Technische Universität Berlin

LLVM Berlin Meetup

Auto-tuning Compiler Transformations with Machine Learning

Mozilla Berlin Community Space | November 30, 2017

Dr. **Biagio Cosenza**
Embedded Systems Architecture Group
Faculty IV EECS, TU Berlin

In collaboration with Ben Juurlink, Angela Pohl, Daniel Maier, Nikita Popov (TU Berlin), Stefano Ermon (Stanford University), Thomas Fahringer, Klaus Kofler, Ivan Grasso (University of Innsbruck), Juan Durillo (Leibniz Supercomputing Centre)

Outline

- Why automatic tuning
- Auto-tuning with machine learning
  - Four challenges
- Auto-tuning by example
  - Classification, for heterogenous task partitioning
  - Regression, for vectorization cost model
  - Ordinal regression, for stencil computations
- Conclusion
  - Auto-tuning and programming models
  - Importance of structural approaches

Why Automatic Tuning (1)

- Simple example: loop unrolling

```
for(int i=0;i<1000;i++){
  a[i] = b[i] + c[i];
}
```

unroll factor 2

```
for(int i=0;i<1000;i+=2){
  a[i]   = b[i]   + c[i];
  a[i+1] = b[i+1] + c[i+1];
}
```

- What is the best loop unrolling factor?
  - Transformation space is small
  - Prediction is still challenging

> Mark Stephenson, Saman P. Amarasinghe:
> Predicting Unroll Factors Using Supervised Classification. CGO 2005: 123-134

Why Automatic Tuning (2)

- Example: six-point von Neumann stencil

```
for(int t=1; t<nt; t++)
 for(int x=0; x<nx; x++)
  for(int y=0; y<ny; y++)
   for(int z=0; z<nz; z++)
   {
   out[x,y,z; t] =
     in[x-1,y,z;t-1] + in[x,y+1,z;t-1] +
     in[x+1,y,z;t-1] + in[x,y,z-1;t-1] +
     in[x,y-1,z;t-1] + in[x,y,z+1;t-1];
   }
```

For each time step **`t`**

For each cell **`(x,y,z)`**

One write:
the element **`(x,y,z)`**
at time **`t`**

We call the read-point pattern
stencil shape

Some stencils have reads on older time steps: **`t-2, t-3,`** …

Why Automatic Tuning (3)

- Stencil computation

```
for(int t=1; t<nt; t++)
 for(int x=0; x<nx; x++)
  for(int y=0; y<ny; y++)
   for(int z=0; z<nz; z++)
   {
   out[x,y,z; t] =
     in[x-1,y,z;t-1] + in[x,y+1,z;t-1] +
     in[x+1,y,z;t-1] + in[x,y,z-1;t-1] +
     in[x,y-1,z;t-1] + in[x,y,z+1;t-1];
   }
```

Blocking on **(x,y,z)**

Unrolling the innermost loop

Multi-threading + SIMD
(**chunk** number of consecutive tiles)

➢ Transformation space is large (~16K configurations) and complex (i.e., with discontinuities)

Autotuning Stencil Computations with Structural Ordinal Regression Learning.
Cosenza, Durillo, Ermon, Juurlink. IPDPS 2017

Why Automatic Tuning (4)

Project	Benchmark	Possible configurations
PetaBricks	Poisson	10^{3657}
gcc/g++ flags	all	10^{806}
Halide	Bilateral	10^{176}
PetaBricks	Sort	10^{90}
Halide	Blur	10^{25}
Unitary	n/a	10^{21}
Stencil/OpenTuner	all	$10^{6.5}$
Stencil/Patus*	all	10^{4}

PetaBricks: A Language and Compiler for Algorithmic Choice.
Ansel, Chan, Wong, Olszewski, Zhao, Edelman, Amarasinghe. PLDI 2009

OpenTuner: An Extensible Framework for Program Autotuning.
Ansel, Kamil, Veeramachaneni, Ragan-Kelley, Bosboom, O'Reilly, Amarasinghe. PACT 2014

Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines
Ragan-Kelley, Barnes, Adams, Paris, Durand, Amarasinghe. PLDI 2013

Desiderata for Automatic Tuning

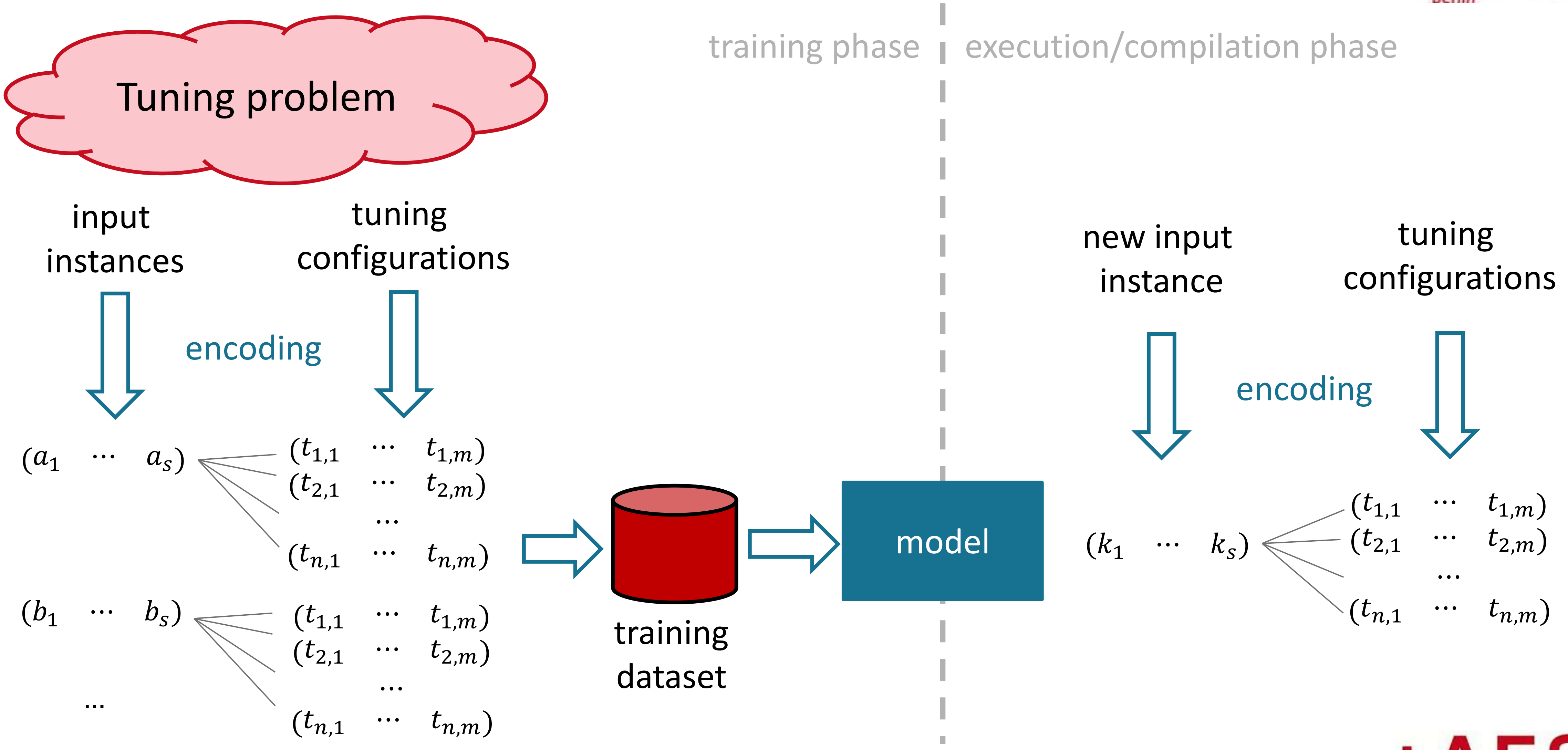
- Ideally, we would like to have autotuners that are
  - Fast, to be integrated into common compilers
  - Accurate, to deliver good, close-to-peak solutions
  - Flexible, to adapt to any possible input problem
  - Portable, to target any hardware
- Traditional approaches
  - Analytical model
    - generic, but hard to build, require domain expertise
    - far from peak performance
  - Iterative-compilation with search heuristics
    - accurate solution, but long compilation time
    - heuristics: genetic algorithms, differential evolution, …

