method learns how tuning parameters influence performance counters
- during tuning space search, expert system navigates search towards softening observed bottlenecks (changes tuning parameter values in order to modify performance counters)
- can be used to navigate tuning search of known application at unseen hardware, unseen input and arbitrary optimizations

We have implemented the first version of profile-based searcher.

Searching speed

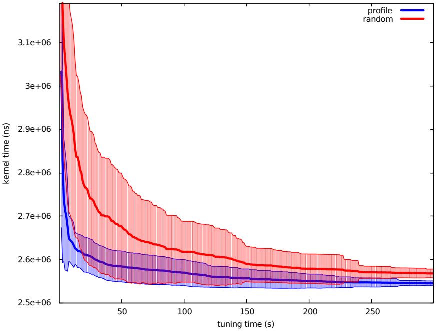

Figure: GEMM, $2048 \times 2048 \times 2048$, GTX 2080, model from GTX 1070

Searching speed

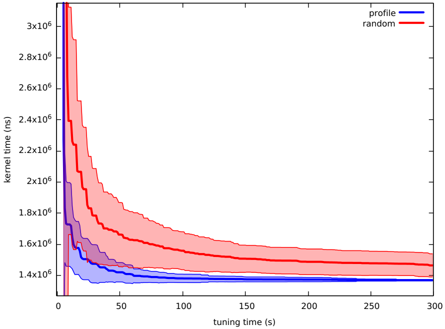

Figure: Matrix transposition, 8192×8192, GTX 2080, model from GTX 1070

Searching speed

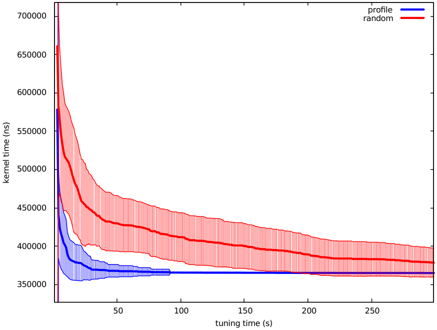

Figure: Convolution, 4096 × 4096, GTX 2080, model from GTX 1070

Autotuning in task-based system

When and where to run auto-tuning

- tuning space search has overhead, so it does not always improve speed
- depends on how many times we execute tuned code
- depends on how much performance we can get, and how long time we need to invest into tuning
  - analysis of performance counters and historical data
- becomes increasingly complicated when we have more processors and accelerators

Autotuning in task-based systems

- task-based system distribute data-dependent computing tasks in hererogeneous node(s)
- those tasks can be inherently auto-tuned
- challenging in scheduling and SW engineering

We have prototype of KTT integrated into StarPU.

Autotuning in higher-order languages

Autotuning with KTT requires expert-programmers

- CUDA/OpenCL programming is difficult
- identification of relevant tuning parameters requires intimate knowledge of hardware

Higher-order languages ease programming

- often at the cost of code efficiency
- but higher-order functions can be tuned according to HW, input and lower-order function, such tuning is transparent for the programmer
- for fine-tuning, we can also expose tuning parameters for lower-order functions

We are currently experimenting with integration of KTT into Thrust (skeletal programming).