Influences of MLDL on compiler technologies:

COMPILER OPTIMIZATIONS FOR MLDL APPLICATIONS,

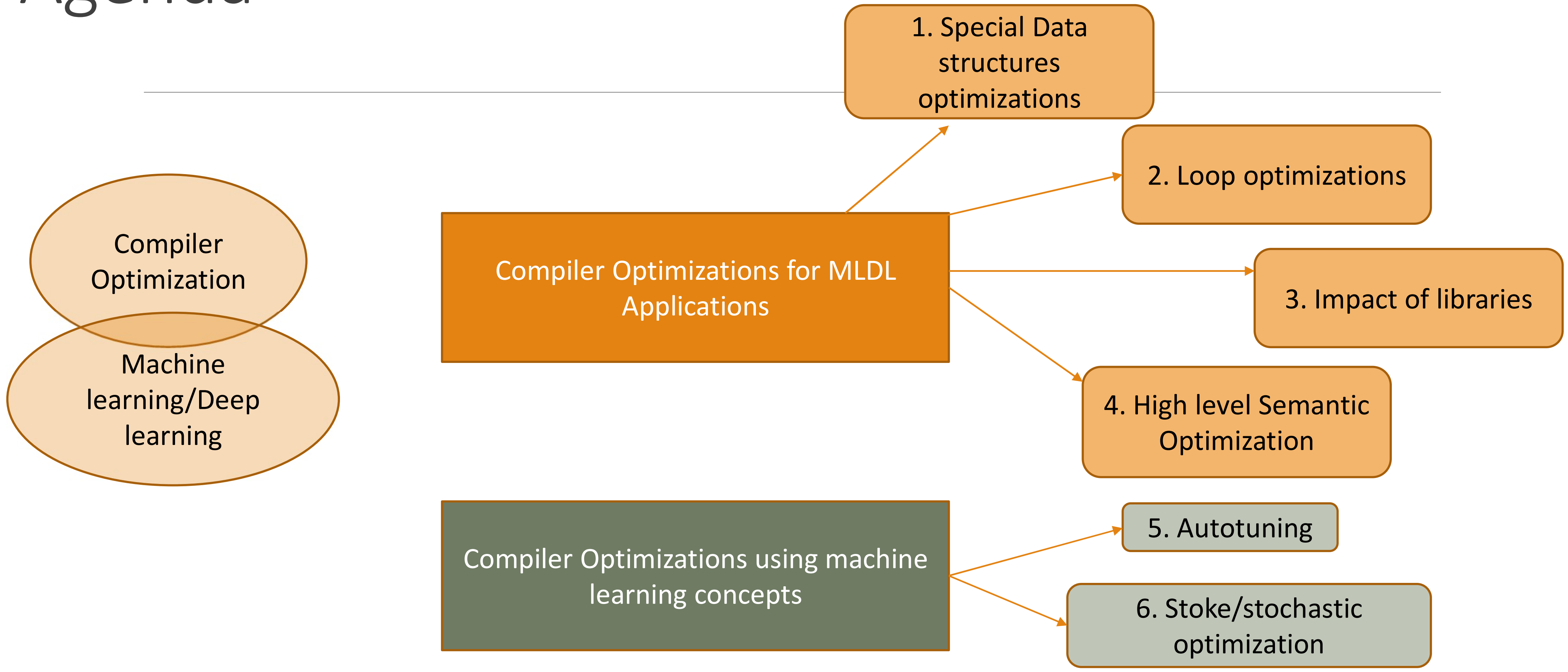
COMPILER OPTIMIZATION USING MACHINE LEARNING

A Ravindar, S Vinod and S Sadasivam

Power Systems Performance,

IBM Systems

Agenda


Compiler Optimization
Machine learning/Deep learning
Compiler Optimizations for MLDL Applications
1. Special Data structures optimizations
2. Loop optimizations
3. Impact of libraries
4. High level Semantic Optimization
Compiler Optimizations using machine learning concepts
5. Autotuning
6. Stoke/stochastic optimization

1. Focus on Special Data structures and operations

Predominantly MLDL applications work on very specific data structures such as matrices, tensors, sparse matrices