Autotuning with Machine Learning

- Autotuning with Machine Learning: Supervised Machine Learning
  - Build a model in a preprocessing stage, later reuse the model for a new a new input
  - Fast: can be used in compilers (i.e., fast compilation time)
  - Portable: just build a new model for a new hardware/platform
- Machine learning is already successful in different fields
  - Image recognition, speech recognition, NLP
- However
  - Compilers and software optimization present some unique challenges
  - Existing methods do not apply so well
    - Too little data for deep neural networks
    - Training data has different structure

We need fundamentally new approaches

Autotuning with Supervised Learning



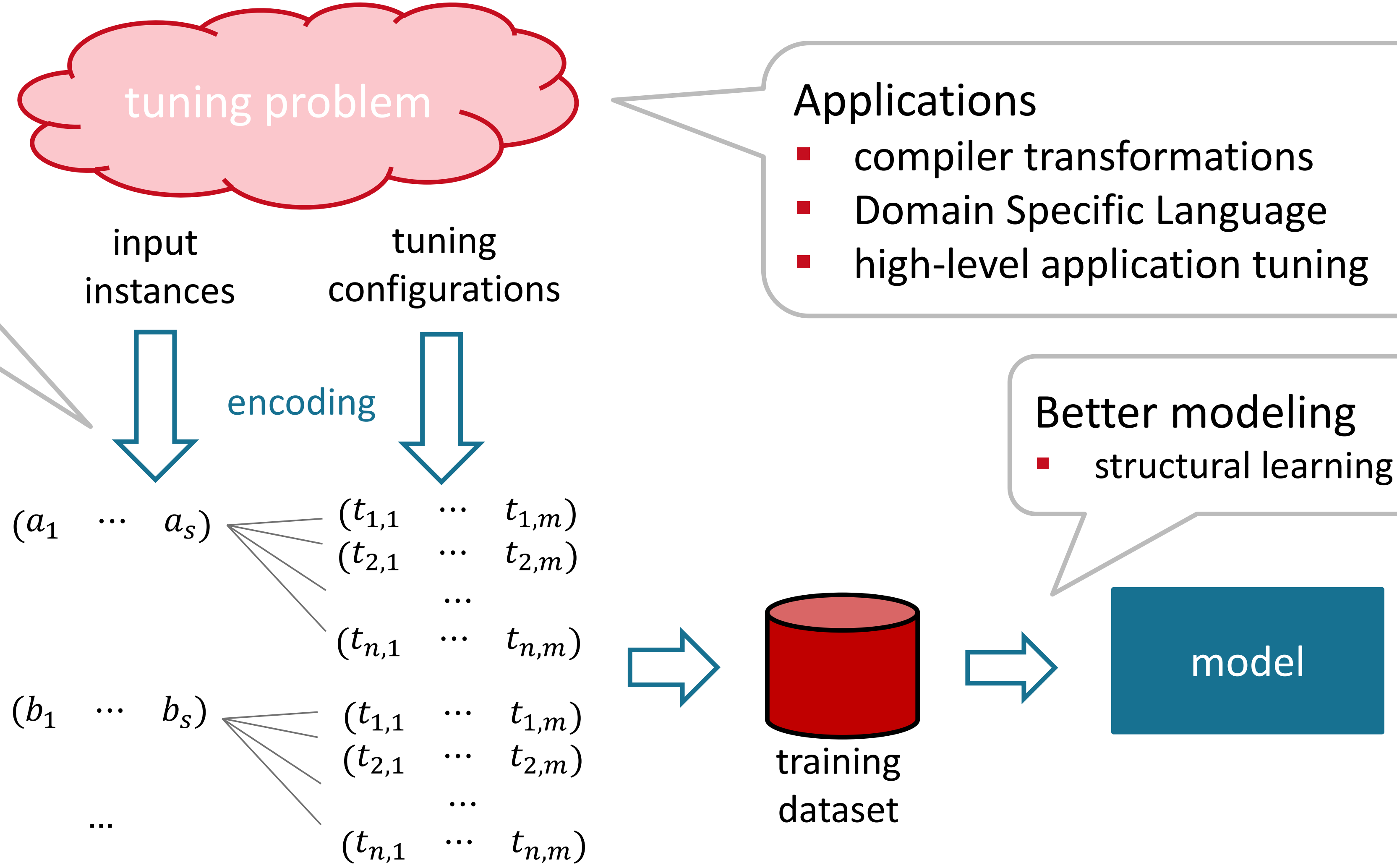
Four Research Aspects of ML-based Autotuning

Better problem-specific encoding
- tree
- graph
- polyhedron

More & better training data
- synthetic code generation
- space pruning

Applications
- compiler transformations
- Domain Specific Language
- high-level application tuning

Better modeling
- structural learning



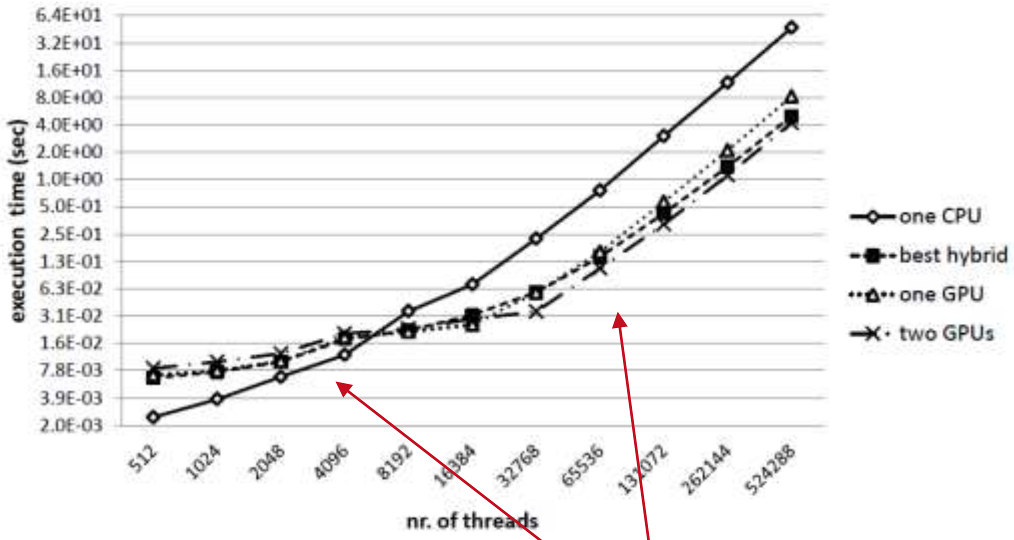
Autotuning by Example

- Three examples
  - Working on real compiler infrastructure
  - Using different modeling
  - Tuning different compiler transformations

1. Classification
   - For automatic task partitioning
2. Regression
   - For vectorization cost model in LLVM
3. Ordinal Regression
   - For stencil tuning computations

Automatic Heterogenous Task Partitioning

- Problem: Task Partitioning on Heterogenous Device
  - Hardware: One multi-core CPU, two GPUs
  - Where do run partition our OpenCL task?

