Daisytuner

A Self-learning Compiler for Tensor Processing on the Edge

Nora Hagmeyer, Ramon Wirsch, Adrian Schmitz, Lukas Trümper

hessian.AI

esa

BUSINESS INCUBATION CENTRE

push! 2024

Ein Förderprogramm des Hessischen Ministeriums für Wirtschaft, Energie, Verkehr, Wohnen und ländlichen Raum

Table of Contents

Motivation

Moore's Law:

"The number of transistors doubles every two years"

1970-2000:
2x Speedup every two years on **the same code**

2000+:
What's happening now?

CPU	Year	Transistors	Clock	Structure	Caches
4004	1971	2,300	740 kHZ	20 Micro	
8008	1972	3,500	500 kHz	10 Micro	
8086	1978	29,000	10 MHz	3 Micro	
80286	1982	134,000	25 MHz	1.5 Micro	
80386	1985	275,000	33 MHz	1 Micro	
80486	1989	1,2M	50 MHz	0.8 Micro	8K
Pentium I	1994	3.1M	66 MHz	0.8 Micro	8K
Pentium II	1997	7.5M	300 MHz	0.35 Micro	16K/512K
Pentium III	1999	9.5M	600 MHz	0.25 Micro	16K/512K
Pentium IV	2000	42M	1.5 GHz	0.18 Micro	8K/256K
P IV F	2005	**~160M**	**2.8 GHz**	90 nm	16K/2MB
Core I7	2008	**781M**	**3.2 GHz**	45 nm	32/256/8
Ivy Bridge	2013	**2,890M**	**3.3 GHz**	22 nm	32/256/30

Motivation

Problem:

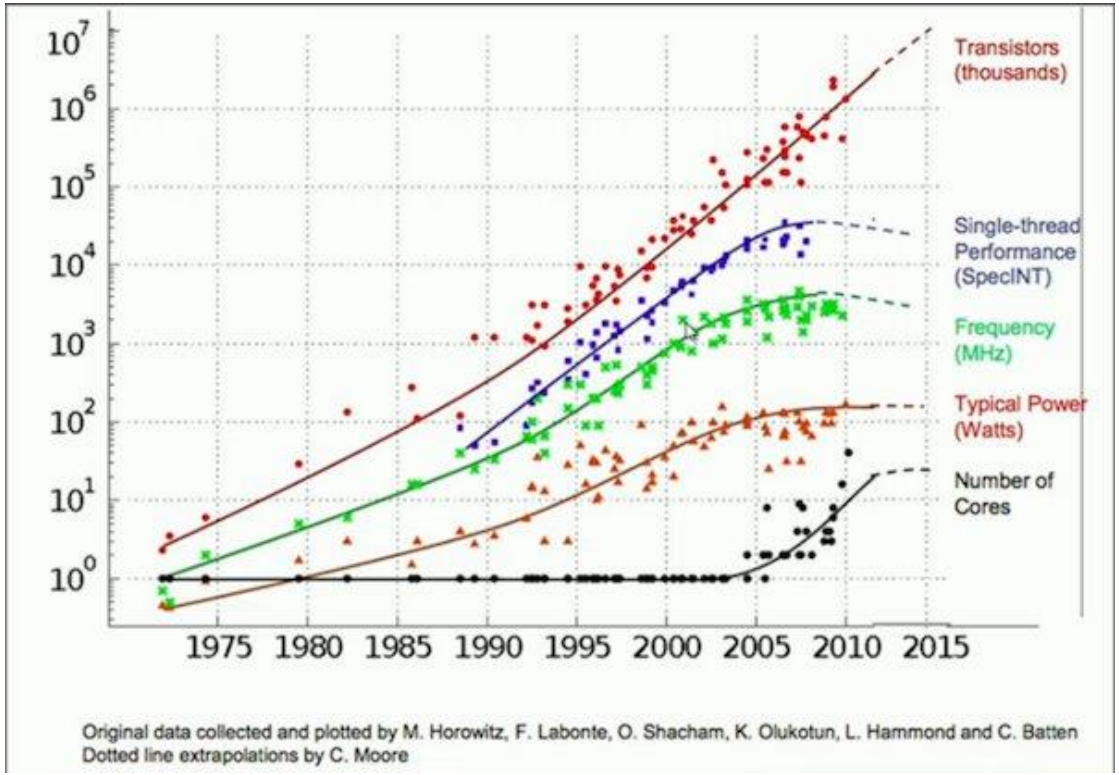
Power = clock³

-> Top 500 consume 3 Billion kWh per year
-> Use of AI & digital twins leads to increasing resource needs

Motivation

Solution:

Const: frequency
Grow: cores (and caches)



Adapt your code to the hardware!

Motivation

Solution:

Const: frequency
Grow: cores (and caches)
Specialization: accelerated computing

Adapt your code to the hardware!

Motivation

Solution:

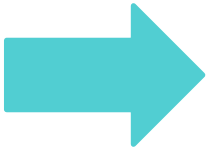
Const: frequency
Grow: cores (and caches)
Specialization: accelerated computing

Performance is a primary design goal. Rewriting code is expensive!

Motivation

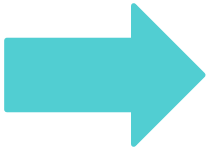
Motivation

HPC Application



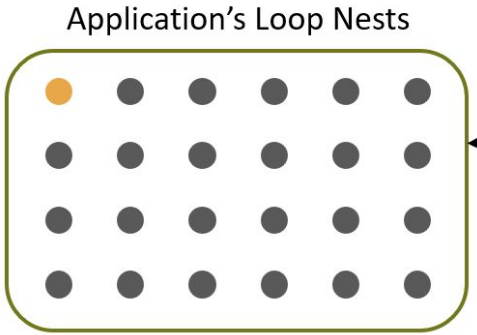
Loop Nests

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

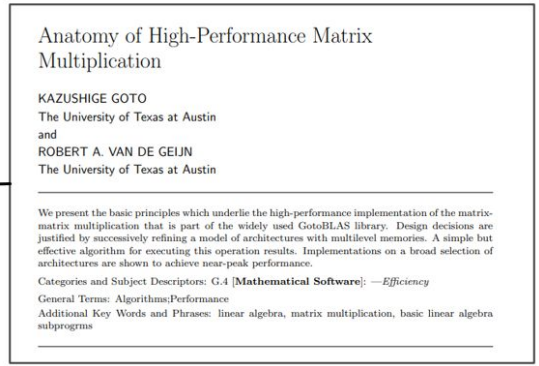


Optimized HPC Application

Motivation



-Omatmul



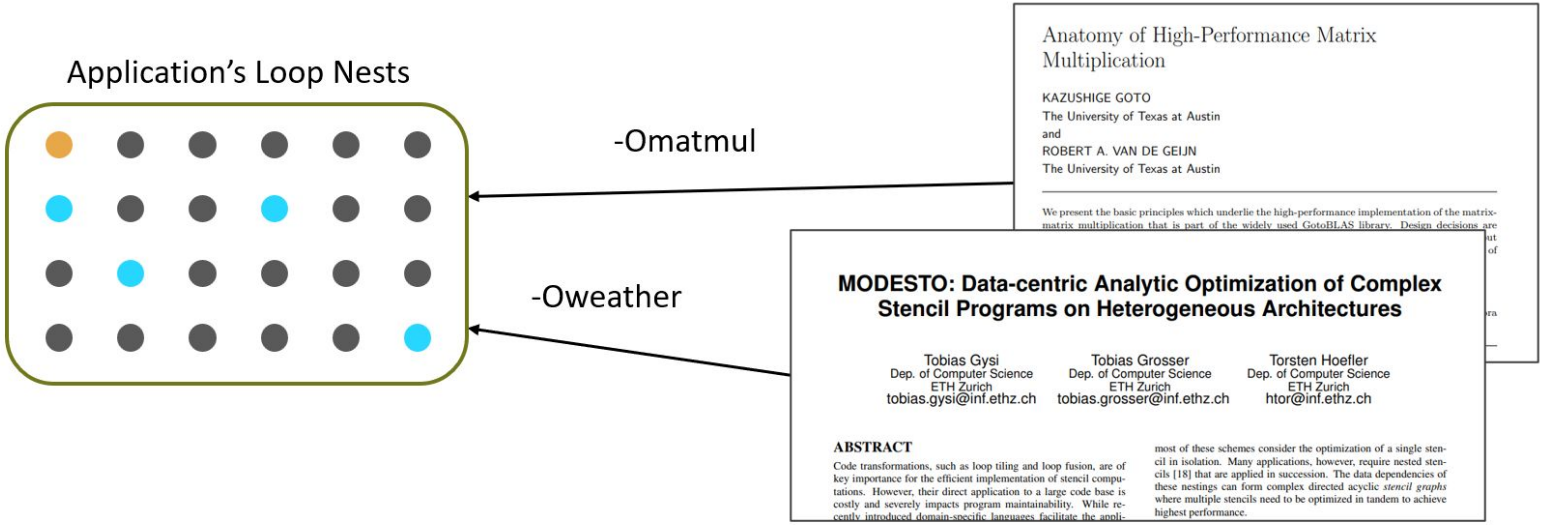
Anatomy of High-Performance Matrix Multiplication

KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

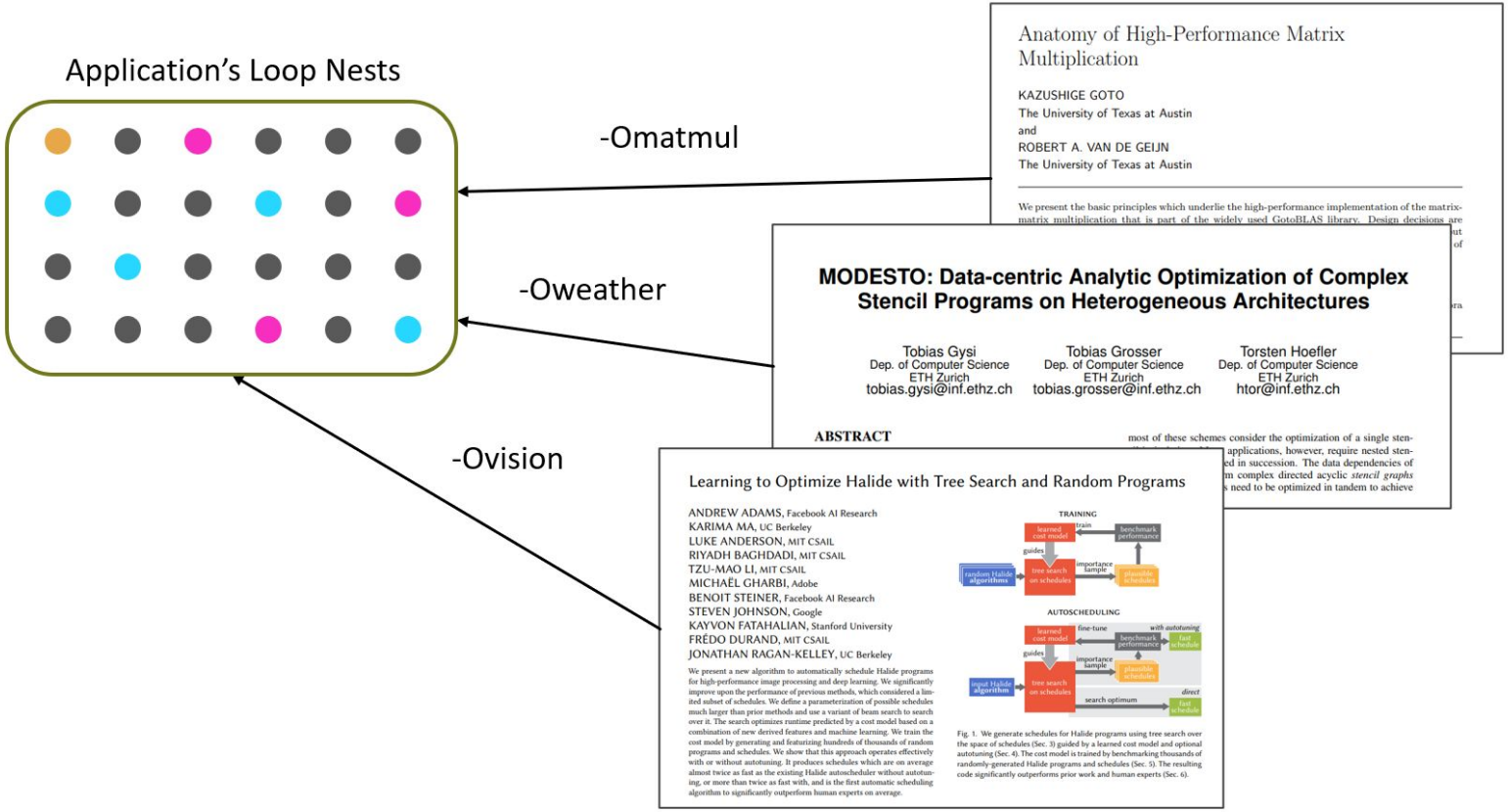
We present the basic principles which underlie the high-performance implementation of the matrix-matrix multiplication that is part of the widely used GotoBLAS library. Design decisions are justified by successively refining a model of architectures with multilevel memories. A simple but effective algorithm for executing this operation results. Implementations on a broad selection of architectures are shown to achieve near-peak performance.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: —*Efficiency*
General Terms: Algorithms;Performance
Additional Key Words and Phrases: linear algebra, matrix multiplication, basic linear algebra subprogrms

