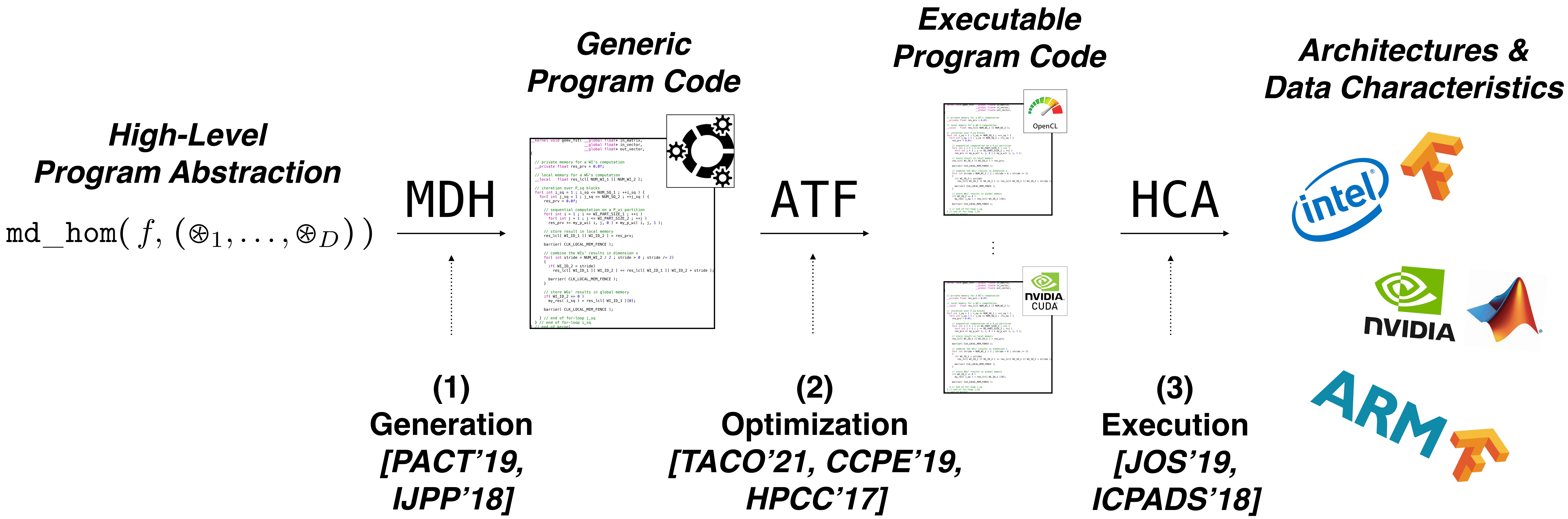
Auto-Tuning Framework (ATF)

Ari Rasch, Richard Schulze, …

Who are we?

<u>We are the developers of the **MDH+ATF+HCA** approach:</u>



Richard Schulze

Ari Rasch

A holistic approach to code *generation* (MDH) & *optimization* (ATF) & *execution* (HCA):