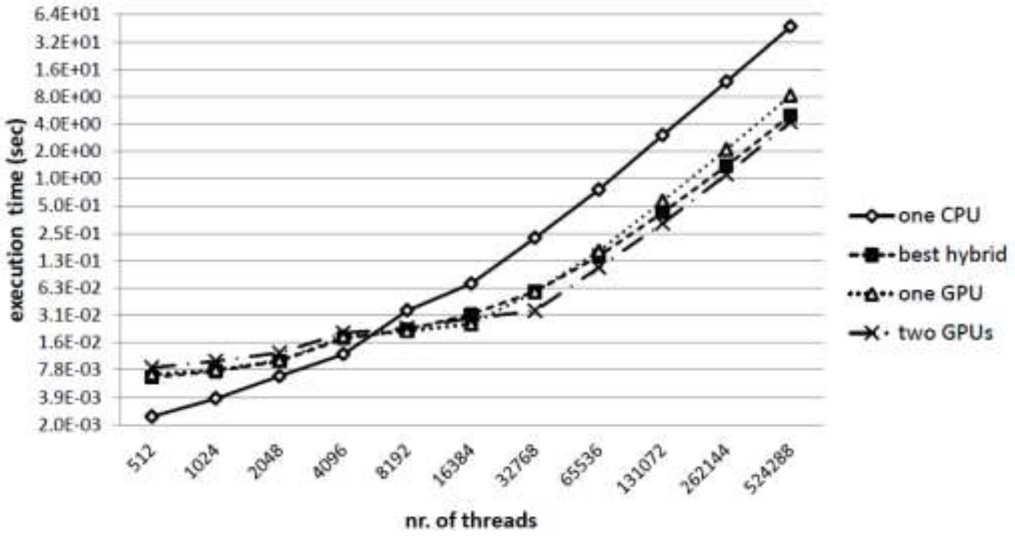


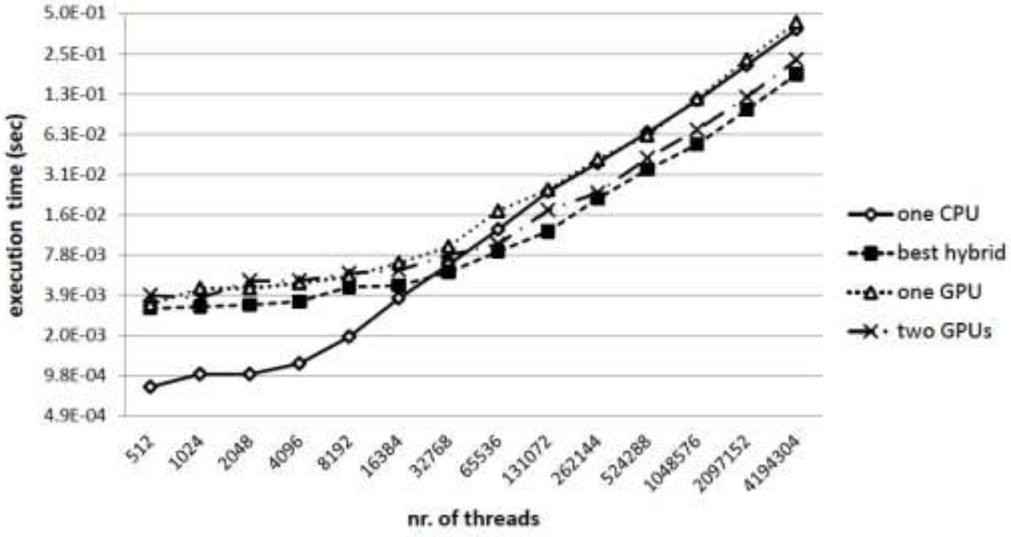
same hardware, same code, different size

Code: linear regression kernel
Hardware: CPU AMD 2x Opteron 6168 + 2x Radeon HD 5870

Automatic Heterogenous Task Partitioning

- Problem: Task Partitioning on Heterogenous Device





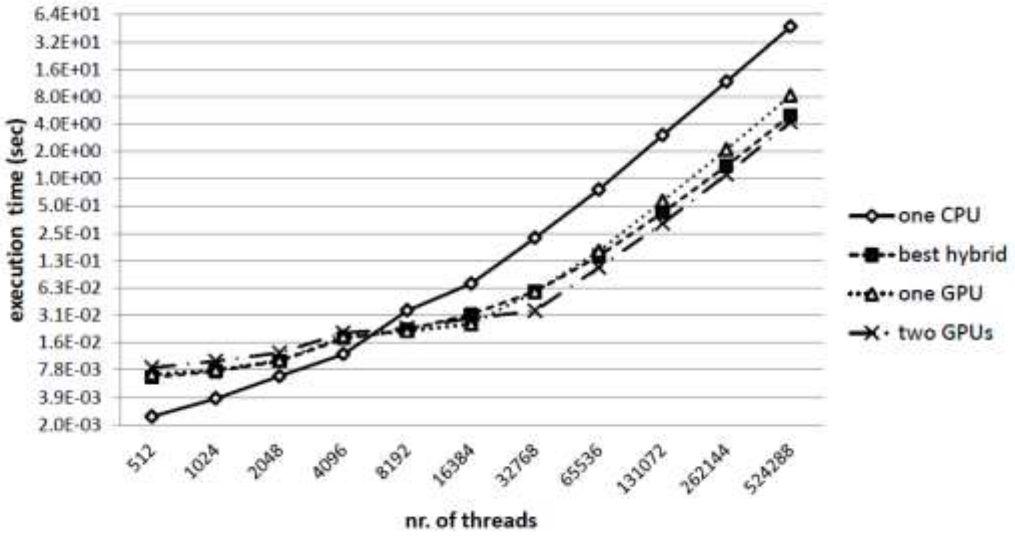
Code: linear regression kernel
Hardware: CPU AMD 2x Opteron 6168 + 2x Radeon HD 5870