Motivation


Application's Loop Nests
-Omatmul
-Oweather
Anatomy of High-Performance Matrix Multiplication
KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin
We present the basic principles which underlie the high-performance implementation of the matrix-
MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures
Tobias Gysi
Dep. of Computer Science
ETH Zurich
tobias.gysi@inf.ethz.ch
Tobias Grosser
Dep. of Computer Science
ETH Zurich
tobias.grosser@inf.ethz.ch
Torsten Hoefler
Dep. of Computer Science
ETH Zurich
htor@inf.ethz.ch
ABSTRACT
Code transformations, such as loop tiling and loop fusion, are of key importance for the efficient implementation of stencil computations. However, their direct application to a large code base is costly and severely impacts program maintainability. While recently introduced domain-specific languages facilitate the appli-
most of these schemes consider the optimization of a single stencil in isolation. Many applications, however, require nested stencils [18] that are applied in succession. The data dependencies of these nestings can form complex directed acyclic stencil graphs where multiple stencils need to be optimized in tandem to achieve highest performance.

Motivation


Application's Loop Nests
-Omatmul
-Oweather
-Ovision
Anatomy of High-Performance Matrix Multiplication
KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin
We present the basic principles which underlie the high-performance implementation of the matrix-
MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures
Tobias Gysi
Dep. of Computer Science
ETH Zurich
tobias.gysi@inf.ethz.ch
Tobias Grosser
Dep. of Computer Science
ETH Zurich
tobias.grosser@inf.ethz.ch
Torsten Hoefler
Dep. of Computer Science
ETH Zurich
htor@inf.ethz.ch
ABSTRACT
most of these schemes consider the optimization of a single sten-
applications, however, require nested sten-
ed in succession. The data dependencies of
m complex directed acyclic stencil graphs
need to be optimized in tandem to achieve
Learning to Optimize Halide with Tree Search and Random Programs
ANDREW ADAMS, Facebook AI Research
KARIMA MA, UC Berkeley
LUKE ANDERSON, MIT CSAIL
RIYADH BAGHDADI, MIT CSAIL
TZU-MAO LI, MIT CSAIL
MICHAËL GHARBI, Adobe
BENOIT STEINER, Facebook AI Research
STEVEN JOHNSON, Google
KAYVON FATAHALIAN, Stanford University
FRÉDO DURAND, MIT CSAIL
JONATHAN RAGAN-KELLEY, UC Berkeley
We present a new algorithm to automatically schedule Halide programs for high-performance image processing and deep learning. We significantly improve upon the performance of previous methods, which considered a limited subset of schedules. We define a parameterization of possible schedules much larger than prior methods and use a variant of beam search to search over it. The search optimizes runtime predicted by a cost model based on a combination of new derived features and machine learning. We train the cost model by generating and featurizing hundreds of thousands of random programs and schedules. We show that this approach operates effectively with or without autotuning. It produces schedules which are on average almost twice as fast as the existing Halide autoscheduler without autotuning, or more than twice as fast with, and is the first automatic scheduling algorithm to significantly outperform human experts on average.
TRAINING
learned cost model
train
benchmark performance
guides
random Halide algorithms
tree search on schedules
importance sample
plausible schedules
AUTOSCHEDULING
learned cost model
fine-tune
with autotuning
benchmark performance
fast schedules
guides
importance sample
plausible schedules
input Halide algorithm
tree search on schedules
search optimum
direct
fast schedules
Fig. 1. We generate schedules for Halide programs using tree search over the space of schedules (Sec. 3) guided by a learned cost model and optional autotuning (Sec. 4). The cost model is trained by benchmarking thousands of randomly-generated Halide programs and schedules (Sec. 5). The resulting code significantly outperforms prior work and human experts (Sec. 6).

Motivation

A long term consideration would be to introduce a different memory layout, with better cache locality and clearer access patterns for the compiler to vectorize with more ease. Figure 6 depicts the current memory layout and a reordered

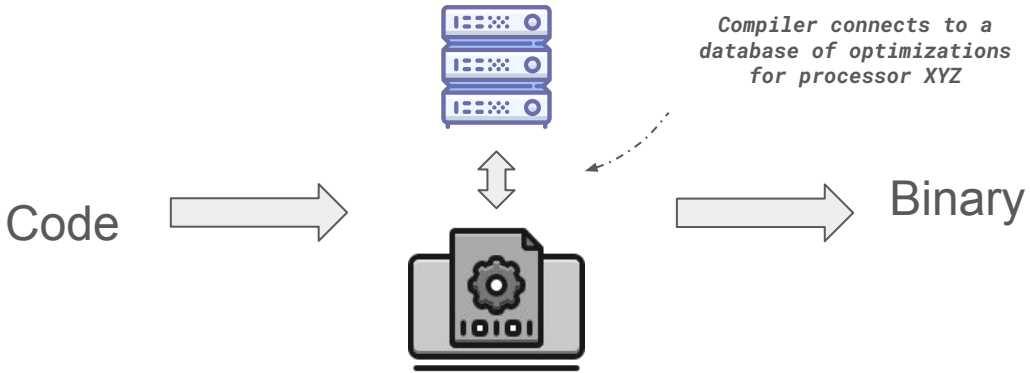
Motivation

A long term consideration would be to introduce a different memory layout, with better cache locality and clearer access patterns for the compiler to vectorize with more ease. Figure 6 depicts the current memory layout and a reordered

Can we talk to the compiler (to generate those insights automatically) ?

Motivation

DOCC: A compiler that learns with every program (based on LLVM plugins, cf. transfer tuning)



Daisytuner

Team

Lukas Trümper

Co-Founder
Former: RWTH Aachen, ETH Zurich
Computer Science

Adrian Schmitz

Co-Founder
RWTH Aachen
Computer Science

Nora Hagmeyer

Former: TU Munich, Bundeswehr
University Munich, INRIA
Computational Mechanics

Ramon Wirsch

Former: TU Darmstadt
Elektro- und Informationstechnik

Scientific Advisors

Torsten Hoefler

Professor @ ETH Zurich
ACM Fellow, IEEE Fellow

Tal Ben-Nun

Researcher @ Lawrence Livermore
Forbes Israel's 30u30

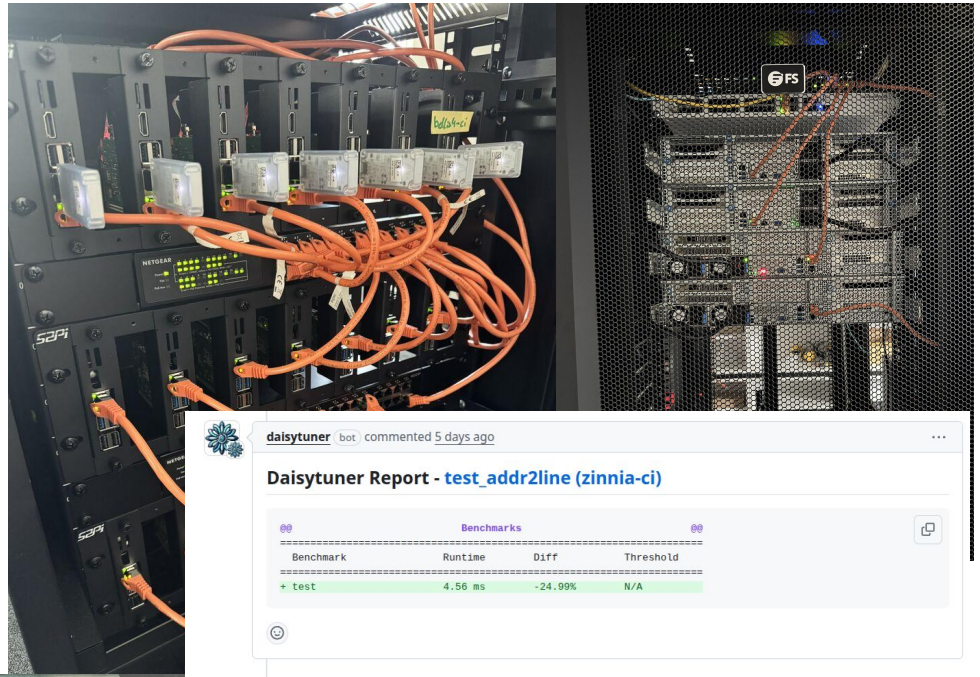
Christian Terboven

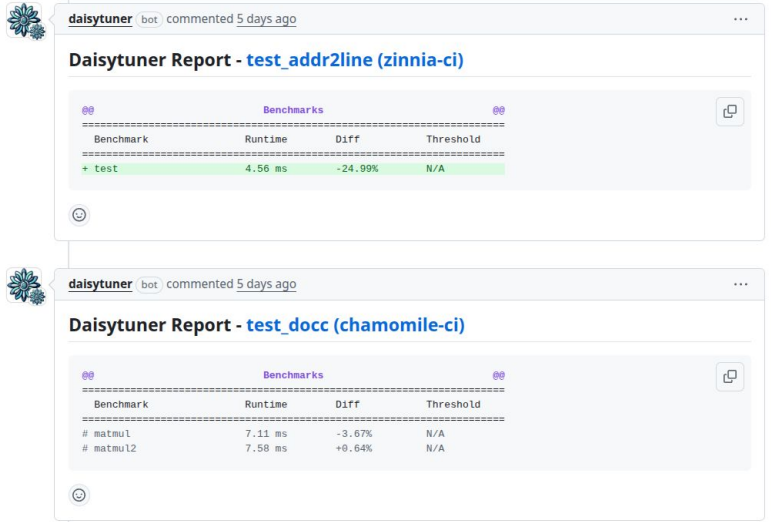
Researcher @ RWTH Aachen
OpenMP committee