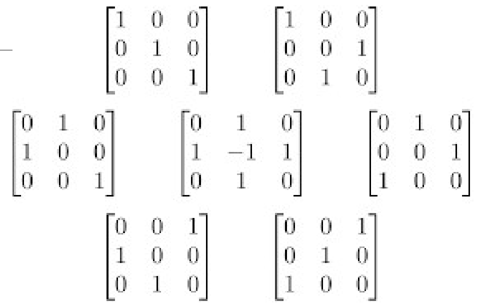
Operations on these type of data structures also tend to be very context specific

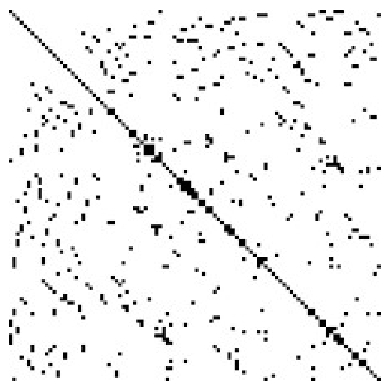
Specific compiler optimizations pertain to such structures and operations

Optimizations that work at a higher level and change layout of these data structures to optimize access time and reuse – could involve transforming a given data structure A to an optimized data structure B that could be more efficient to work with and thereby transforming all operations on A to operations on B.

For example sparse matrices are transformed into an equivalent data structures likewise operations also get transformed.

Lower level optimizations that improve the code already generated for the access of these data structures either in terms of scheduling or generating specialized ISA





https://fr.m.wikipedia.org/wiki/Fichier:Finite_element_sparse_matrix.png
https://commons.wikimedia.org/wiki/File:Alternating_Sign_Matrices_of_Size_3.svg

2. Focus on loop optimizations

Several of these operations on special data structures involve loops

Popular optimizations that typically work on the loops accessing the special structures have been- Loop reordering, Tiling, Fusion, splitting, blocking

Depending on the compiler, some of these are automatically enabled under certain optimization levels while some require explicit user pragmas in source code

Loop vectorization – Depending on the ISA support of the underlying hardware architecture, loops tend to be vectorized to give greater throughput

Other Specialized ISA optimizations such as FMA can also contribute in performance improvement

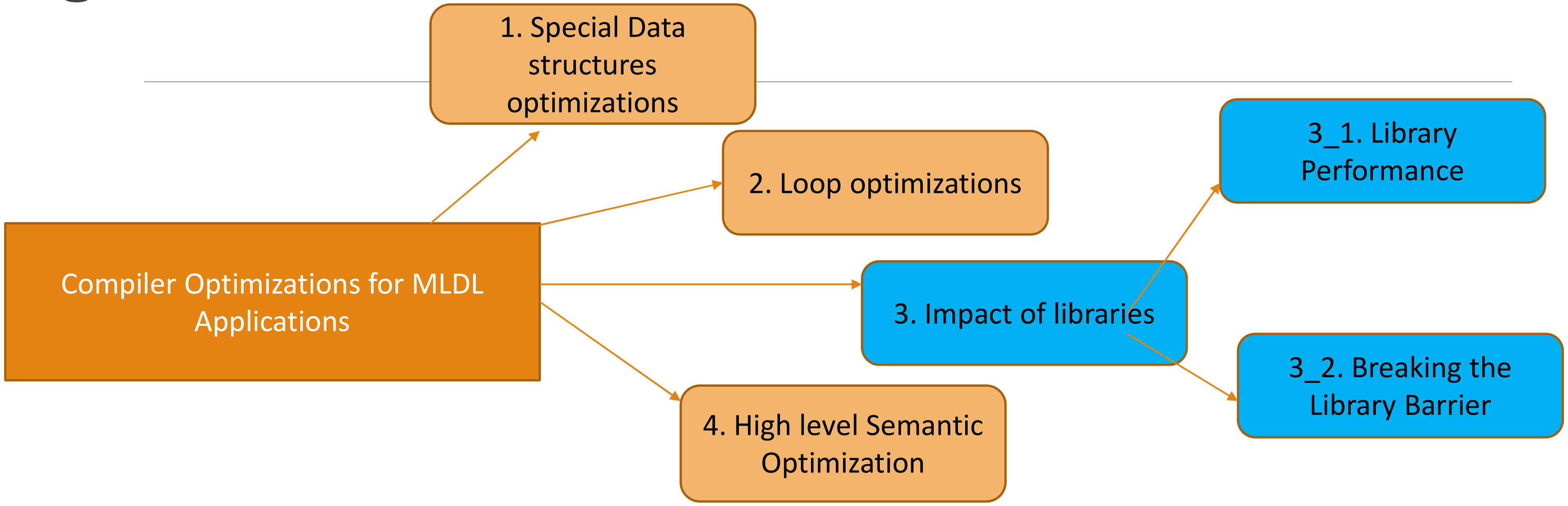
```
int a[5000], b[5000], c[5000];

#pragma unrollandfuse(2)
for(i=1;i<5000;i++) {
  for(j=1;j <5000; j++)
     a[j][i]=b[i][j] * c[j][i];
}

Becomes
for(i=1;i<5000;i+=2) {
  for(j=1;j <5000; j++)  {
     a[j][i]=b[i][j] * c[j][i];
     a[j][i+1]=b[i][j+1] * c[j][i+1];
  }
}

A similar benefit can be got by loop
reordering and unrolling
```