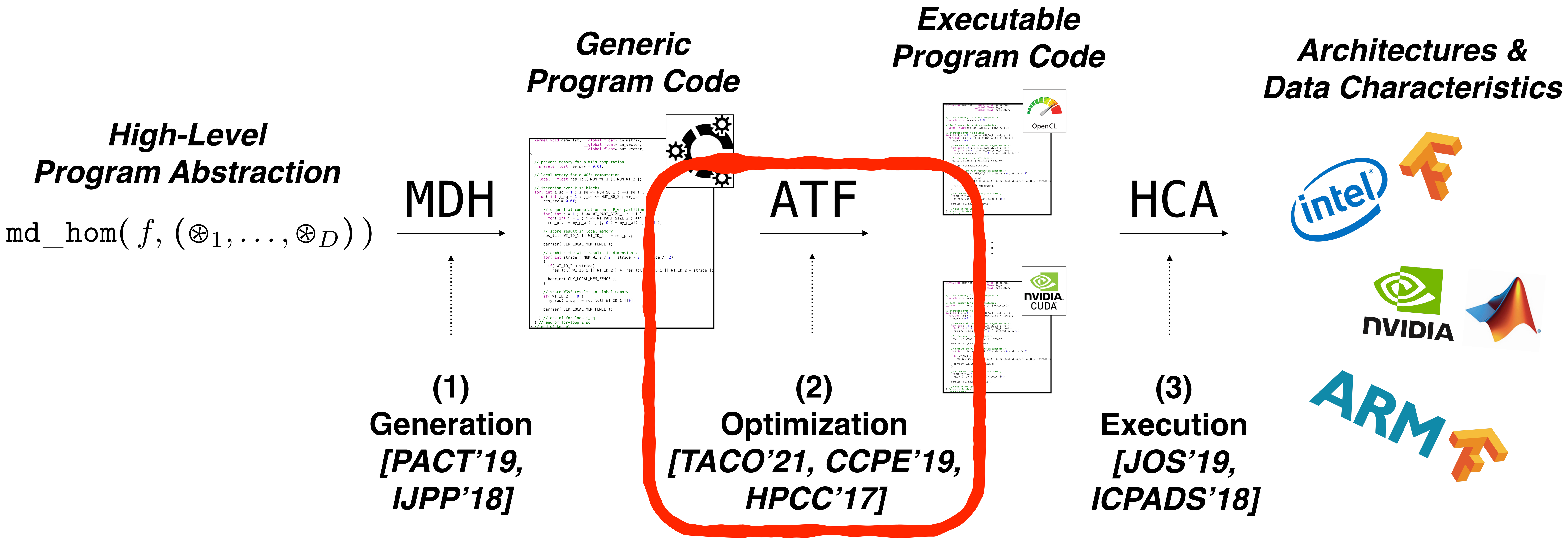
(1) <u>MDH (*Multi-Dimensional Homomorphisms*):</u> How to generate auto-tunable code?

(2) <u>ATF (*Auto-Tuning Framework*):</u> How to auto-tune code?

(3) <u>HCA (*Host Code Abstraction*):</u> How to execute code on (distr.) multi-dev. systems?

Who are we?

We are the developers of the **MDH+ATF+HCA** approach:



Richard Schulze

Ari Rasch

A holistic approach to code *generation* (MDH) & *optimization* (ATF) & *execution* (HCA):

(1) MDH (*Multi-Dimensional H* … generate auto-tunable code?

(2) ATF (*Auto-Tuning Framew* … ode?

(3) HCA (*Host Code Abstraction*): How to execute code?

our focus this week

ATF — Yet another Auto-Tuning Framework?

We have seen *KernelTuner (KT)* & *KernelTuningToolkit (KTT)* — why do we need ATF?

ATF [1] efficiently handles

***interdependencies* among tuning parameters**

via optimized processes to

generating & *storing* & *exploring*

the spaces of interdependent parameters.

→ **We illustrate ATF by comparing it to *CLTune* [MCSoC'15] — the foundation of KT & KTT.**

[1] Rasch, Schulze, Steuwer, Gorlatch, "Efficient Auto-Tuning of Parallel Programs with Interdependent Tuning Parameters via Auto-Tuning Framework (ATF)", TACO'21

ATF — Yet another Auto-Tuning Framework?