Continuous Benchmarking

Partition	CPU	Accelerators	Description
`bellis4`	ARM Cortex A-72	Coral Edge TPU (USB)	Raspberry Pi 4
`bellis5`	ARM Cortex A-76	HAILO 8L	Raspberry Pi 5
`tansy`	ARM Cortex A-78	NVIDIA Ampere	NVIDIA Jetson Orin Nano Super
`zinnia`	AMD EPYC 9554	NVIDIA A16, Tenstorrent Wormhole N150, Axelera AI Metis	
`chamomile`	Intel Xeon Silver 4516Y+	NVIDIA A10, Tenstorrent Wormhole N150, Axelera AI Metis	
`aster` (soon)	Ampere Altra Max M128-30	NVIDIA L4, Tenstorrent Wormhole N150, Axelera AI Metis	

NOTE: We're currently working on bringing up the *Axelera AI Metis* and *HAILO 8L* processors.





daisytuner.com

DOCC

DOCC: A compiler that learns with every program (based on LLVM plugins, cf. transfer tuning)

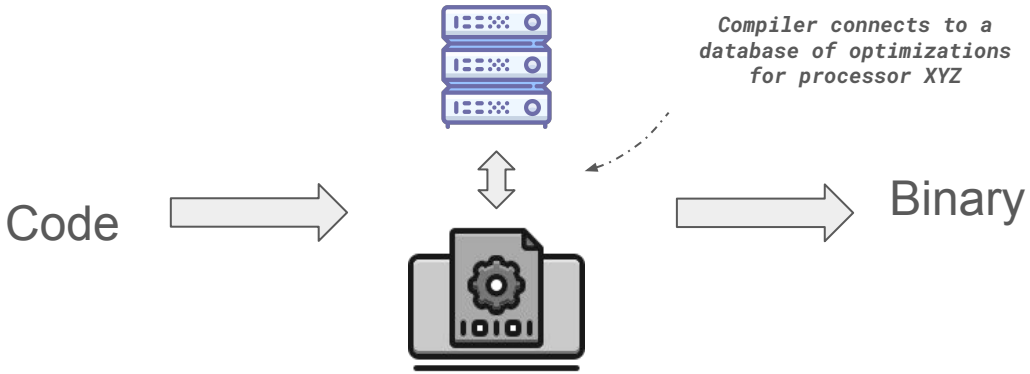
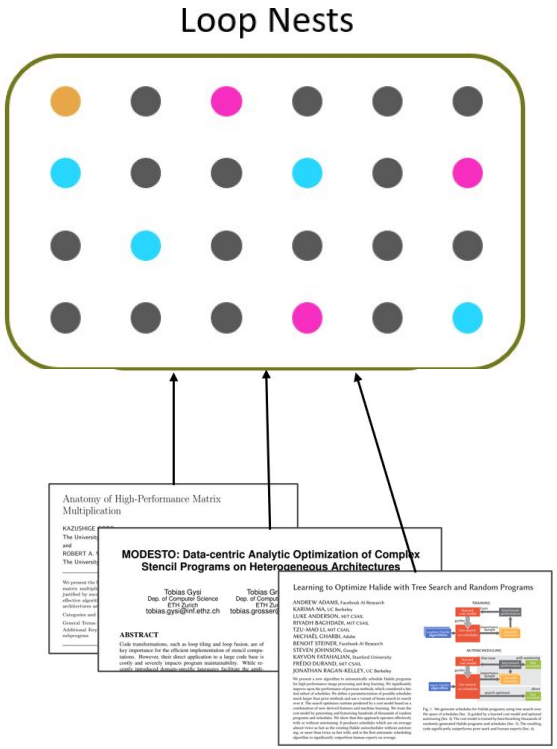


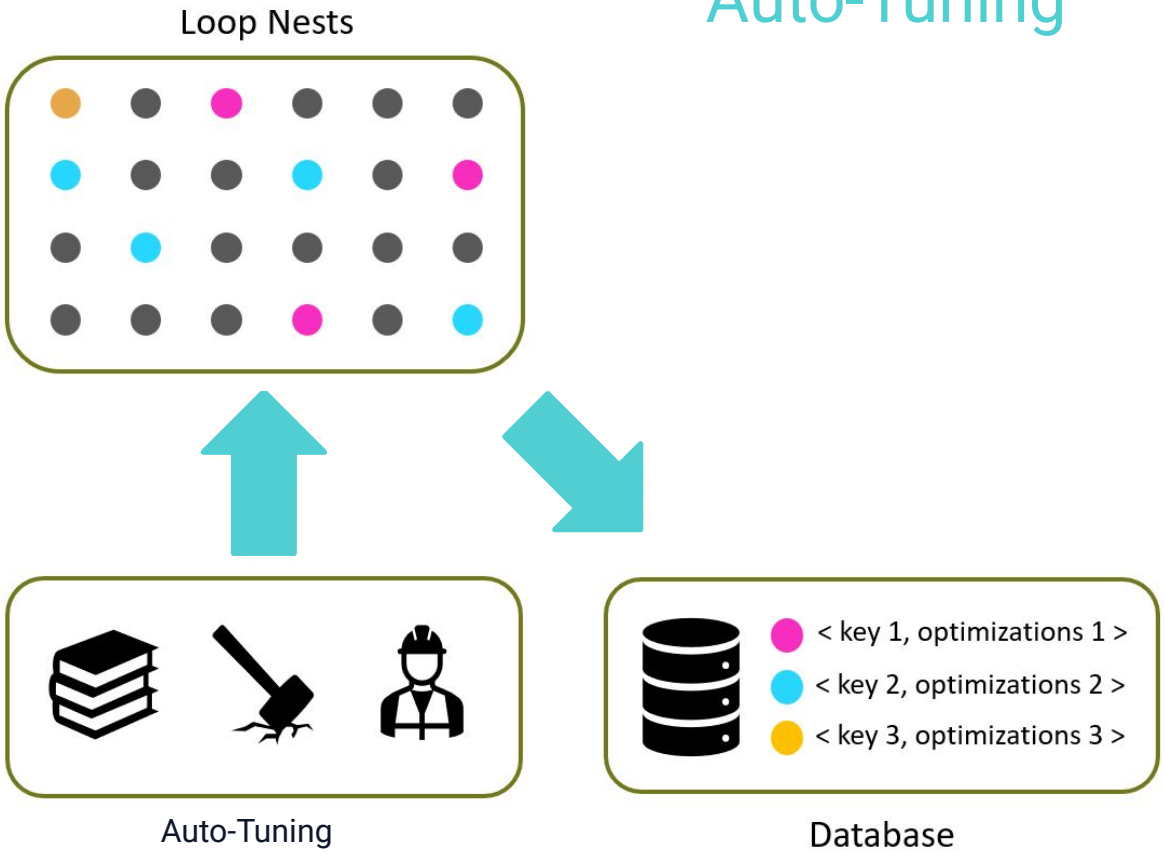
Table of Contents

Tuning

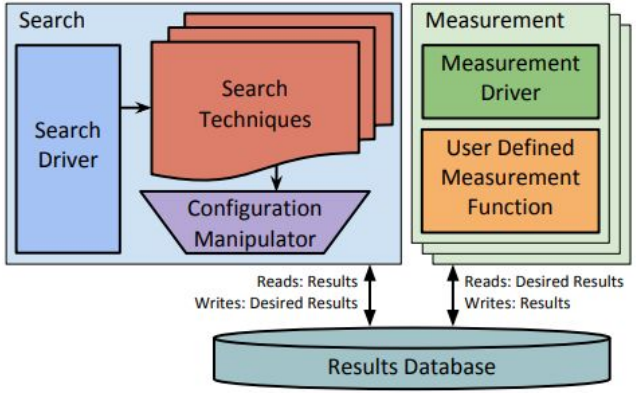


ICS '23: Proceedings of the 37th International Conference on Supercomputing • Pages 50 - 62
https://doi.org/10.1145/3577193.3593714

Auto-Tuning


Loop Nests
< key 1, optimizations 1 >
< key 2, optimizations 2 >
< key 3, optimizations 3 >
Auto-Tuning
Database

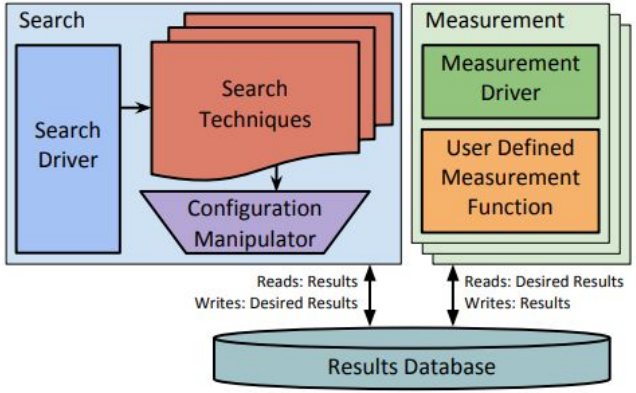
Auto-Tuning



Challenge:

- Sensible optimization schema
- “Good” optimization function
- Ensurance of correctness

Auto-Tuning



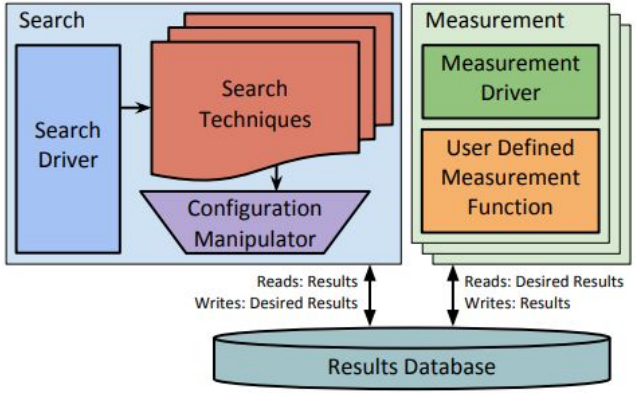
Initial state

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Challenge:

- Sensible optimization schema
- “Good” optimization function
- Ensurance of correctness

Auto-Tuning



Challenge:

- Sensible optimization schema
- “Good” optimization function
- Ensurance of correctness