Agenda


Compiler Optimizations for MLDL Applications
1. Special Data structures optimizations
2. Loop optimizations
3. Impact of libraries
3_1. Library Performance
3_2. Breaking the Library Barrier
4. High level Semantic Optimization

3_1: Impact of Libraries: Library performance

Most MLDL / AI applications use libraries

Most operations occur at the granularity of matrices or tensors or other such special data structures

Most operations are provided in terms of specialized library API inorder to absolve the user from having to code these on their own

Common examples include Math libraries such as OpenBLAS, Intel MKL, Nvidia that supports different libraries across the stack such as cublas , cudnn, cuSPARSE

This also means that the end application performance depends on underlying library performance

3_2: Impact of Libraries: Breaking the Library barrier

Libraries at the same time also seem to have a downside

Most MLDL/AI application frameworks are constituted of operations on various special data structures thereby consisting of library calls and interactions among the calls

Though a library call in itself can be optimized to be really fast, the maximum performance it can achieve is speeding up the individual *call* and not the overall application

There have been experiments and studies conducted on optimizing the overall application by lowering the application and library calls into a common IR and then optimizing the overall application across the calls and have shown to yield speedups of upto 25x in certain cases

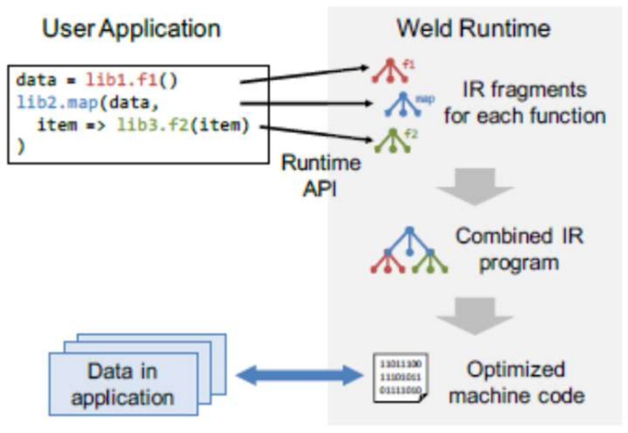


Figure 1: As applications call Weld-enabled libraries, the runtime collects fragments of IR code, which it then combines into a single Weld program. This complete program is optimized jointly and then compiled to parallel machine code, which executes against the application's in-memory data.