Excursion: *Interdependent Tuning Parameters* (in a nutshell)

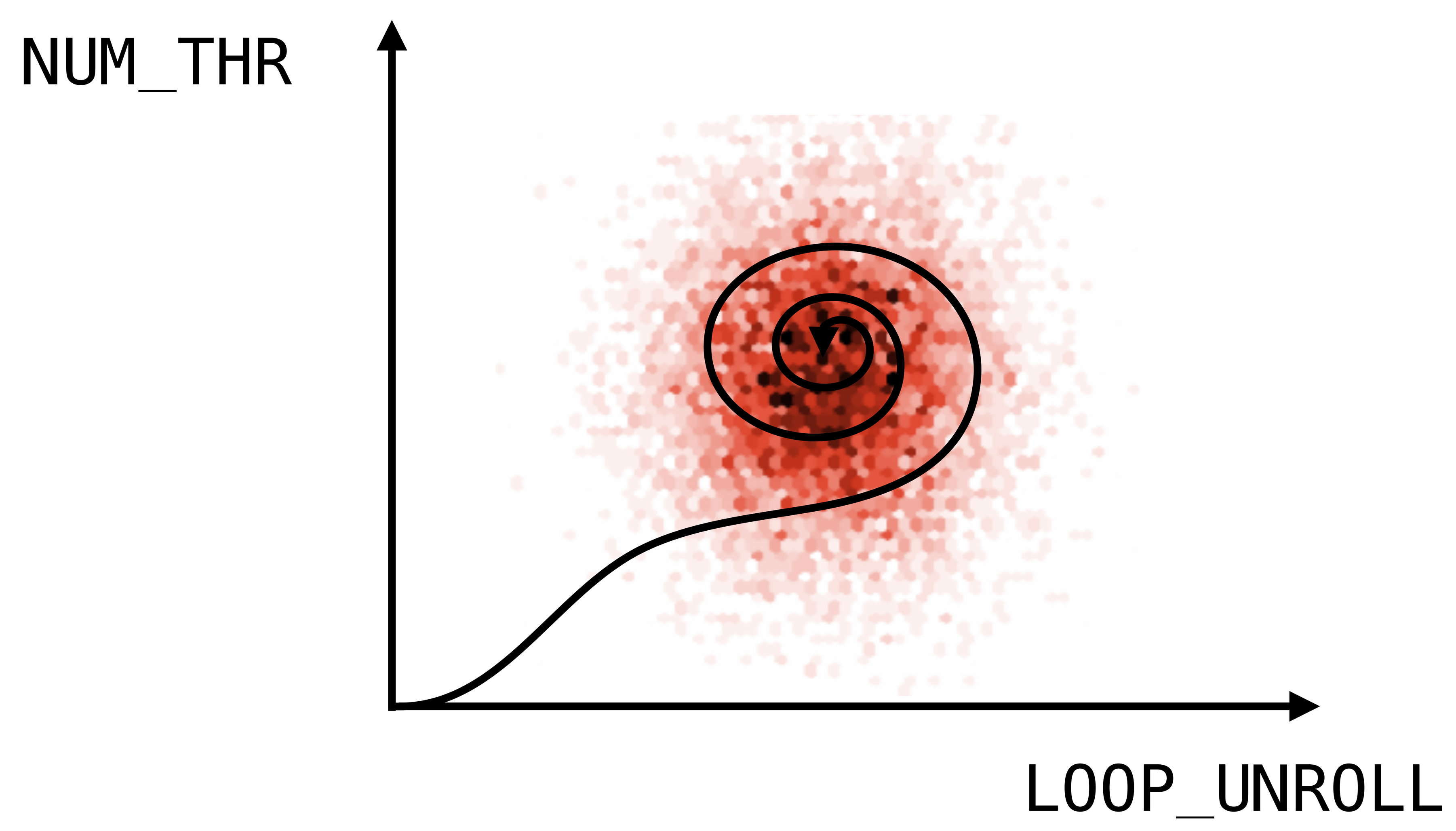
- Classic approaches (like OpenTuner [PACT'14]) are inherently designed toward parameters without interdependencies among them, e.g.:
  - `NUM_THR ∈ int`
  - `LOOP_UNROLL ∈ int`
- Parameter configurations: (`NUM_THR, LOOP_UNROLL) ∈ int x int`
- Each configuration is considered as valid by the classic approaches!