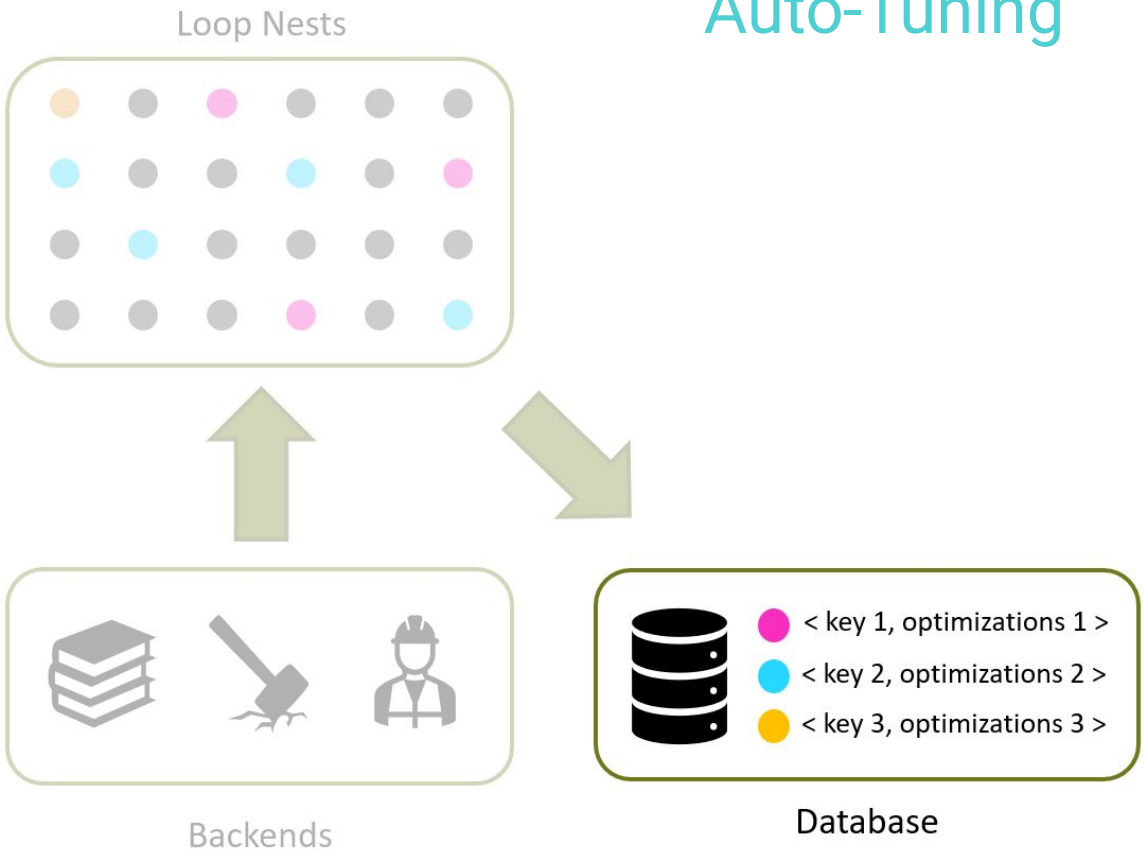
Initial state

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

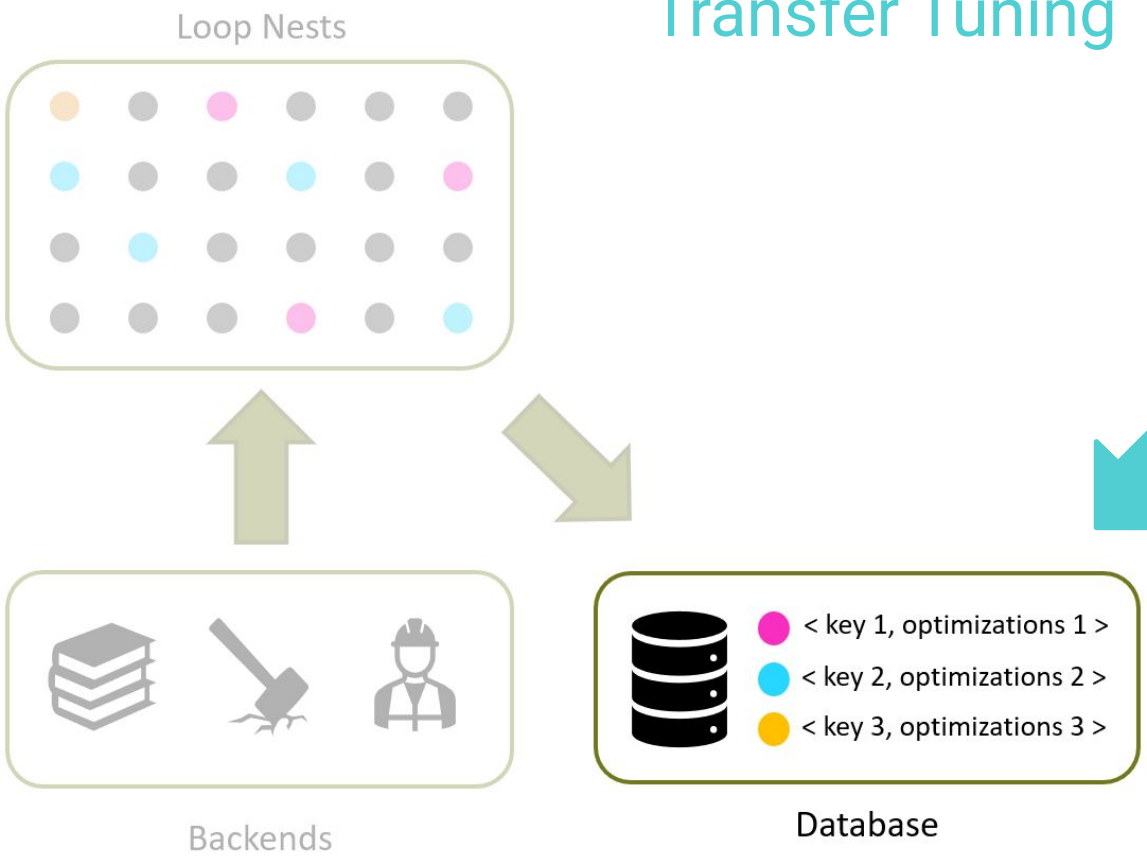
Target state

```
for (int k_tile = 0; k_tile < K; k_tile += 512) {
    for (int i_tile = 0; i_tile < N; i_tile += 32) {
        for (int j_tile = 0; j_tile < M; j_tile += 32) {
            for (int i = i_tile; i < i_tile + 32; i++) {
                for (int k = k_tile; k < k_tile + 512; k++) {
                    for (int j = j_tile; j < j_tile + 32; j++) {
                        C[i][j] += A[i][k] * B[k][j];
                    }
                }
            }
        }
    }
}
```

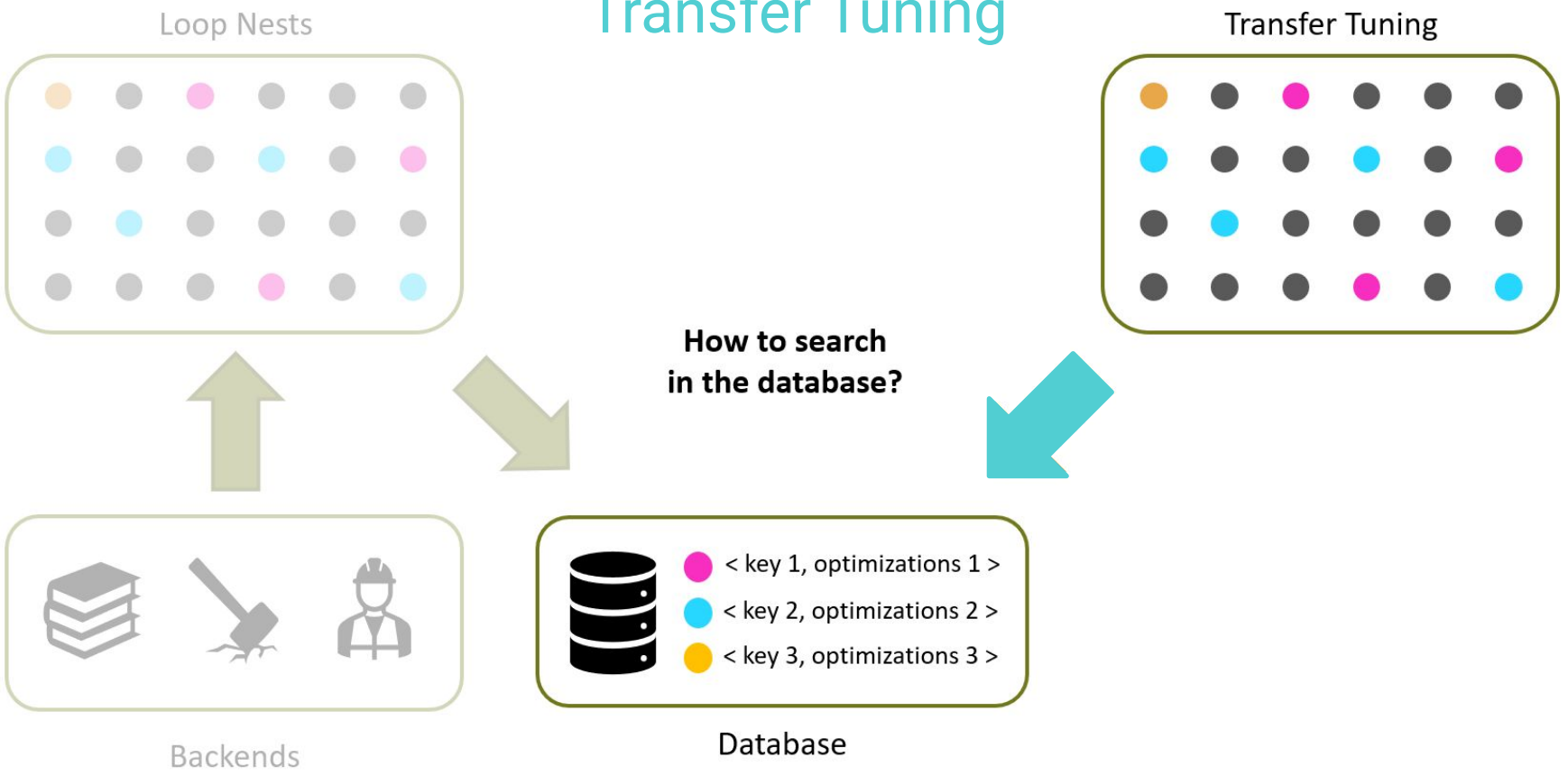
Auto-Tuning


Loop Nests
< key 1, optimizations 1 >
< key 2, optimizations 2 >
< key 3, optimizations 3 >
Backends
Database