S. Palkar et al, WELD: A Common Runtime for High Performance Data Analytics

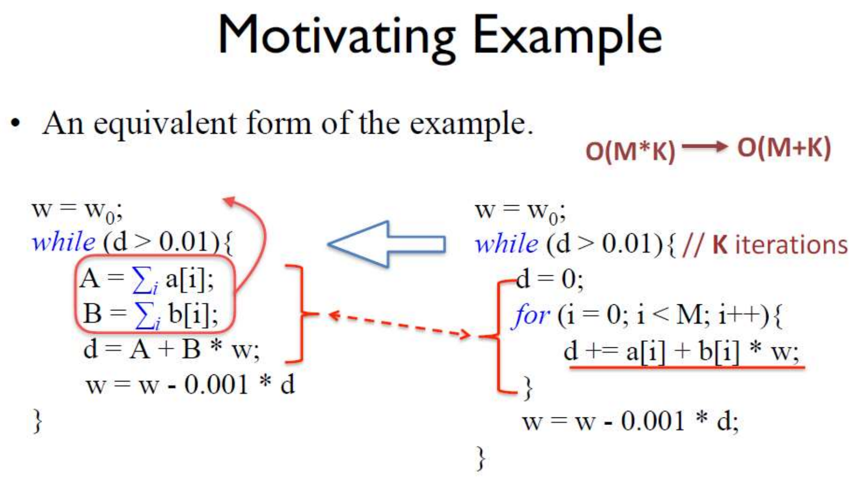
4. Taking a step up – Semantic level optimization

Many a times, if we take a step up the ladder of abstraction we find newer optimizations

Generalized Loop Redundancy Removal (GLORE)- Y.Ding et al, OOPSLA 2017

<u>If analysis scope becomes larger, our field of possible optimizations tend to widen</u>

Use the right abstractions coupled with algorithmic optimizations we can leverage higher level semantics thereby advancing the underlying compiler technology.



Motivating Example

- An equivalent form of the example.

O(M*K) ⟶ O(M+K)

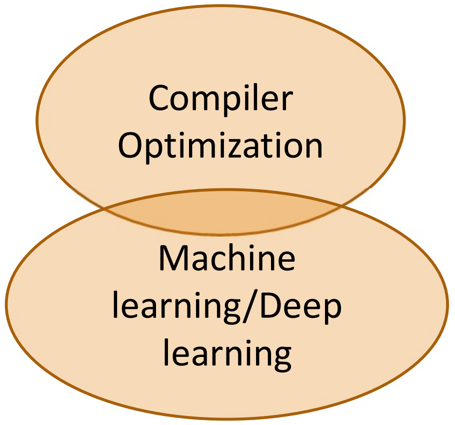
```
w = w0;
while (d > 0.01){
    A = ∑i a[i];
    B = ∑i b[i];
    d = A + B * w;
    w = w - 0.001 * d
}
```

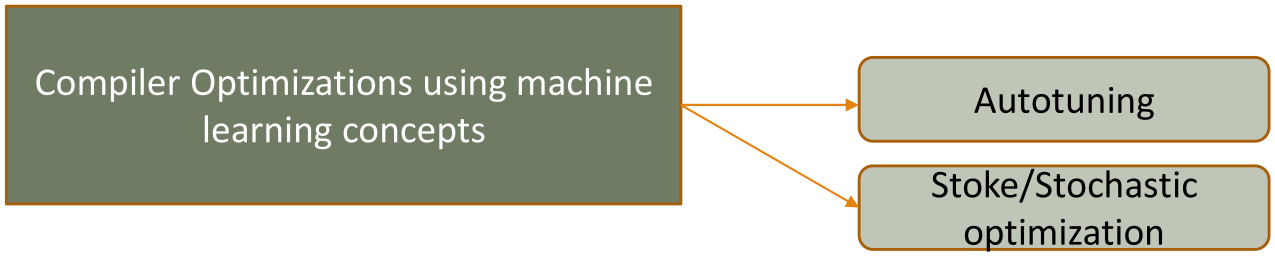
```
w = w0;
while (d > 0.01){ // K iterations
    d = 0;
    for (i = 0; i < M; i++){
        d += a[i] + b[i] * w;
    }
    w = w - 0.001 * d;
}
```

i stays the same across while loop but varies across for loop
w stays the same across for loop but varies across while loop
Traditional compiler would not optimize this loop

Image courtesy: Prof. Xipeng Shen, Rethinking Compilers in the Rise of Machine Learning and AI, CC18 Keynote