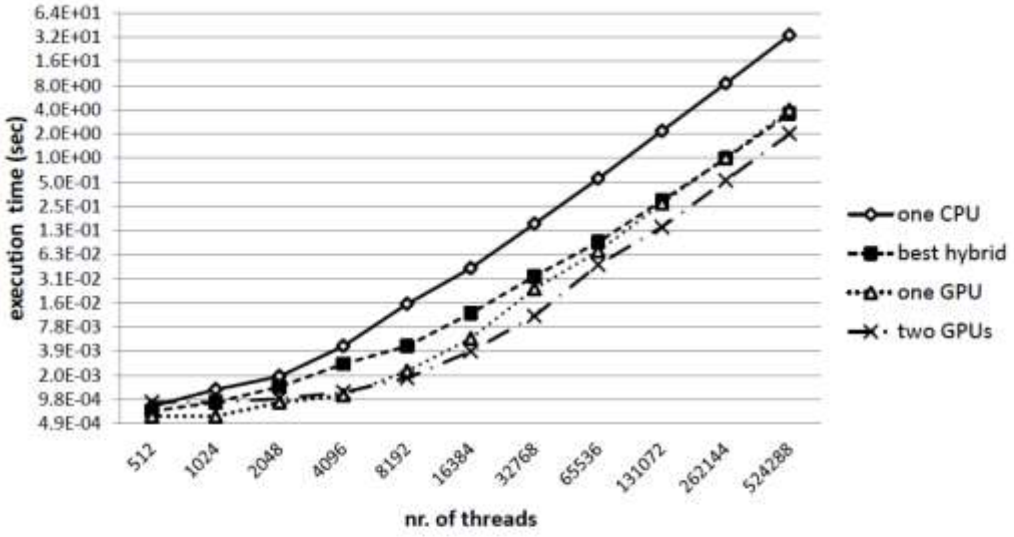
Code: reduction kernel
Hardware: CPU AMD 2x Opteron 6168 + 2x Radeon HD 5870

same hardware, different code, same size

Automatic Heterogenous Task Partitioning

- Problem: Task Partitioning on Heterogenous Device





Code: linear regression kernel
Hardware: CPU AMD 2x Opteron 6168 + 2x Radeon HD 5870

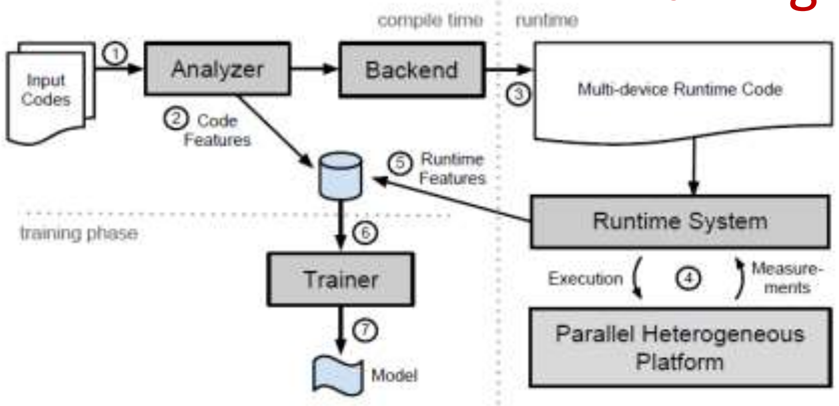
Code: linear regression kernel
Hardware: CPU Intel 2x Xeon X5650 + 2x NVIDIA GeForce GTX480

different hardware, same code, same size

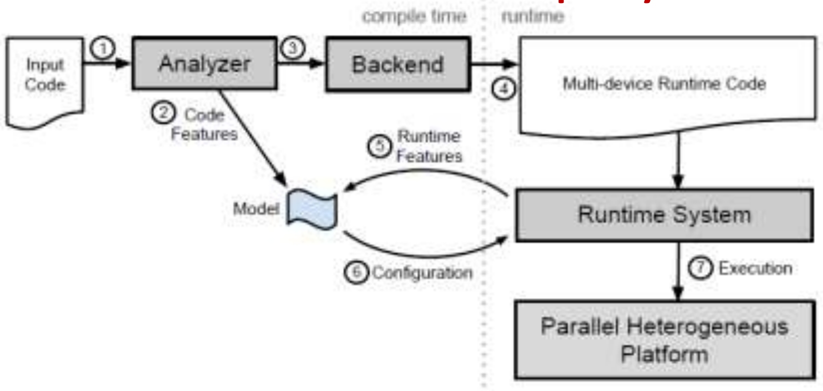
Automatic Heterogenous Task Partitioning

- Difficult problem
  - Depends on the code, the hardware and the (input) size
- What about heterogenous partitioning?
  - Support hybrid partitioning
  - Example: 20% on CPU, 40% on GPU1, 40% on GPU2
- Solution: machine learning
  - Training phase: build a partitioning model for a specific hardware
  - Deployment phase: infer the model to select the best-performing partitioning

Training



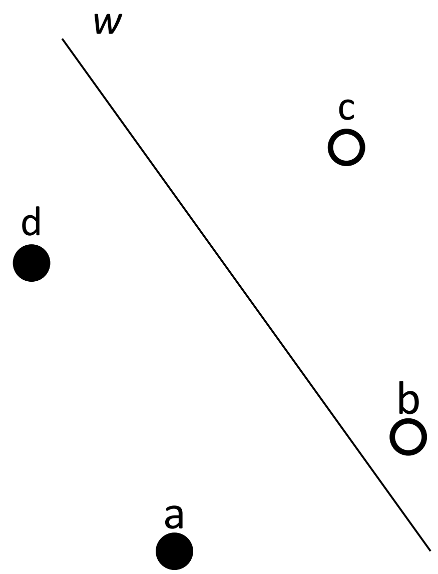
Deployment



Automatic Task Partitioning: Modeling

- Partitioning as classification
  - Classes are partitioning, e.g.
    - (20,40,40) means 20% CPU, 40% each GPU
    - (100,0,0) CPU only
    - (0,0,100) one GPU
    - (0,50,50) two GPUs
  - Overall 21 classes
- Classification algorithms
  - Support Vector Machine (SVM)
  - Artificial Neural Network (ANN)

SVM Classification



Automatic Task Partitioning: Features

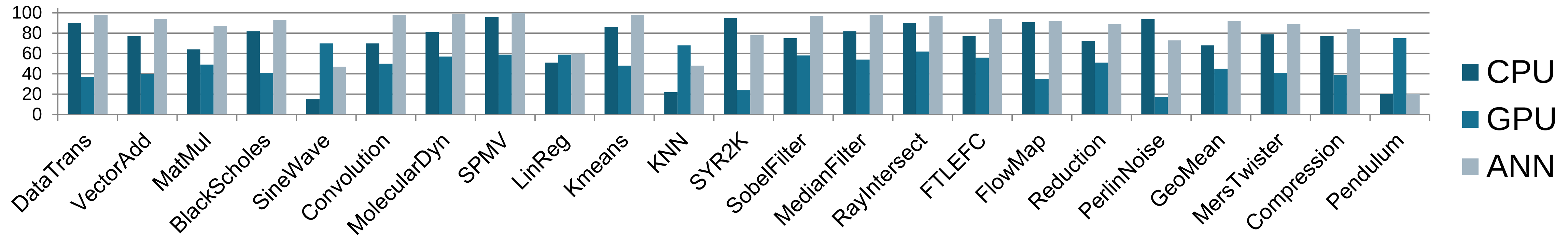
- 24 static features extracted from INSPIRE
  - INSieme Parallel Intermediate Representation
  - Examples: # of builtin, # of branches, # of scalar float op., # of vector float op.
- 9 dynamic features extracted at runtime
  - Examples: number of global work items, read and write buffer size
- PCA on static features
- Training on 23 test programs, each executed with different sizes
- Compiler infrastructure
  - Insieme compiler with OpenCL frontend / backend

    An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning.
    Kofler, Grasso, Cosenza, Fahringer. ACM ICS 2013