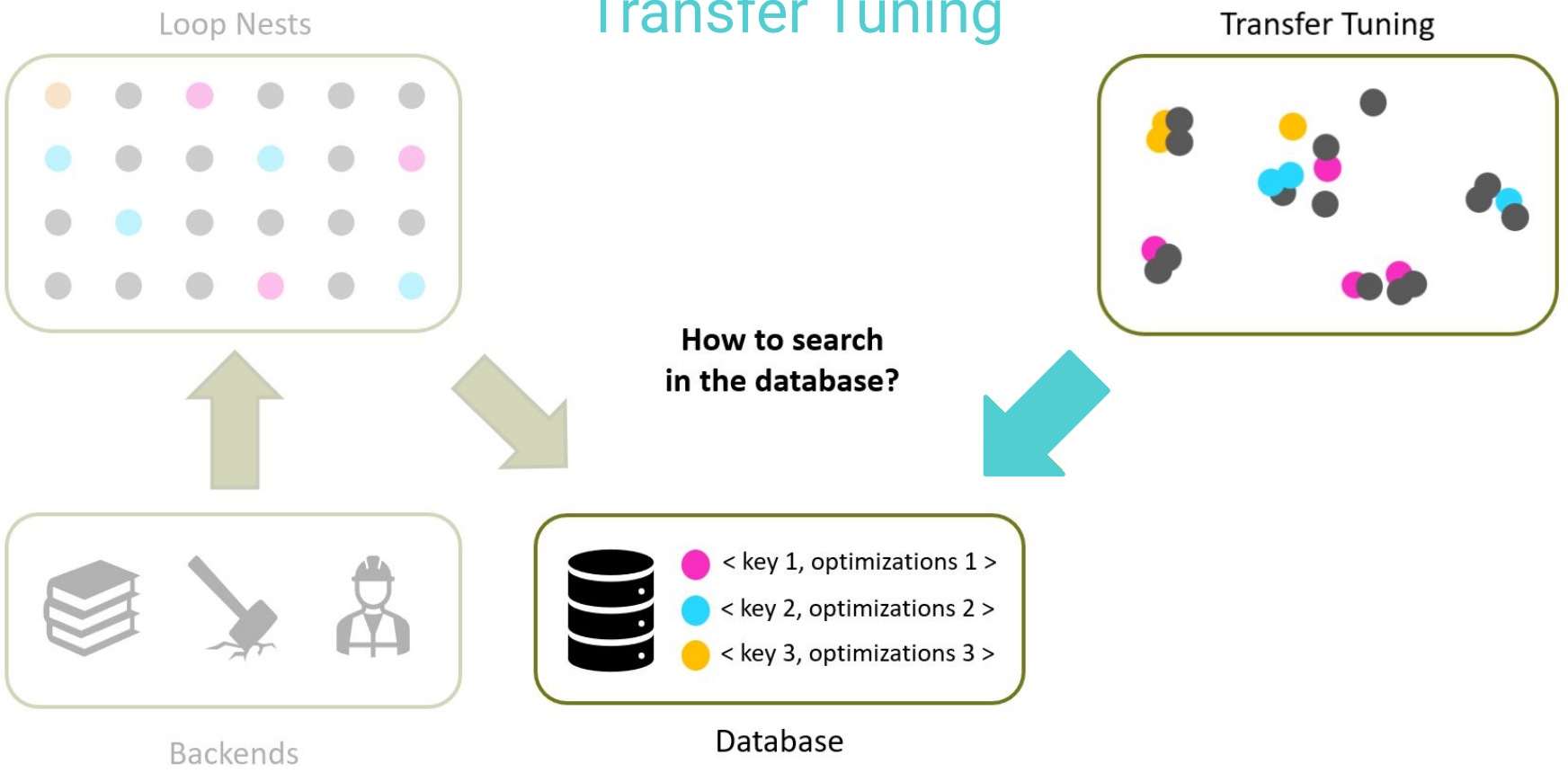
Transfer Tuning



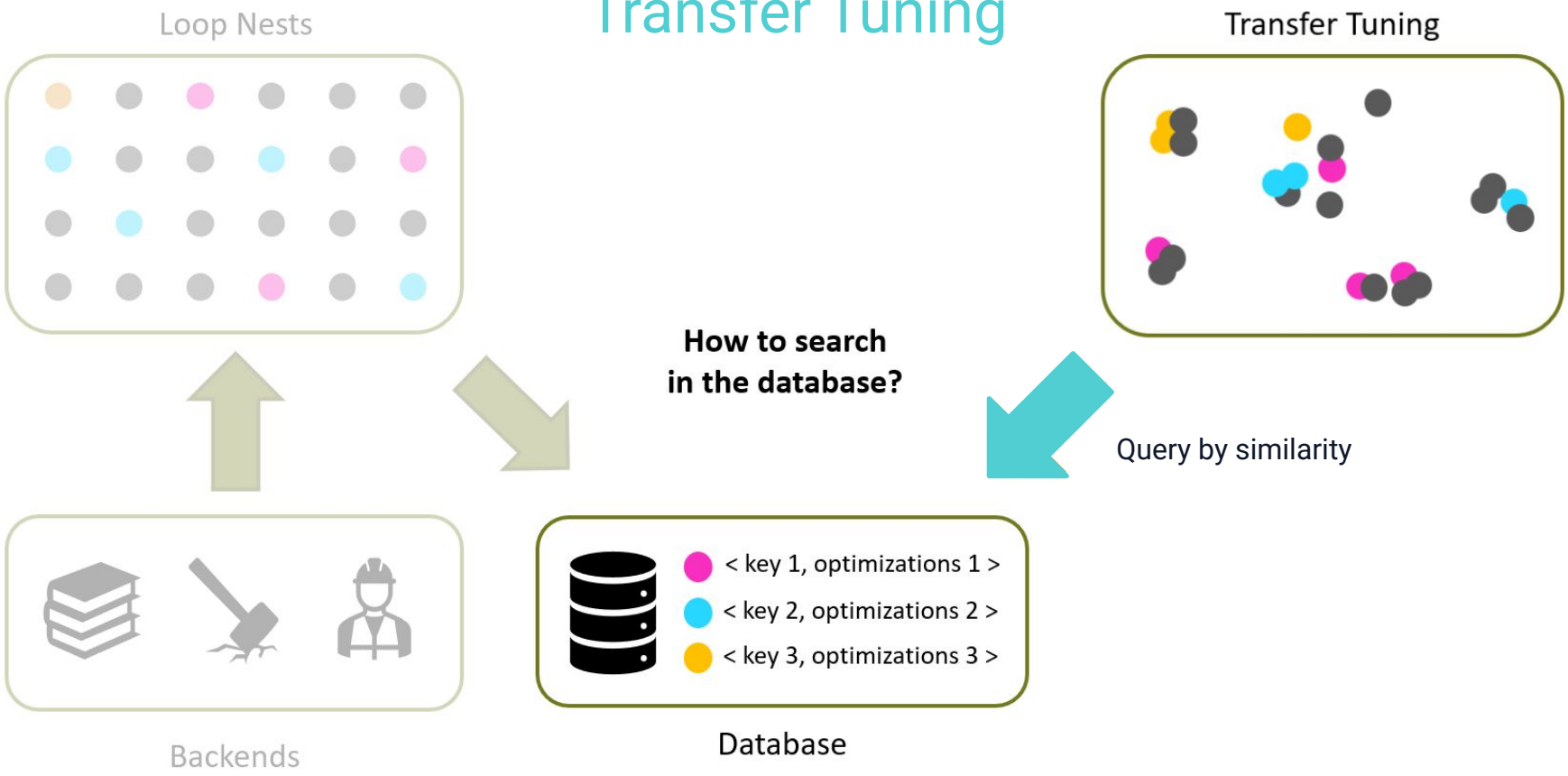
Transfer Tuning



Transfer Tuning



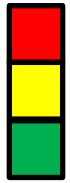
Transfer Tuning



Embeddings


```
#pragma omp parallel for
for (int i=0; i < 1024; i++)
  for (int j=0; j < 1024; j++)
    for (int k=0; k < 1024; k++)
      C[i][j] += A[i][k] * B[k][j]
```

Loop Nest



Vector Key
(embedding)

Encodes:

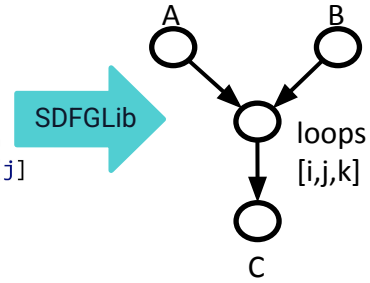
- Loop Structure
- Memory Accesses
- Memory bandwidth
- Cache misses

[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

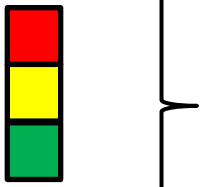
Embeddings


```
#pragma omp parallel for
for (int i=0; i < 1024; i++)
  for (int j=0; j < 1024; j++)
    for (int k=0; k < 1024; k++)
      C[i][j] += A[i][k] * B[k][j]
```

Loop Nest



SDFG[1]



Vector Key
(embedding)

Encodes:

- Loop Structure
- Memory Accesses
- Memory bandwidth
- Cache misses

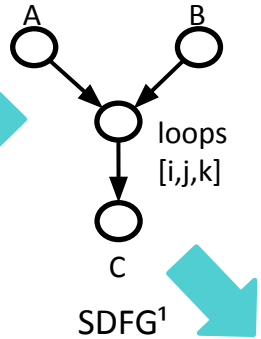
[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Embeddings

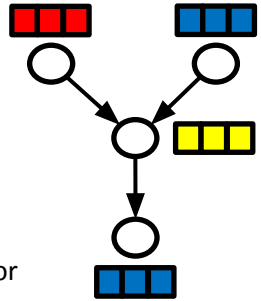

```
#pragma omp parallel for
for (int i=0; i < 1024; i++)
  for (int j=0; j < 1024; j++)
    for (int k=0; k < 1024; k++)
      C[i][j] += A[i][k] * B[k][j]
```

Loop Nest

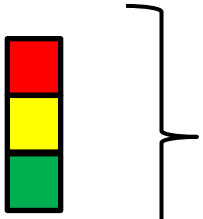
SDFGLib



SDFG[1]



Static Features



Vector Key
(embedding)

Encodes:

- Loop Structure
- Memory Accesses
- Memory bandwidth
- Cache misses

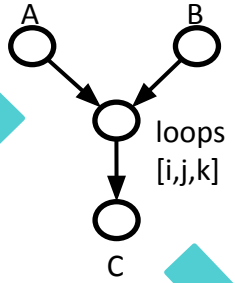
[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Embeddings

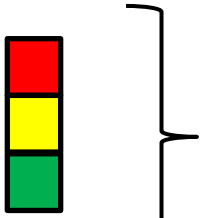

```
#pragma omp parallel for
for (int i=0; i < 1024; i++)
  for (int j=0; j < 1024; j++)
    for (int k=0; k < 1024; k++)
      C[i][j] += A[i][k] * B[k][j]
```

Loop Nest

SDFGLib



SDFG[1]

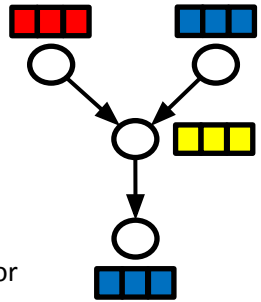


Vector Key
(embedding)

Encodes:

- Loop Structure
- Memory Accesses
- Memory bandwidth
- Cache misses

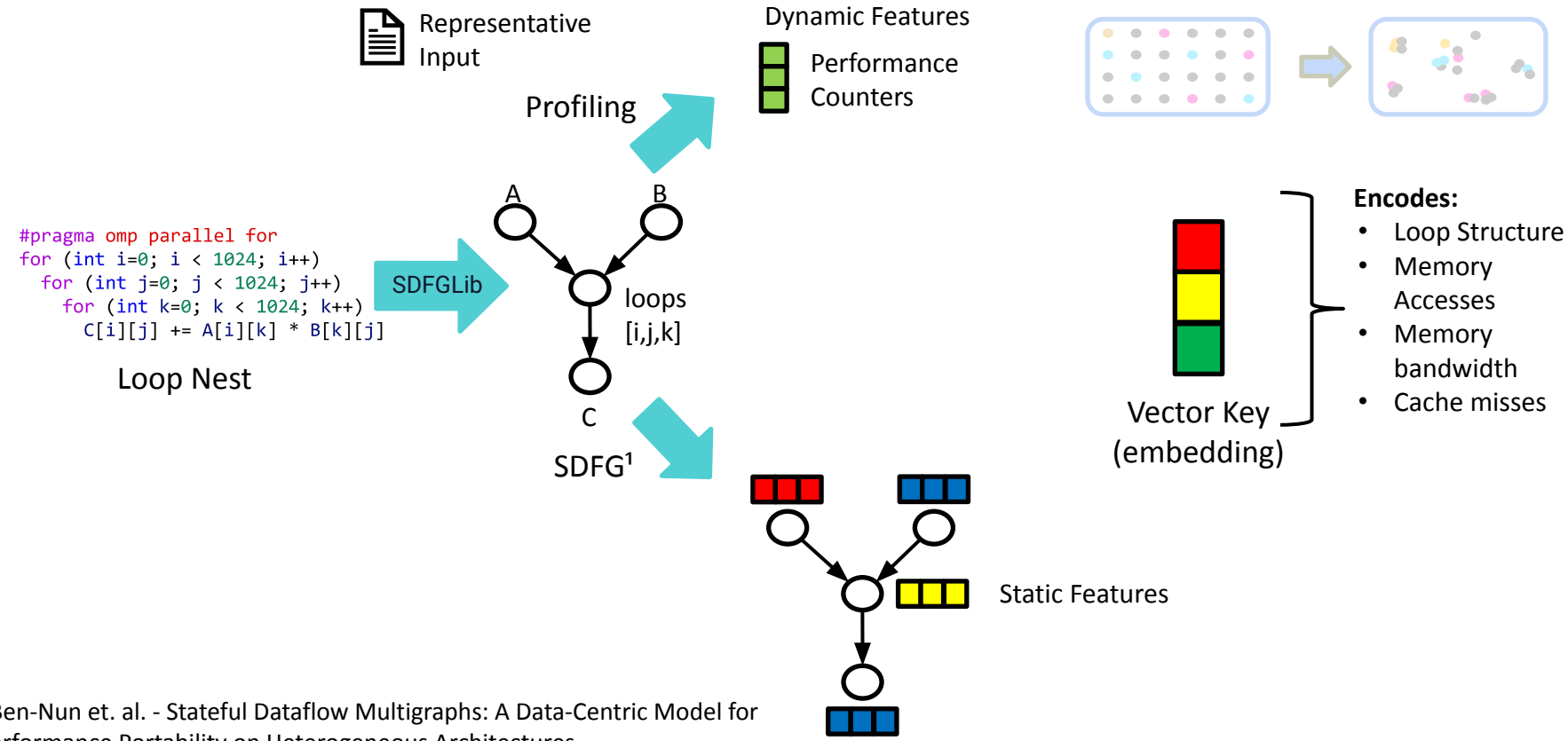
input-dependency?



Static Features

[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Embeddings



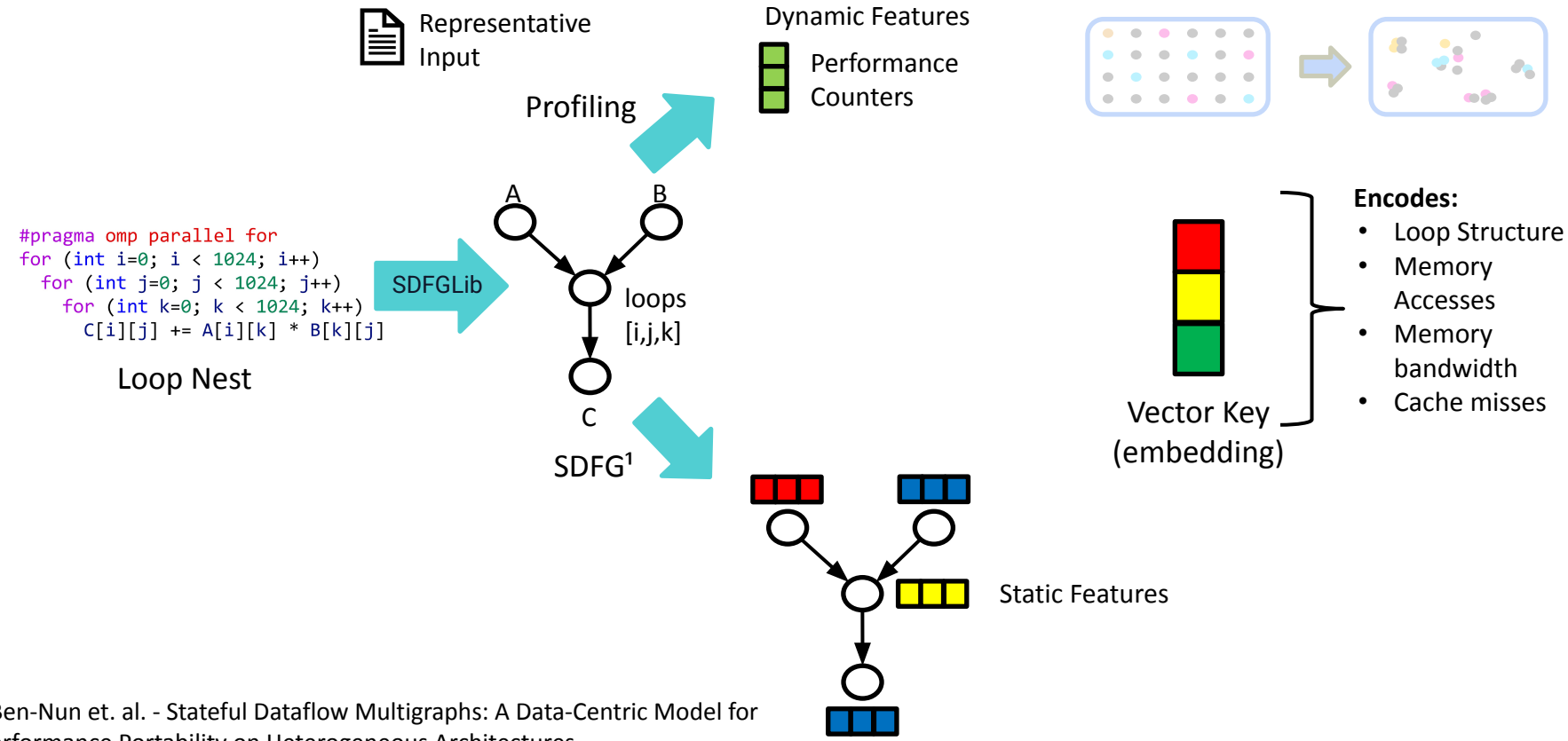
[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Embeddings

- Generate new code
- Introduce source-level instrumentation (PAPI)
- Measure representative performance counter

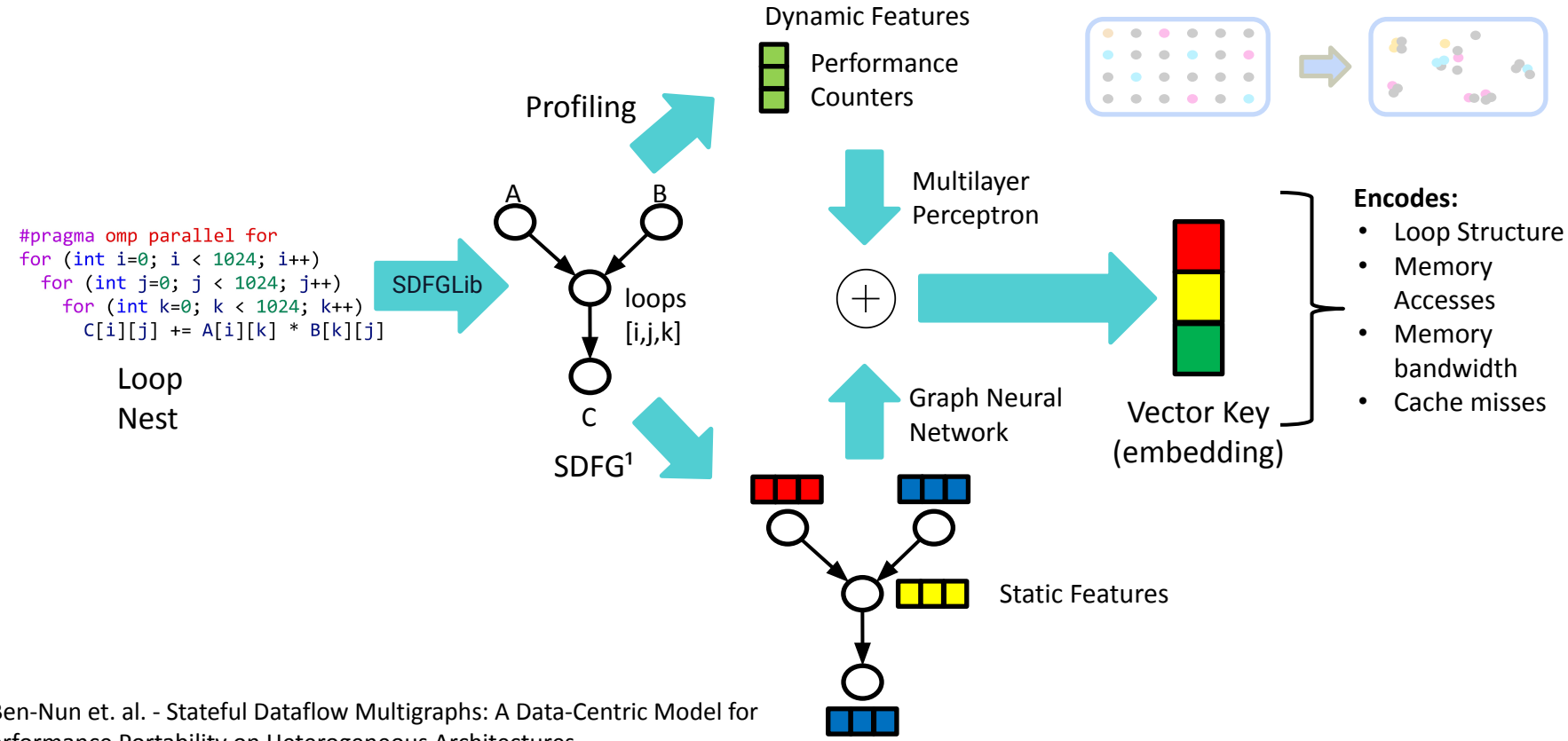
```
+-------------------------------------------------+
|                    Processor                    |
+-------------------------------------------------+
| Branch Misprediction Ratio:                0.02 |
| Cycles Per Instruction:                    0.44 |
| FLOP [DP]:                           8.28 MFLOP |
| Runtime:                              19.10 ms  |
+-------------------------------------------------+
                        |
+-------------------------------------------------+
|                     Caches                      |
+-------------------------------------------------+
| L2 Volume:                          113.16 MB   |
+-------------------------------------------------+
                        |
+-------------------------------------------------+
|                   Main Memory                   |
+-------------------------------------------------+
| Memory Volume:                       32.36 MB   |
+-------------------------------------------------+
```