Problem:

Applications for modern architectures often have *interdependencies* among their tuning parameters

What happens inside the classic tuners:

Search Space



*The space is explored using a **search technique**.*

Main challenge in classic approaches

ATF — Yet another Auto-Tuning Framework?

Excursion: *Interdependent Tuning Parameters* (in a nutshell)

- We briefly illustrate *interdependent tuning parameters* using two simple, example parameters from OpenCL — a *modern* programming approach:
  - `GS ∈ int` (*global size*): total numbers of threads
  - `LS ∈ int` (*local size*): number of threads per group
- Parameter configurations are of the form: `(GS,LS) ∈ int x int`
- Configurations are valid *iff(!)* they satisfy the following two *constraints*:
  1. `LS` divides `GS`
  2. `GS` is smaller than or equal to the input size `N`
- For example, for `N=8`, configuration `(4,2)` is valid, but `(4,3)` and `(10,5)` are not valid.

Modern approaches like CLTune (KT, KTT, …) allow interdependencies among tuning parameters

ATF — Yet another Auto-Tuning Framework?

Excursion: *Interdependent Tuning Parameters* (in a nutshell)

As CLTune (pseudo)code:

```
int N = /* input size */ ;

tuner.addParameter( “GS”, int );
tuner.addParameter( “LS”, int );

tuner.addConstraint(

  (int GS, int LS)
  {

    return (GS % LS == 0) && (GS <= N);
  }

);

tuner.tune( application );
```

Summary (slide 6)

- `GS ∈ int`
- `LS ∈ int`
- **LS** divides **GS**
- **GS** is smaller/equal **N**

`addConstraint()` *not available in classic auto-tuning approaches (like OpenTuner)*

ATF — Yet another Auto-Tuning Framework?

Excursion: *Interdependent Tuning Parameters* (in a nutshell)

Generation

Configurations are only added to the space iff the constraint is satisfied

```
for GS ∈ int
  for LS ∈ int
  {
    if( check_constraint() )
      add_config( GS,LS );
  }
```

Storing

Valid configurations are stored in an array

```
SP :=[ (1,1) | (2,1) | (2,2) | … ]
```

Search Space *(1D Array)*

Exploration

Space is explored via a search technique (e.g., simulated annealing)

```
best_config = explore_1d_space(
  search_technique,
  SP,
  application
);
```

CLTune: Behind the Scenes

ATF — Yet another Auto-Tuning Framework?

Why ATF when we have CLTune?

M,N,K = 1024	CLTune	ATF
Generation Time	***>10^12 Years!***	***6 min***

Analysis of CLTune's process to *generating* search spaces, using CLTune's own GEMM example *(14 tuning parameters with various interdependencies)*

→ CLTune requires from user **hand-pruning** parameter ranges, e.g., using {16, 32, 64, 128} instead of {1,….,N} for tile sizes.

need to perform well for all potential target architectures and input sizes

CLTune requires **hand pruning** also for *Storing* and *Exploration* processes [TACO'21]:

CLTune's *Storing Process* memory intensive:
CLTune stores parameter configurations in 1D array
→ many redundancies

CLTune's *Exploration* Process hampered:
CLTune explores 1D index range of array only,
→ looses space's multi-dimensional locality information

Problem: Hand pruning search spaces is complex
→ well performing parameter configurations are often not intuitive

ATF — Yet another Auto-Tuning Framework?

Real-World Input Sizes from DL

Severeness of CLTune's hand pruning for its GEMM example:

Speedup (higher is better) of CLTune's GEMM auto-tuned via ATF over auto-tuning via CLTune on Intel CPU (top part of figure) and NVIDIA GPU (bottom part).

By avoiding hand-pruned parameter ranges, ATF achieves up to 38x better performance than CLTune for CLTune's GEMM example.

ATF — Yet another Auto-Tuning Framework?