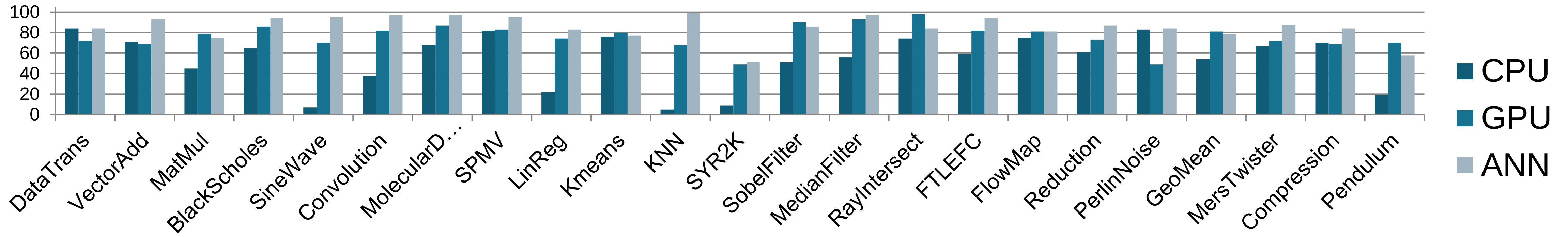
  - Currently reproducing on SYCL with a LLVM-based compilation infrastructure

Automatic Task Partitioning: Results

- 86% accuracy (ANN + PCA) on *mc1*, leave-one-out cross validation



- 88% accuracy (ANN + PCA) on *mc2*, leave-one-out cross validation



Feature Analysis with Greedy Feature Selection

- Portable approach: different platforms exploit different features

Rank	Static program features	MSE
2	OpenCL built-in functions	76.3
3	Number of branches / number of statements	64.4
4	Scalar float operations /number of statements	61.1

Rank	Dynamic runtime features	MSE
1	Data transfer size for splittable buffer (device to host)	99.7
5	Number of global work items	60.0
6	Data transfer size for splittable buffer (host to device)	47.6
7	Runtime feature #6 / total number of arithmetic operations	47.5

Rank	Static program features	MSE
1	Number of branches / number of statements	91.6
2	Scalar float operations / number of statements	75.8
4	OpenCL built-in functions / number of statements	66.9
6	Scalar int operations / number of statements	56.6
7	Vector float operations / number of statements	52.2
8	Number of loops / number of statements	48.6
9	Scalar int operations	47.5
10	Scalar float operations	46.9

Rank	Dynamic runtime features	MSE
3	Data transfer size for splittable buffer (host to device)	69.6
5	Data transfer size for splittable buffer (device to host)	64.0

An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning. Kofler, Grasso, Cosenza, Fahringer. ACM ICS 2013

Automatic Vectorization

```
for (int i = 0; i < N; i++){
  a[i] = b[i] + k;
}
```

Automatic Vectorization

```
for (int i = 0; i < N; i++){
  a[i] = b[i] + k;
}
```

x86-64 clang 5.0.0 -O0
https://godbolt.org/g/orjryg

```
mov dword ptr [rbp - 24], 0
.LBB0_1: # =>This Inner Loop Header: Depth=1
cmp dword ptr [rbp - 24], 1024
jge .LBB0_4
# BB#2: # in Loop: Header=BB0_1 Depth=1
mov rax, qword ptr [rbp - 16]
movsxd rcx, dword ptr [rbp - 24]
movss xmm0, dword ptr [rax + 4*rcx] # xmm0=mem[0],0,0,0
addss xmm0, dword ptr [rbp - 20]
mov rax, qword ptr [rbp - 8]
movsxd rcx, dword ptr [rbp - 24]
movss dword ptr [rax + 4*rcx], xmm0
# BB#3: # in Loop: Header=BB0_1 Depth=1
mov eax, dword ptr [rbp - 24]
add eax, 1
mov dword ptr [rbp - 24], eax
jmp .LBB0_1
```

x86-64 clang 5.0.0 -O3 (loop tail omitted)
https://godbolt.org/g/xttYbr

```
shufps xmm0, xmm0, 0 # xmm0 = xmm0[0,0,0,0]
xor eax, eax
.LBB0_5: # =>This Inner Loop Header: Depth=1
movups xmm1, xmmword ptr [rsi + 4*rax]
movups xmm2, xmmword ptr [rsi + 4*rax + 16]
addps xmm1, xmm0
addps xmm2, xmm0
movups xmmword ptr [rdi + 4*rax], xmm1
movups xmmword ptr [rdi + 4*rax + 16], xmm2
movups xmm1, xmmword ptr [rsi + 4*rax + 32]
movups xmm2, xmmword ptr [rsi + 4*rax + 48]
addps xmm1, xmm0
addps xmm2, xmm0
movups xmmword ptr [rdi + 4*rax + 32], xmm1
movups xmmword ptr [rdi + 4*rax + 48], xmm2
add rax, 16
cmp rax, 1024
jne .LBB0_5
```

Vectorization Cost Modeling

- Automatic vectorization
  - From scalar to vector (packed) instructions
- Two approaches
  - Loop-level vectorization (LLV)
  - Superword Level Parallelism (SLP)
- Two questions
  - Is vectorization allowed?
  - Is vectorization beneficial?
    - Here good modeling is required
- Problem: predicting performance improvement (speedup) after vectorization
  - Used by loop vectorizer and SLP vectorizer
  - Solution: modeling as regression problem

Stencil Computation

- Example: six-point von Neumann stencil

```
for(int t=1; t<nt; t++)
 for(int x=0; x<nx; x++)
  for(int y=0; y<ny; y++)
   for(int z=0; z<nz; z++)
   {
   out[x,y,z; t] =
     in[x-1,y,z;t-1] + in[x,y+1,z;t-1] +
     in[x+1,y,z;t-1] + in[x,y,z-1;t-1] +
     in[x,y-1,z;t-1] + in[x,y,z+1;t-1];
   }
```

For each time step `t`

For each cell `(x,y,z)`

One write:
the element `(x,y,z)`
at time `t`

We call the read-point pattern
stencil shape

Some stencils have reads on older
time steps: `t-2, t-3, …`

Stencil Code Tuning

- Tunable transformations in the Patus stencil compiler [Christen et al., SC 12]

```
for(int t=1; t<nt; t++)
 for(int x=0; x<nx; x++)
  for(int y=0; y<ny; y++)
   for(int z=0; z<nz; z++)
   {
   out[x,y,z; t] =
     in[x-1,y,z;t-1] + in[x,y+1,z;t-1] +
     in[x+1,y,z;t-1] + in[x,y,z-1;t-1] +
     in[x,y-1,z;t-1] + in[x,y,z+1;t-1];
   }
```

Blocking on (`x,y,z`)

Unrolling the innermost loop

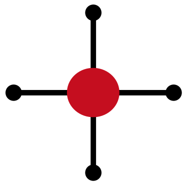
Multi-threading + SIMD
(**chunk** number of consecutive tiles)

Stencil Auto-tuning

- Common approach: search-based iterative compilation
  - long compilation time for each code, but converges to a good solution
- Machine learning
  - machine-dependent model (in preprocessing phase)
  - then reuse it for new input stencil codes: fast compilation time
- Our approach: exploit structure of the problem (structural learning)
  1. Encode stencils and generate synthetic training set
  2. Build machine learning model using ordinal regression SVMs
  3. For new stencil code, use model to rank configurations and select best one