Agenda


Compiler
Optimization
Machine
learning/Deep
learning


Compiler Optimizations using machine learning concepts
Autotuning
Stoke/Stochastic optimization

Autotuning frameworks

Autotuning refers to be able to optimize a program without much of a user intervention and usually involves machine learning internally

A most recent survey of techniques are presented in Ashouri et al, A Survey on Compiler Auto-tuning using Machine learning https://arxiv.org/pdf/1801.04405.pdf that describes several efforts in this field including ones by Ashouri

We pick CK one of the frameworks for discussion; CK can be used for collaborative tuning across multiple organizations. In itself, it supports optimization across compilers, libraries, run-time systems and platforms

It works on databases built with collecting measurements of performance of applications with combinations of various compilers, libraries, operating systems, hardware platforms

Uses mature machine learning algorithms underneath to come up with ideal configuration, environment, flags to run the application with

Examples are CK which works with GCC, LLVM compilers and is currently being used by ARM

Is open source https://github.com/ctuning/ck-autotuning

G. Fursin et al, A Collective knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques, https://arxiv.org/pdf/1801.08024.pdf

Stoke/Stochastic Optimization

Targeted for short sequences of loop free fixed point assembly code sequences in intel x86_64 architecture

Correctness conditions and performance improvement modelling is encoded as cost functions

MCMC (markov chain monte carlo) sampler is used to rapidly explore space of all possible sequences and based on the optimization of the cost function the optimal sequence is generated.

eq(.) returns zero only if the original code T and rewritten code R are equivalent semantically

$$\mathrm{cost}(\mathcal{R}; \mathcal{T}) = w_e \cdot \mathrm{eq}(\mathcal{R}; \mathcal{T}) + w_p \cdot \mathrm{perf}(\mathcal{R}) \qquad \{\, r \mid \mathrm{perf}(r) \leq \mathrm{perf}(\mathcal{T}) \wedge \mathrm{eq}(r; \mathcal{T}) = 0 \,\}$$

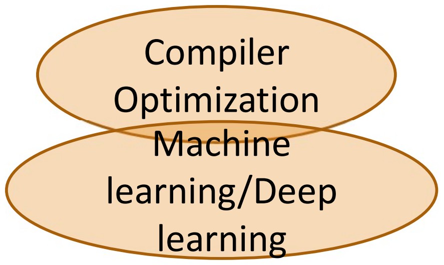
Lower perf(.) value; greater the benefit; an optimization is when performance of rewrite is greater than the target;

Search space taken up by Stoke is radically different from that of conventional optimization and builds on incremental synthesis.

Prototype code for intel is available at https://github.com/StanfordPL/stoke

More information on this technology can be gleaned from http://stoke.stanford.edu/

Summary



The two fields- compiler optimization and machine learning have increasingly begun to intersect in recent times.

We discussed ways in which compilers are being used to tune MLDL/AI applications

Likewise, we saw how the revolution in MLDL/AI are impacting the way in which compiler optimizations have been carried out and are being instrumental in adding new advances to compiler technology itself

We are in an exciting period as newer innovations in both fields are feeding off each other

Compiler Optimizations for MLDL Applications

Data structures, Loops, Libraries, High level Semantic optimizations

Compiler Optimizations using machine learning concepts

Autotuning, stochastic optimization

Backup

Flags for SIMD, FMA, unroll

Unrolling

- GCC, clang: -funroll-loops
- XL: -qunroll

SIMD

- GCC: -ftree-vectorize, clang: vectorization enabled by default, control width of vectorization by -mllvm –force-vector-width=n, -fslp-vectorize
- XL: -qsimd=auto

FMA

- GCC, clang: –ffp-contract=fast that is usually enabled by -Ofast

Specific Optimizations in Inferencing and Training Passes

Training phase is highly compute intensive and also parallelizable

Ideal to run in accelerators

Inferencing also shares characteristics with Training

Access latency of constituent elements / data transfer latencies assume equal importance as compute

Data layout, access sequence optimization using flags and specific optimization flags assume importance

Since sequences of code used in inferencing and training are used repeatedly they qualify as hot sequences and additional compiler optimization can kick in to optimize these