In a nutshell

How does ATF achieve its efficiency for interdependent tuning parameters:

ATF introduces *parameter constraints*

```
tuner.addParameter( “tp_1”, T1 );
tuner.addParameter( “tp_2", T2 );
// ...

tuner.addConstraint(
  [](T1 tp_1, T2 tp_2, … ) -> bool
  { /* ... */ }
)
```

CLTune constraints

Defined on:
search space **(CLTune)**
vs. ***parameters*** **(ATF)**

```
tuner.addParameter( “tp_1”, T1, [](T1 tp_1) -> bool { /* … */ } );
tuner.addParameter( “tp_2”, T2, [](T2 tp_2) -> bool { /* … */ } );
```

ATF constraints

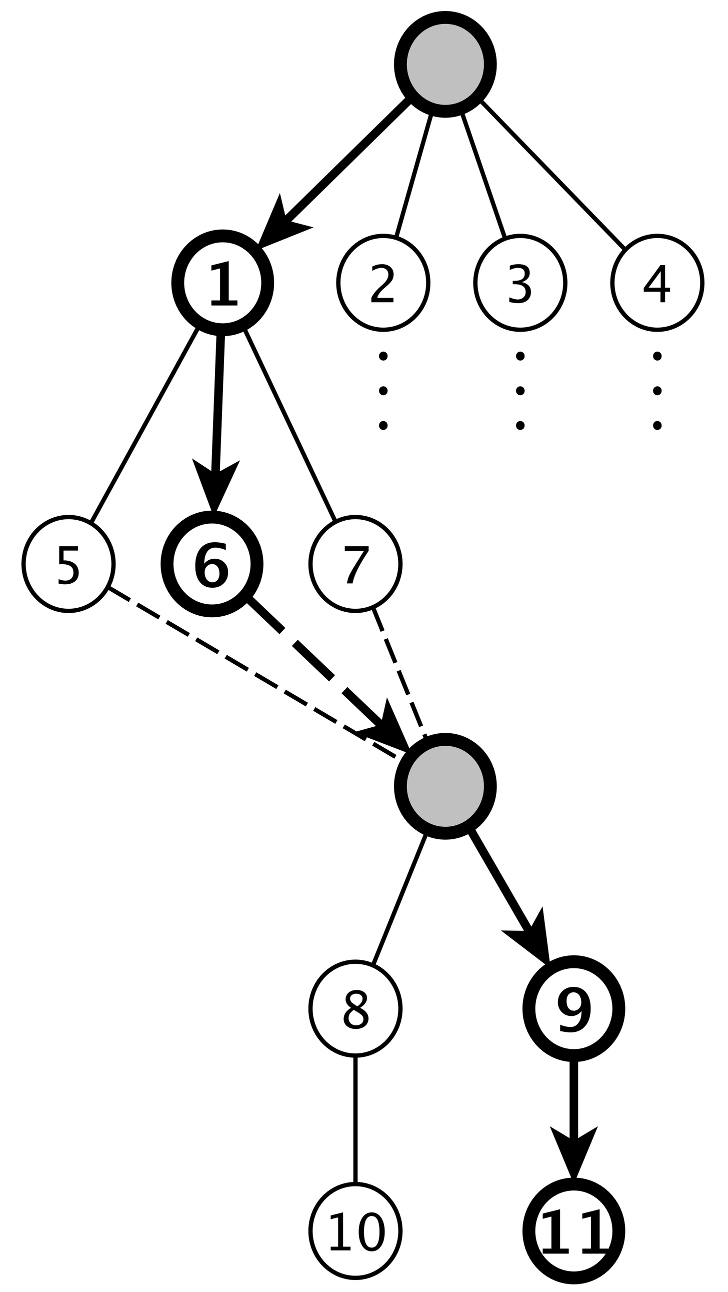
ATF introduces the *chain-of-trees* space structure

```
SP := [ (1,1) | (2,1) | (2,2) | … ]
```

CLTune search space

verbose & 1D **(CLTune)**
vs. ***compact & nD*** **(ATF)**

ATF search space



ATF introduces novel processes to
generating & ***storing*** & ***exploring***
the spaces of interdependent tuning parameters, based on its
constraint design and *search space structure*.