Stencil Encoding Example: Five-point Laplacian Stencil

- Stencil features
  - Shape

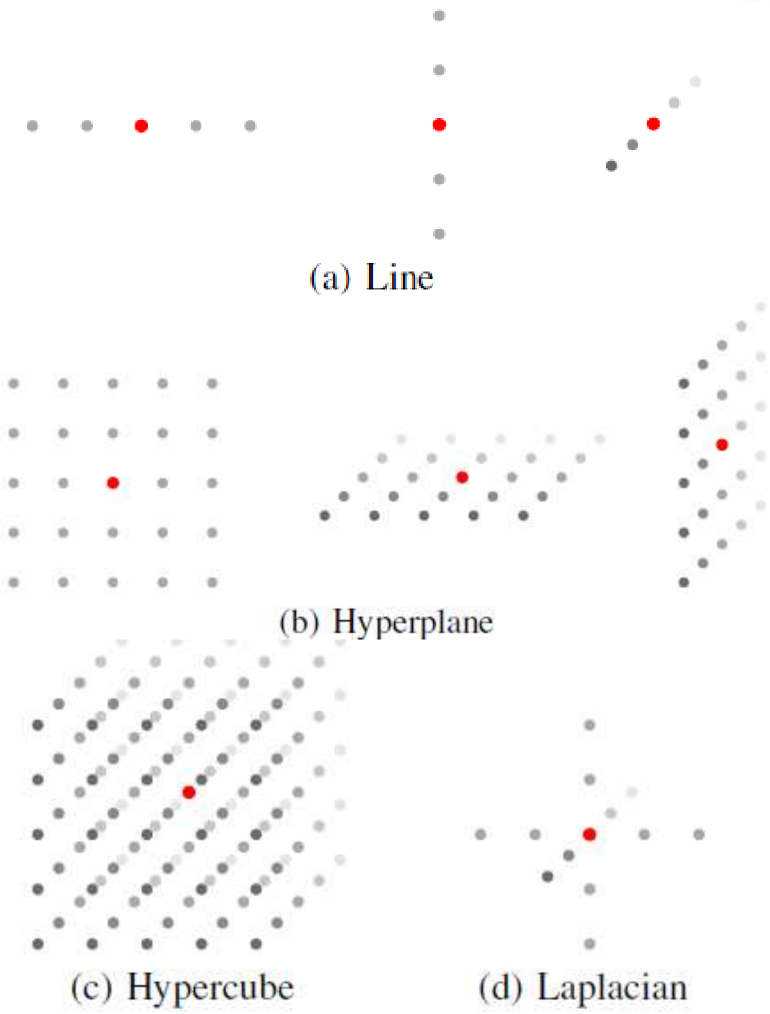


$$\begin{bmatrix} p_{(-1,1)} & p_{(0,1)} & p_{(1,1)} \\ p_{(-1,0)} & p_{(0,0)} & p_{(1,0)} \\ p_{(-1,-1)} & p_{(0,-1)} & p_{(1,-1)} \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

  - Number of buffers
  - Data type (0 for float, 1 for double)
  - Input size
- Tuning features
  - Blocking size
  - Chunking size
  - Unrolling factor

Synthetic Training Set Generation

- A code generator generates stencil codes with different
  - shapes: line, hyperplane, hypercube and Laplacian
  - number of buffers
  - buffer types
- The generated stencil codes are executed with different
  - input sizes
  - tuning parameters



(a) Line

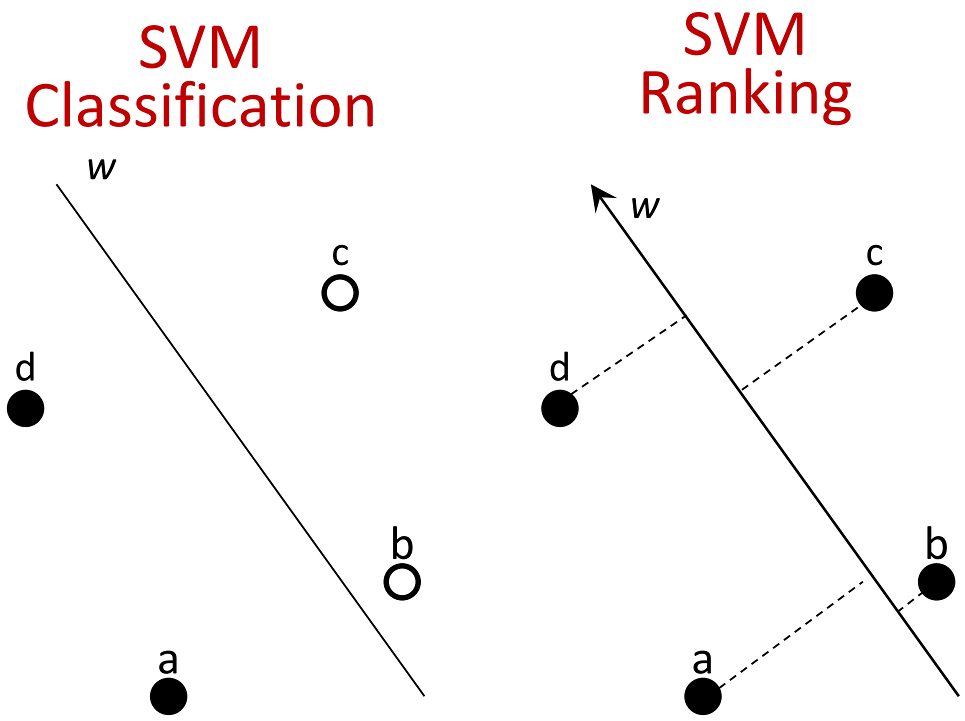
(b) Hyperplane

(c) Hypercube

(d) Laplacian

Modeling with Ordinal Regression

- Classification
  - Training set in terms of classes
  - Problem with large transformation space
- Regression
  - Training set as numerical performance values
  - Difficult problem: performance prediction
- Ordinal Regression
  - Training set as partially ordered set
  - Prediction through ranking function
  - We select the top-ranked transformation



Training Set as Rankings

- The training set arranged in terms of (partial) rankings
- From ranking to inequalities
  - transitive inequalities are omitted

stencil instance: kernel shape + sizes

tuning setting (unroll factor, block size, chunk)

Execution	Input	Tuning	Runtime	Rank
1	$q_1 = (k_1, s_1)$	t_{e_1}	$12ms$	1
2	$q_1 = (k_1, s_1)$	t_{e_2}	$13ms$	2
3	$q_1 = (k_1, s_1)$	t_{e_3}	$20ms$	3
4	$q_2 = (k_1, s_2)$	t_{e_4}	$10ms$	1
5	$q_2 = (k_1, s_2)$	t_{e_5}	$36ms$	3
6	$q_2 = (k_1, s_2)$	t_{e_6}	$35ms$	2
7	$q_3 = (k_2, s_1)$	t_{e_7}	$30ms$	1
8	$q_3 = (k_2, s_1)$	t_{e_8}	$45ms$	2
9	$q_3 = (k_2, s_1)$	t_{e_9}	$47ms$	3
10	$q_4 = (k_2, s_2)$	$t_{e_{10}}$	$25ms$	3
11	$q_4 = (k_2, s_2)$	$t_{e_{11}}$	$21ms$	2
12	$q_4 = (k_2, s_2)$	$t_{e_{12}}$	$12ms$	1

$$t_{e_1} < t_{e_2} \quad t_{e_2} < t_{e_3} \quad t_{e_4} < t_{e_6} \quad t_{e_6} < t_{e_5}$$
$$t_{e_7} < t_{e_8} \quad t_{e_8} < t_{e_9} \quad t_{e_{12}} < t_{e_{11}} \quad t_{e_{11}} < t_{e_{10}}$$