Other Key References for High level Semantic level Optimizations

Other studies by the same group that have results around applications of Higher level semantic optimizations –

Generalizations of the theory and deployment of triangular inequality for compiler-based strength reduction, Y. Ding et al, PLDI 2017

TOP: a framework for enabling algorithmic optimizations for distance-related problems, Y. Ding et al, VLDB 2015

Yinyang K-Means: A Drop-In Replacement of the Classic K-Means with Consistent Speedup, Y. Ding et al, ICML, 2015

LCD: A Fast Contrastive Divergence Based Algorithm for Restricted Boltzmann Machine, ICDM 2017, L. Ning et al

Sweet KNN: An Efficient KNN on GPU through Reconciliation between Redundancy Removal and Regularity, G. Chen et al.

How ML can be used to auto-tune applications

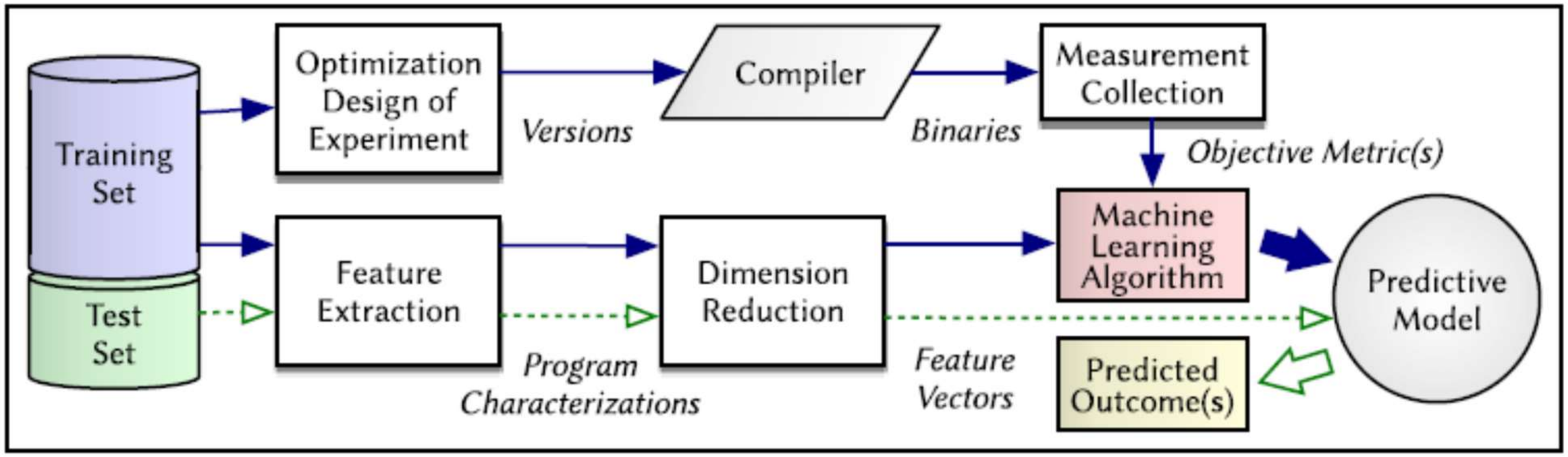


Fig. 2. A Sample Autotuning Framework Using Machine Learning (1) Top: The data flow through the various components of the training phase, where a model is induced based on the characteristics of applications under training set, and, (2) Bottom: The data flow through the various components of the test phase, where the already trained model is used to predict an outcome for a given test application with applications under the test set.