In the Following

<u>We *briefly* outline the following agenda points:</u>

1. ATF's processes to search space:
   i. generation
   ii. storing
   iii. exploration
2. Experimental Results
3. ATF's User Interface
   i. DSL-Based (offline tuning)
   ii. GPL-based (online tuning)
3. Summary
4. Current State & Future Work

We are happy to discuss details throughout the week

1. Generation

Basic Idea

ATF relies on *parameter constraints (PC)*, rather than *search space constraints (SC)*:

```
for ( v1 : r1 )
 ⋱
   for ( vk : rk )
     if( sc(v1,…,vk) )
       add_config( v1,…,vk );
```

Search Space Constraint (SC)

CLTune's Approach

PCs enable generating groups of interdependent parameters independently & in parallel

PCs enable generating individual groups in parallel

PCs enable checking constraints early in the loop nest

```
parallel_for ( G : {G1,…,Gn} )
{
  parallel_for ( v1^G : r1^G )
    if( pc1^G(v1^G) )
   ⋱
     parallel_for ( vtg^G : rtg^G )
       if( pctg^G(vtg^G) )
         for ( vtg+1^G : rtg+1^G )
           if( pc(vtg+1^G) )
          ⋱
            for( vk^G : rk^G )
              if( pc(vk^G) )
                add_config( v1^G,…,vk^G );
}
```

Parameter Constraint (PC)

ATF's Approach

ATF's parameter constraints contain additional semantic information, enabling:

1. Generating groups of interdependent parameters *independently* & in *parallel*
2. Generating individual groups in *parallel*
3. Checking constraints *early*

2. Storing

Basic Idea

ATF relies on a new *chain-of-trees* search space structure & *parameter constraints*:

$p_1 := ($ $n_1,$	$\{22, 35\},$	_	$)$
$p_2 := ($ $n_2,$	$\{2, 5, 7, 11\},$	`divides`(n_1)	$)$
$p_3 := ($ $n_3,$	$\{26, 51\},$	_	$)$
$p_4 := ($ $n_4,$	$\{1, 3, 13, 17\},$	`divides`(n_3)	$)$
$p_5 := ($ $n_5,$	$\{27, 39, 52, 54, 68\},$	`equals`$(n_3 + n_4)$	$)$