Ordinal Regression with Structural SVMs

Technische Universität Berlin

- Let us represent with r^* the real ranking in the training set
  - Given τ, which measures the error between the assigned ranking and the real ranking, we can formalize our problem as

$$\min \sum_{q_i \in Q} \tau_{q_i}(r, r^*)$$

- The learning problem can be solved using structural Support Vector Machines
  - Little training time [T. Joachim, KDD 2002]

trade-off between margin size and training error

weight vector

slack variable

$$\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{m}\sum_{(i,j)\in P} \xi_{i,j}$$

P set of all pairs (i,j) for which the instance i has higher rank than j

m=|P|

$$\text{subject to} \forall (i,j) \in P : (\mathbf{w}^T q_i, \mathbf{t_i}) \geq (\mathbf{w}^T q_j, \mathbf{t_j}) + 1 - \xi_{i,j}$$

feature vectors
stencil instance + tuning setting

AES Embedded Systems Architecture

Ordinal Regression with Partial Ranking

- Problem: we have only partial ranking
  - we cannot compare stencil executions with different stencil kernel k or size s
  - we have full ranking only for subsets of P
- Solution
  - Let us assume that $P_i \subset P$ as the set of inequalities generated by the instance qi
  - Assuming that $P_1, P_2 \cdots P_n$ are all partial rankings available in the training set , we modify the previous equation:

$$\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{m'}\sum_i \sum_{(j,k)\in P_i} \xi_{j,k}$$

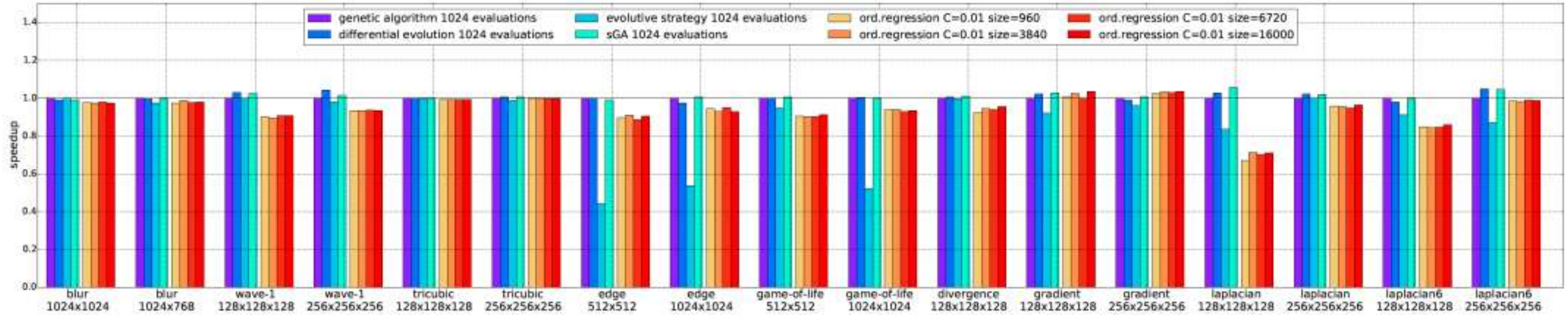
subject to:

$$\forall(j,k)\in P_1 : (\mathbf{w}^T q_1, \mathbf{t_j}) \geq (\mathbf{w}^T q_1, \mathbf{t_k}) + 1 - \xi_{j,k}$$
$$\forall(j,k)\in P_2 : (\mathbf{w}^T q_2, \mathbf{t_j}) \geq (\mathbf{w}^T q_2, \mathbf{t_k}) + 1 - \xi_{j,k}$$
$$\cdots$$
$$\forall(j,k)\in P_n : (\mathbf{w}^T q_n, \mathbf{t_j}) \geq (\mathbf{w}^T q_n, \mathbf{t_k}) + 1 - \xi_{j,k}$$

$$\text{where } m' = |\bigcup_i P_i| \text{ and } n = |Q|$$

Autotuning with Ordinal Regression
Quality of the Top-ranked version

- Good results also with small training dataset
  - with 8K-point dataset, 15 of 17 benchmarks performs >90% than an iterative-search approach



- Larger training dataset increase the ability of the model to correctly rank code versions
  - Kendall's Tau distribution
- Promising approach: can be applied to other tuning problems
  - as soon as code variants can be organized as partial rankings

Ranking Evaluation: Kendall's τ

- How to evaluate whether a ranking is good?
- Kendall's τ coefficient
  - Given two finite orderings $r_a, r_b \subset Q \times Q$, where Q is the set of all stencil instances

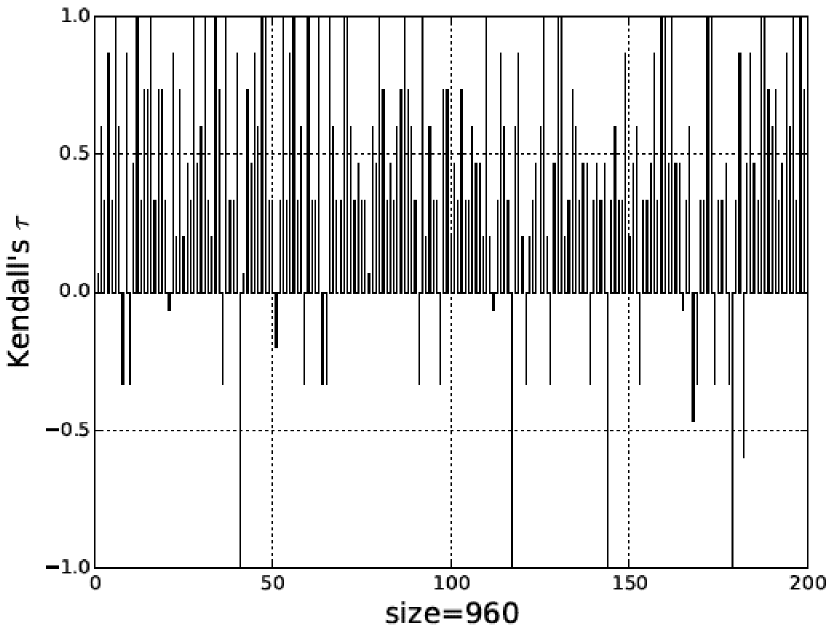
$$\tau(r_a, r_b) = \frac{Con - Dis}{Con + Dis} = 1 - \frac{2Dis}{\binom{m}{2}}$$