CK: Modelling Behavior of a program b in terms of c,f,s

b : behaviour vector (exec time, codesize, param of optimization)

c: choices (design knobs such as algorithm parameters, model topology, flags, hardware characteristics, source to source transformations)

f: feature vector (program and data set features, hardware counters, platform properties)

s: state vector (run time system state, hardware frequencies, network status, cache state etc)

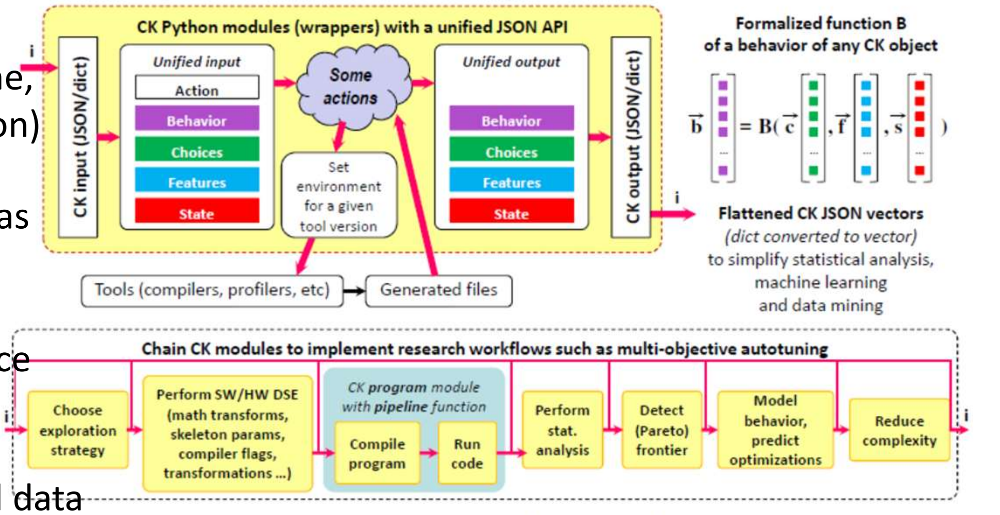

CK Python modules (wrappers) with a unified JSON API
i
CK input (JSON/dict)
Unified input
Action
Behavior
Choices
Features
State
Some actions
Set environment for a given tool version
Unified output
Behavior
Choices
Features
State
CK output (JSON/dict)
i
Tools (compilers, profilers, etc)
Generated files
Formalized function B of a behavior of any CK object
$\vec{b} = B(\vec{c}, \vec{f}, \vec{s})$
Flattened CK JSON vectors
(dict converted to vector)
to simplify statistical analysis, machine learning and data mining
Chain CK modules to implement research workflows such as multi-objective autotuning
i
Choose exploration strategy
Perform SW/HW DSE (math transforms, skeleton params, compiler flags, transformations ...)
CK program module with pipeline function
Compile program
Run code
Perform stat. analysis
Detect (Pareto) frontier
Model behavior, predict optimizations
Reduce complexity
i

Figure 7: Chaining together various CK modules with JSON API and JSON meta information to implement universal, portable, customizable, multi-dimensional and multi-objective autotuner gradually extended by the community.

G. Fursin et al. A Collective knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques

References for Stoke

Stochastic Superoptimization – ASPLOS 2013

Data-Driven Equivalence Checking – OOPSLA 2013

Stochastic Optimization of Floating-Point Programs with Tunable Precision – PLDI 2014

Conditionally Correct Superoptimization – OOPSLA 2015

Stochastic Program Optimization – CACM 2016

Stratified Synthesis: Automatically Learning the x86-64 Instruction Set – PLDI 2016

Sound Loop Superoptimization for Google Native Client – ASPLOS 2017

Download

Can be downloaded from github.com/StanfordPL/stoke.

Uses intel PIN tool

Tools augmenting compiler analysis pertaining to MLDL development and tuning

There has been substantial efforts in developing supporting tools

Profiling tools for GPU- NVIDIA nvprof, CPU- perf

Tools for Ease of development of AI applications such as building the model easily

POWERAI vision- helps build training models, label, train and deploy these models and can be used by people who have minimal deep learning expertise

Watson machine learning(WML) – CE (community edition) supported on POWER and Intel platforms with NVIDIA GPUs , WML-A (accelerator) for enterprise AI

IBM has a measurement, evaluation and tuning framework to evaluate multiple ML applications and their parameters- MLModelScope

Modelscope has the following features- builtin datasets, integration of various frameworks, system agnostic, workflow analysis, evaluation of data set accuracy, publishing results. Modelscope supports x86 and POWER. https://mlmodelscope.org/