Embeddings



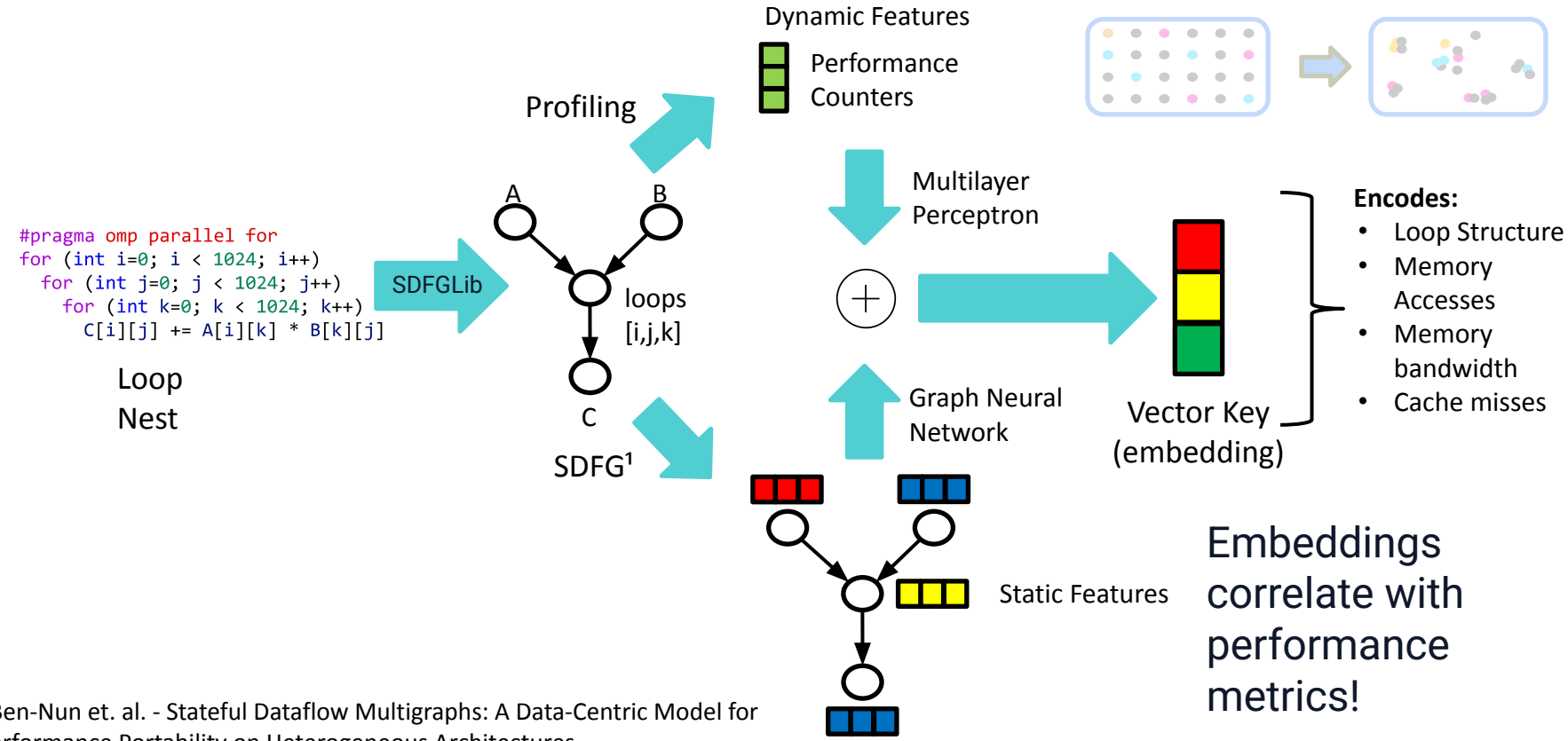
[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Embeddings



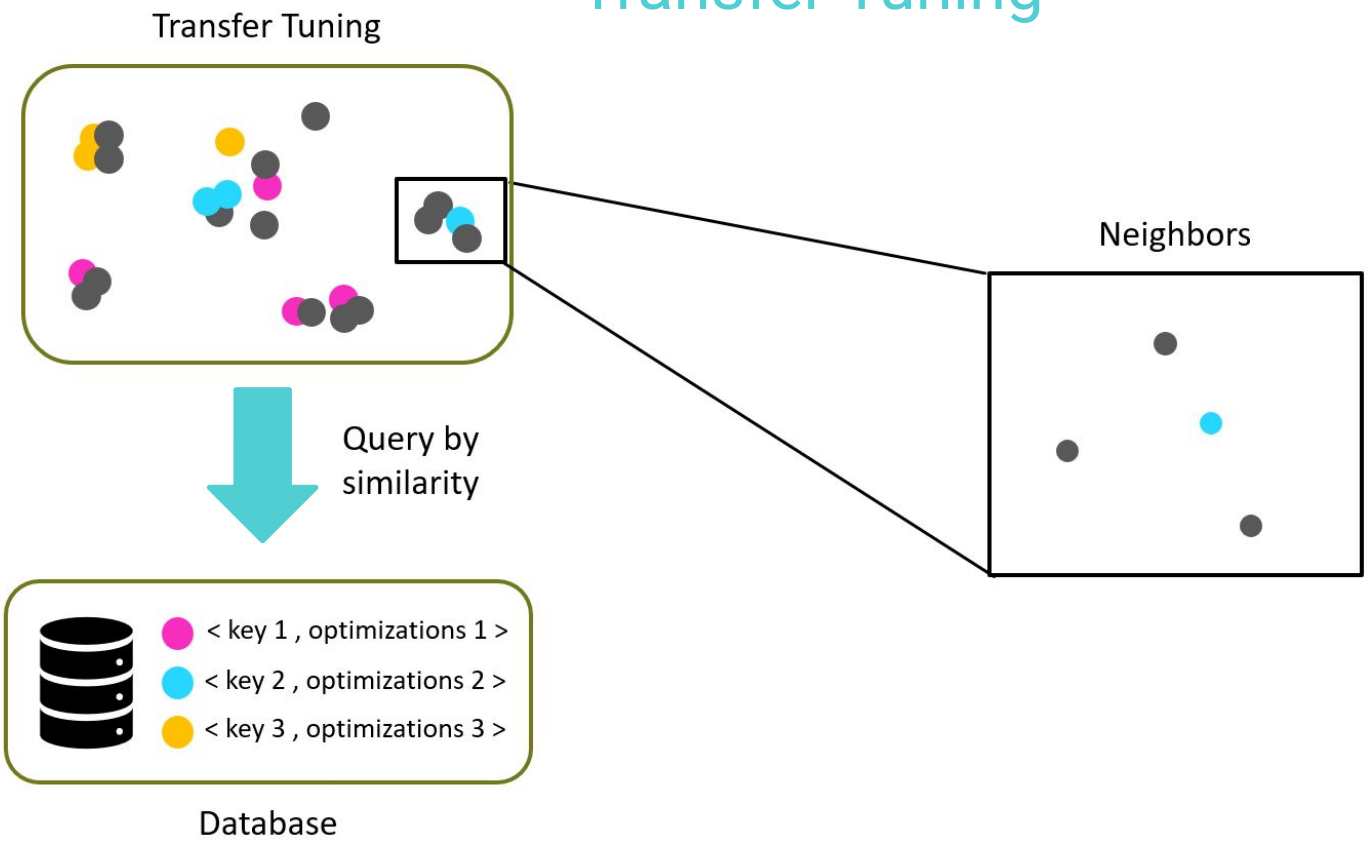
[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Embeddings

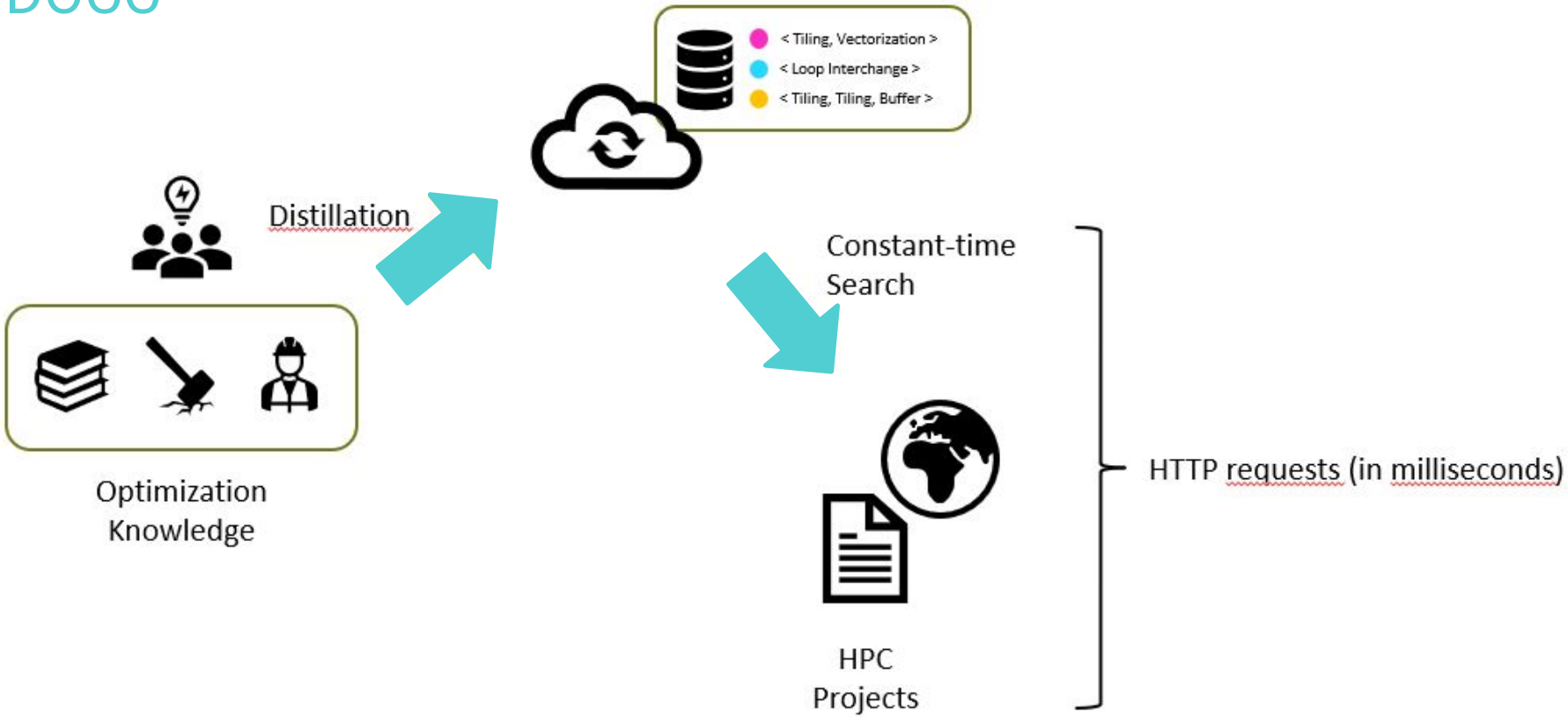


[1] Ben-Nun et. al. - Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Transfer Tuning



DOCC


< Tiling, Vectorization >
< Loop Interchange >
< Tiling, Tiling, Buffer >
Distillation
Optimization Knowledge
Constant-time Search
HPC Projects
HTTP requests (in milliseconds)

Performance Model

Challenge:

- Number of optimizable regions in practice high
- Particularly in engineering, differing program flow depending on input
- Where is the biggest optimization potential?

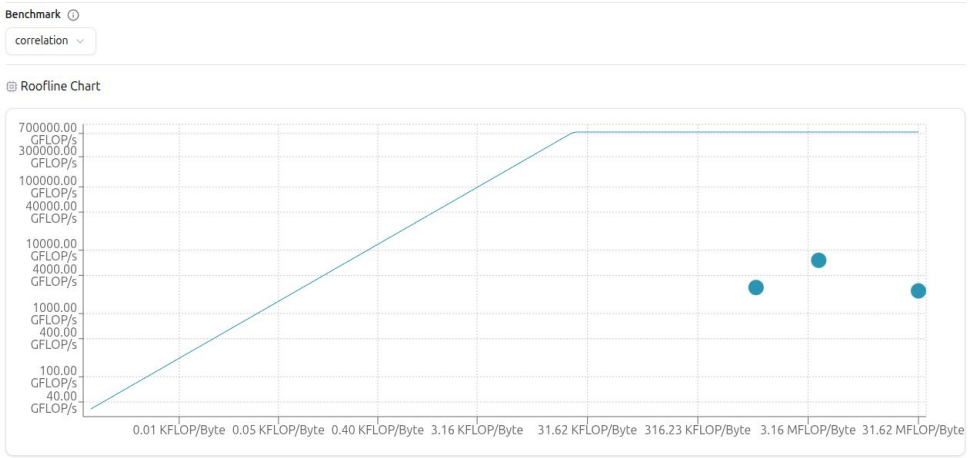
Performance Model

Challenge:

- Number of optimizable regions in practice high
- Particularly in engineering, differing program flow depending on input
- Where is the biggest optimization potential?