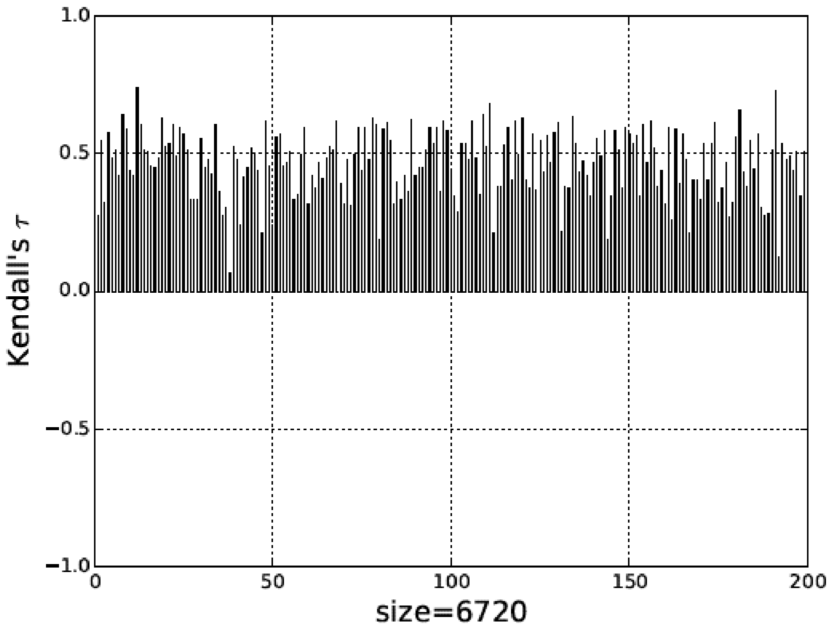
Con concordant pairs
Dis discordant pairs (inversions)

- τ value
  - 1, perfect agreement between r_a and r_b
  - 0, whether r_a and r_b are independent
  - -1, perfect disagreement between r_a and r_b (ranking in perfect reverse order)

Results
Ranking Evaluation with Kendall's τ Values

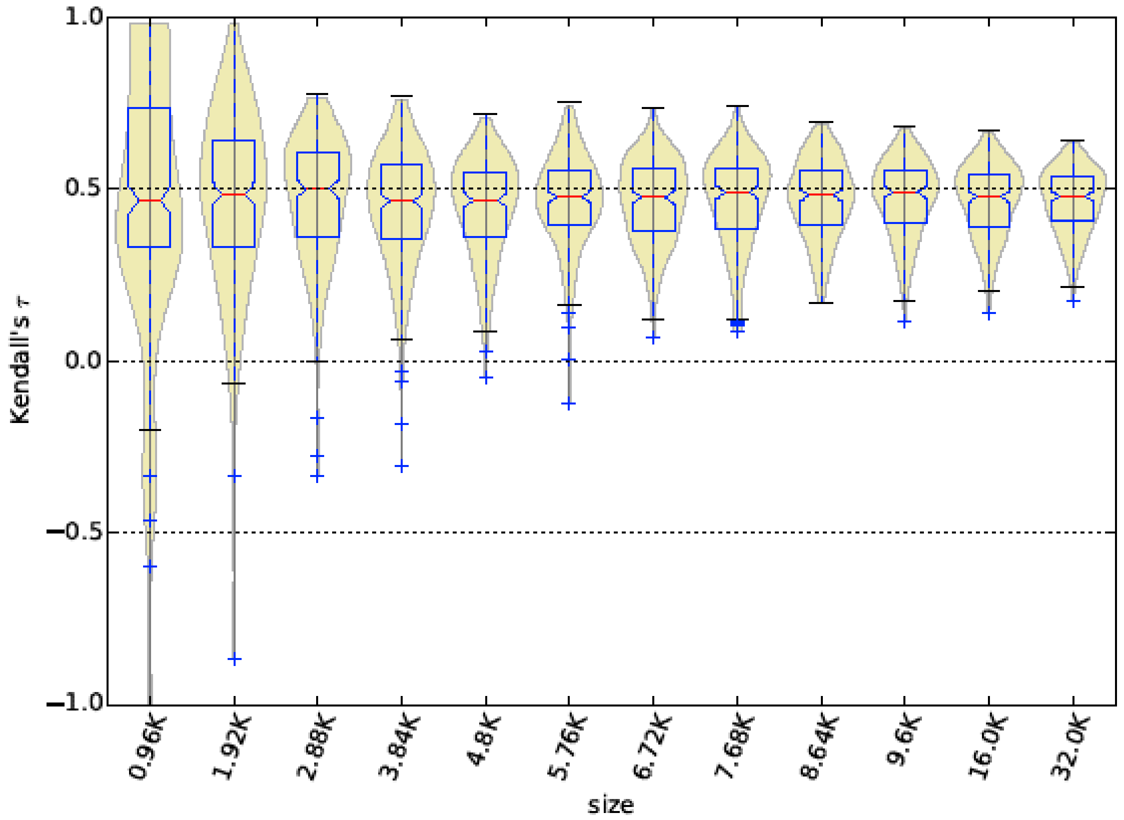
- The Kendall τ values on the training dataset, for two different training set sizes





Results
Ranking Evaluation with Kendall's τ Values

- The Kendall τ values with different training set sizes



Autotuning Stencil Computations with Structural Ordinal Regression Learning. Cosenza, Durillo, Ermon, Juurlink. IEEE IPDPS 2017

Machine learning with Auto-tuning

- Interesting for portability and integration into compilers
- Challenges
  - Applications
  - Training data availability
  - Input encoding
  - Modeling
- Some interesting solutions
  - Automatic synthetic training data generation
    - Domain-specific code generation [Cosenza et at., IPDPS 2017]
    - Generic synthesis approaches [Cummins et al., PACT 2017; CGO 2017]
  - Kernel functions
  - Structural learning

Programming Models & Tuning

The (parallel) programming model matter

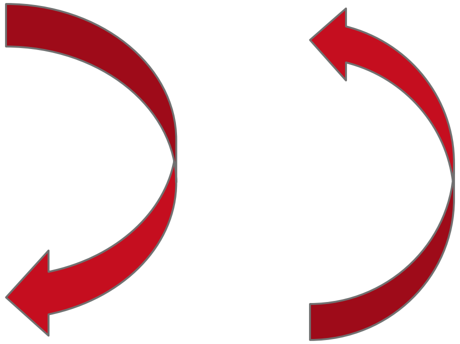
- Library
  - GROMACS parallelization
  - SCALAPACK
- Domain-specific Languages
  - Patus stencil compiler
  - OpenGL Shading Language
- Annotated C program
  - OpenMP
  - OpenACC
- C program
  - Automatic vectorization

High level

Low level

Programming Models & Tuning

- Interaction between high-level and low-level tuning
  - High-level tuning
    - Algorithm choices
    - Mapping, scheduling, parallelism granularity
    - Spatial data structures
  - Low-level tuning
    - Tiling, unrolling, vectorization



- Ongoing research
  - OpenABL: a domain-specific language for agent-based simulation
    - Target: multi-core CPU, GPU, cluster
  - CELERITY: extension of SYCL with compiler, runtime system and modeling
    - Target: High Performance Computing
    - Funded by DFG

Thanks for your attention

Auto-tuning Compiler Transformations with Machine Learning

Biagio Cosenza | LLVM Berlin Meetup | Mozilla Berlin Community Space | November 30, 2017

Acknowledgments: Ben Juurlink, Angela Pohl, Daniel Maier, Nikita Popov (TU Berlin), Stefano Ermon (Stanford University), Thomas Fahringer, Klaus Kofler, Ivan Grasso (University of Innsbruck), Juan Durillo (Leibniz Rechenzentrum)