- Find bottlenecks
- Approximate optimization potential

Performance Model

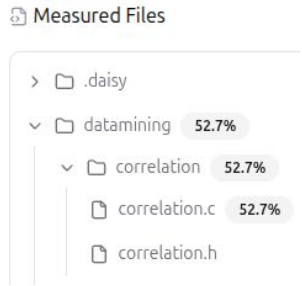


daisytuner bot commented 2 weeks ago

Daisytuner Report - polybench (raspi5)

Hotspots

Regions	Operational Intensity	Peak Performance	Relative Runtime
Region starting at line 112 in tests/polybench/datamining/correlation/correlation.c	0.39 FLOP/Byte	19.57%	47.84%
Region starting at line 103 in tests/polybench/datamining/correlation/correlation.c	0.50 FLOP/Byte	33.99%	0.69%
Region starting at line 90 in tests/polybench/datamining/correlation/correlation.c	0.18 FLOP/Byte	68.37%	0.49%
Region starting at line 81 in tests/polybench/datamining/correlation/correlation.c	0.08 FLOP/Byte	74.41%	0.51%
Region starting at line 36 in tests/polybench/datamining/correlation/correlation.c	24.00 FLOP/Byte	18.47%	2.60%



Daisytuner

daisytuner.com

Thanks! Questions?

Yesterday, your code was fast. Today, it is slow. Why?

Continuous Benchmarking Continuous Tuning

Daisytuner adds performance analysis to your CI pipelines. By combining profiling data with performance models, our platform tells you *where you spend your time* and *how close you are to peak performance*.

What is Loop Scheduling

HPC
Application

Focus on Loop Nests

```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

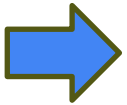
What is Loop Scheduling

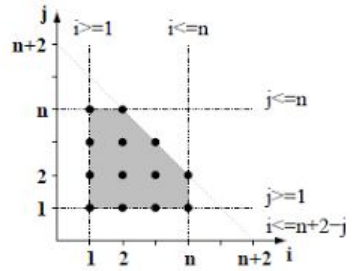
HPC
Application

Focus on Loop Nests

```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

Lifting





Representation, e.g., polyhedral model stateful dataflow multigraphs

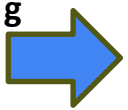
What is Loop Scheduling

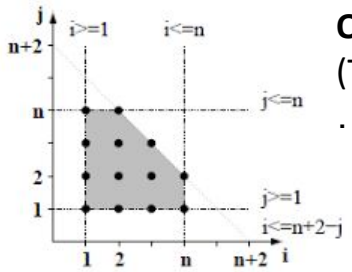
HPC Application

Focus on Loop Nests

```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

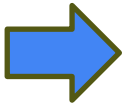
Lifting

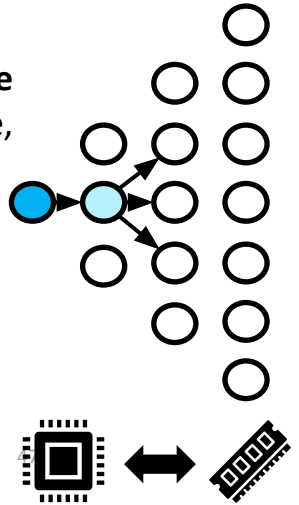




Representation, e.g., polyhedral model stateful dataflow multigraphs

Optimization Space (Tiling, Interchange, …)





Optimize Performance

Normalization Criteria - Intuition

Matmul A (naive)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul B (k swap)

```
for (i = 0; i < N; i++) {
  for (k = 0; k < K; k++) {
    for (j = 0; j < M; j++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul C (with init)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    double sum = 0;
    for (k = 0; k < K; k++) {
      sum += A[i][k] * B[k][j];
    }
    C[i][j] = sum;
  }
}
```

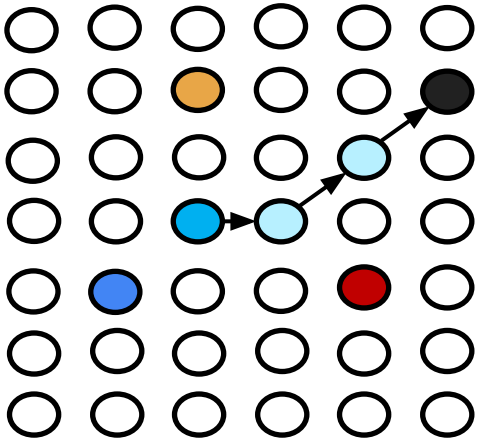
A Priori Loop Nest Normalization - Idea

Designated Initial State

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Algorithmic

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```



Language-specific Variation

```
C = A @ B
```

Pre-Optimized

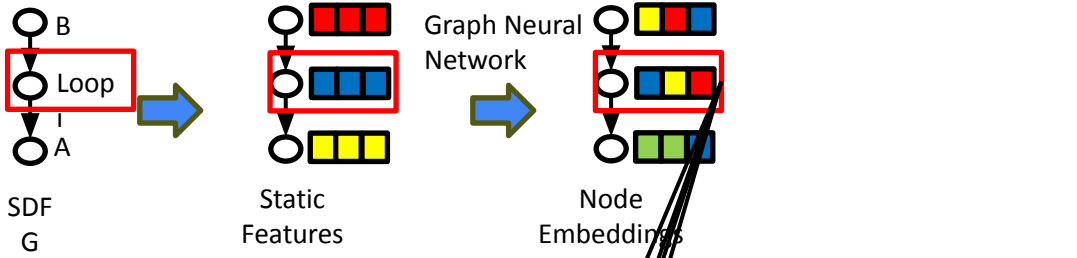
```
for (int i_tile = 0; i_tile < N; i_tile += 8) {
    for (int j_tile = 0; j_tile < M; j_tile += 8) {
        for (int i = i_tile; i < i_tile + 8; i++) {
            for (int k = 0; k < K; k++) {
                for (int j = j_tile; j < j_tile + 8; j++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }
}
```

Matching Program Transformations

Optimization

```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

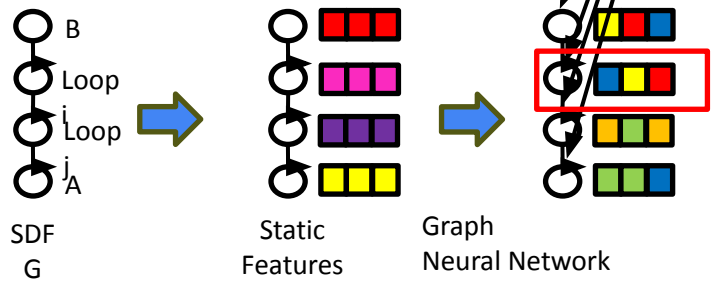
```
#pragma omp parallel for
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```



Hungarian Method
(Kuhn–Munkres algorithm)

Query Loop Nest

```
for (int i=1; i < 10000; i++) {
    A[i] = 0;
    for (int j=-1; j <= 1; j++)
        A[i] += B[i+j];
}
```



Performance Embeddings - Evaluation

Model	Main Memory Bandwidth	Data Locality
Reuse Distance	0.78	0.87
IR2Vec[1]	0.47	0.41
Tiramisu[2]	0.32	0.35
Our Model	**0.25**	**0.31**

Mean variation of neighbors for different performance metrics. A lower value is better.

[1] VenkataKeerthy et al. - IR2Vec: LLVM IR based Scalable Program Embeddings

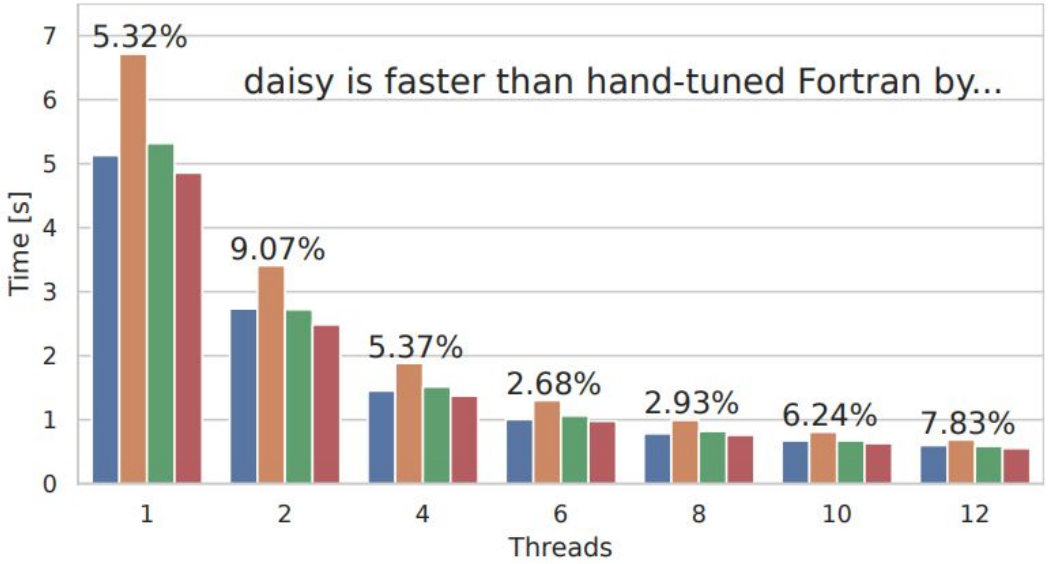
[2] Baghdadi et al. - A Deep Learning Based Cost Model for Automatic Code Optimization

Evaluation

Benchmark	Explored States	Reference [ms]	Transfer Tuning (k=5)	Transfer Tuning (k=10)
mlp	111,508	1.47	+ 38.8%	**+ 37.4%**
softmax	183,427	110.40	+ 0.6%	+ 0.5%
blur filter	1,342	1.03	0.0	0.0
daubechies wavelet	9,101	8.73	- 3.7%	**- 92.0%**
haar wavelet	8,639	0.22	0.0%	0.0%
harris filter	1,651	9.06	+ 0.2%	- 4.0%
histogram filter	147,438	32.51	+ 1.2%	- 4.9%
unsharpening filter	25,080	29.66	+ 3.1%	+ 0.5%
heat 3D	69,080	13428.98	+ 3.6%	+ 2.8%
horizontal diffusion	34,534	7.00	+ 4.8%	+ 2.8%
matmul	65,986	14.17	+ 5.3%	+ 4.1%
min-plus mm	65,999	24.76	+ 9.0%	+ 8.5%

Number of explored states and optimized runtime of the Auto-Scheduler compared to Transfer Tuning a constant number of hypotheses.

Transfer Tuning: CLOUDSC



(a) Strong scaling behavior of CLOUDSC for the Fortran, C, DaCe, and daisy versions from left to right and grouped by the number of threads.