Example Parameters

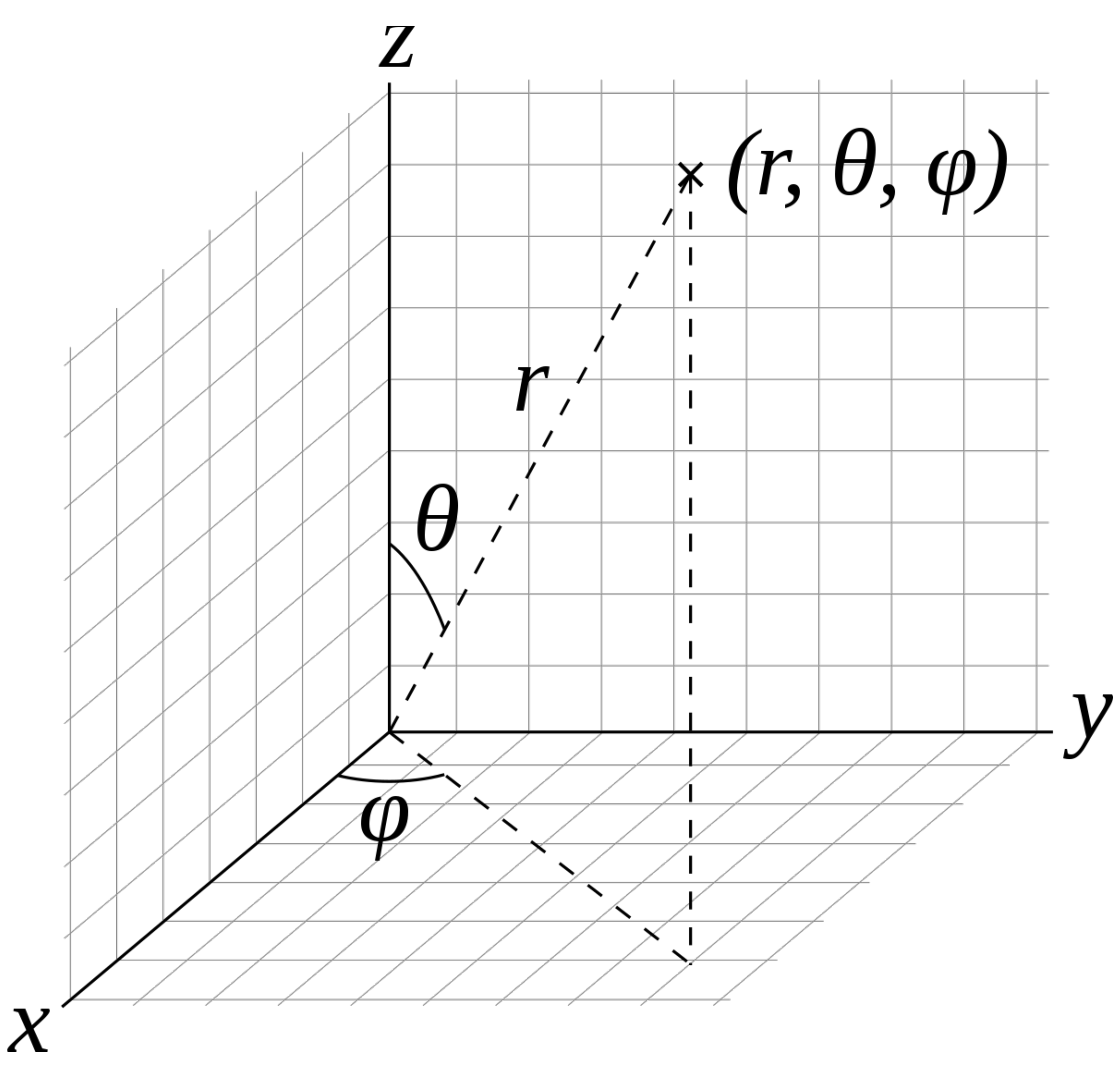
ATF's search space structure avoids memory-intensive redundancies

3. Exploration

Basic Idea

ATF exploits its *chain-of-trees* search space structure for a multi-dimensional search:

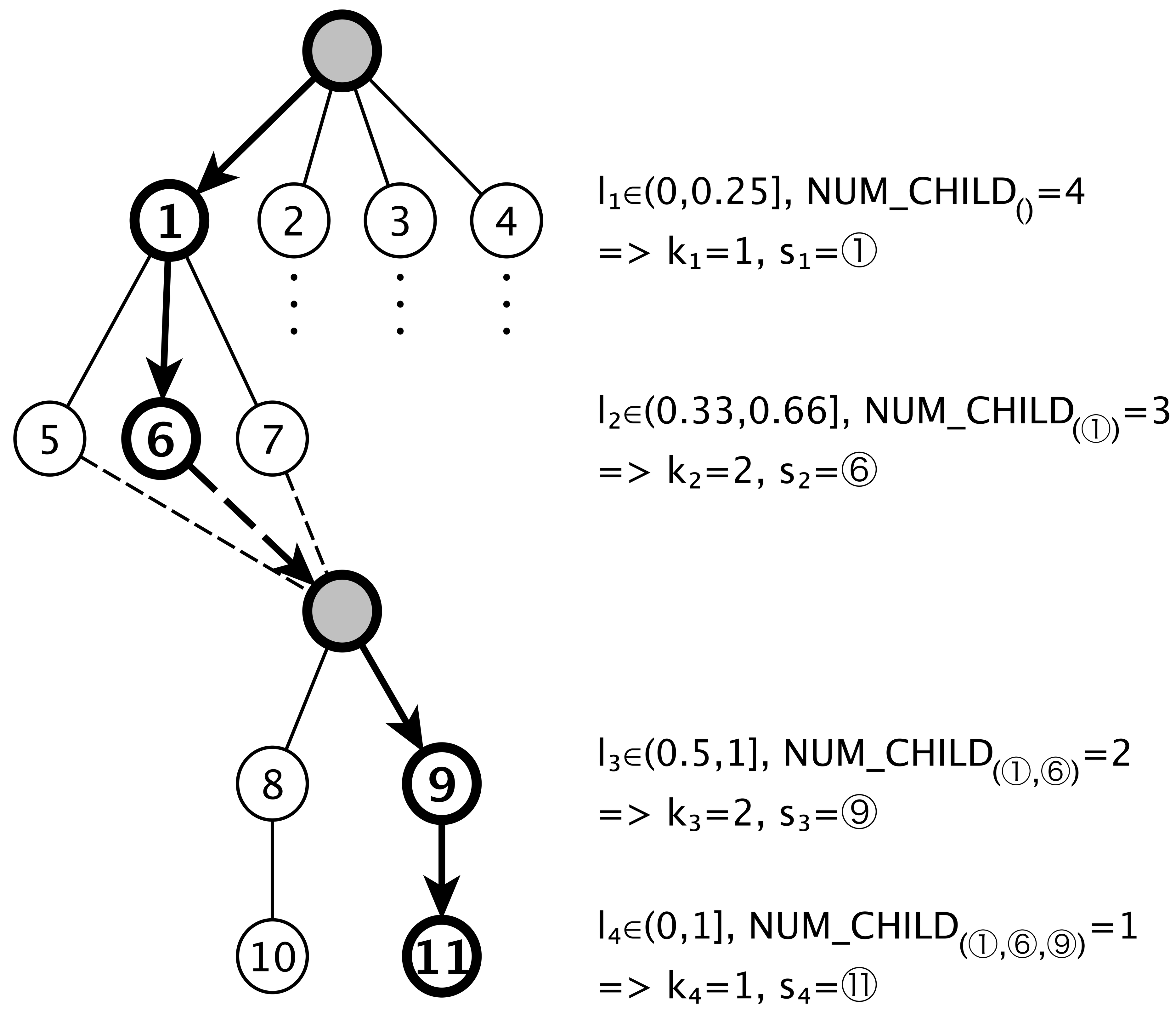
Coordinate Space



Efficient Structure for Search Techniques

mapping

ATF's Search Space



ATF's search space structure enables reducing the complexity of exploration to exploring a *Coordinate Space*

Experimental Evaluation

Highlights from TACO'21

ATF is able to auto-tune important applications for ***CPU & GPU*** on ***real-world data sets*** to high performance:

Stencil

ATF is able to auto-tune the `CONV` implementation in [2] to:

>40x higher performance than CONV+CLTune on CPU

>10⁴x higher performance than CONV+CLTune on GPU

>3x higher performance than Intel MKL-DNN on CPU

>15x higher performance than NVIDIA cuDNN on GPU

Linear Algebra

ATF is able to auto-tune the `GEMM` implementation in [2] to:

>2x higher performance than GEMM+CLTune on CPU

>120x higher performance than GEMM+CLTune on GPU

>2x higher performance than Intel MKL on CPU

>2x higher performance than NVIDIA cuBLAS on GPU

Quantum Chemistry

ATF is able to auto-tune the `CCSD(T)` implementation in [2] to:

>2x higher performance than TensorComprehensions on GPU

CLTune **fails**! (too high search space generation time)

Data Mining

ATF is able to auto-tune the `PRL` implementation in [2] to:

>1.66x higher performance than PRL+CLTune on CPU

>1.07x higher performance than PRL+CLTune on GPU

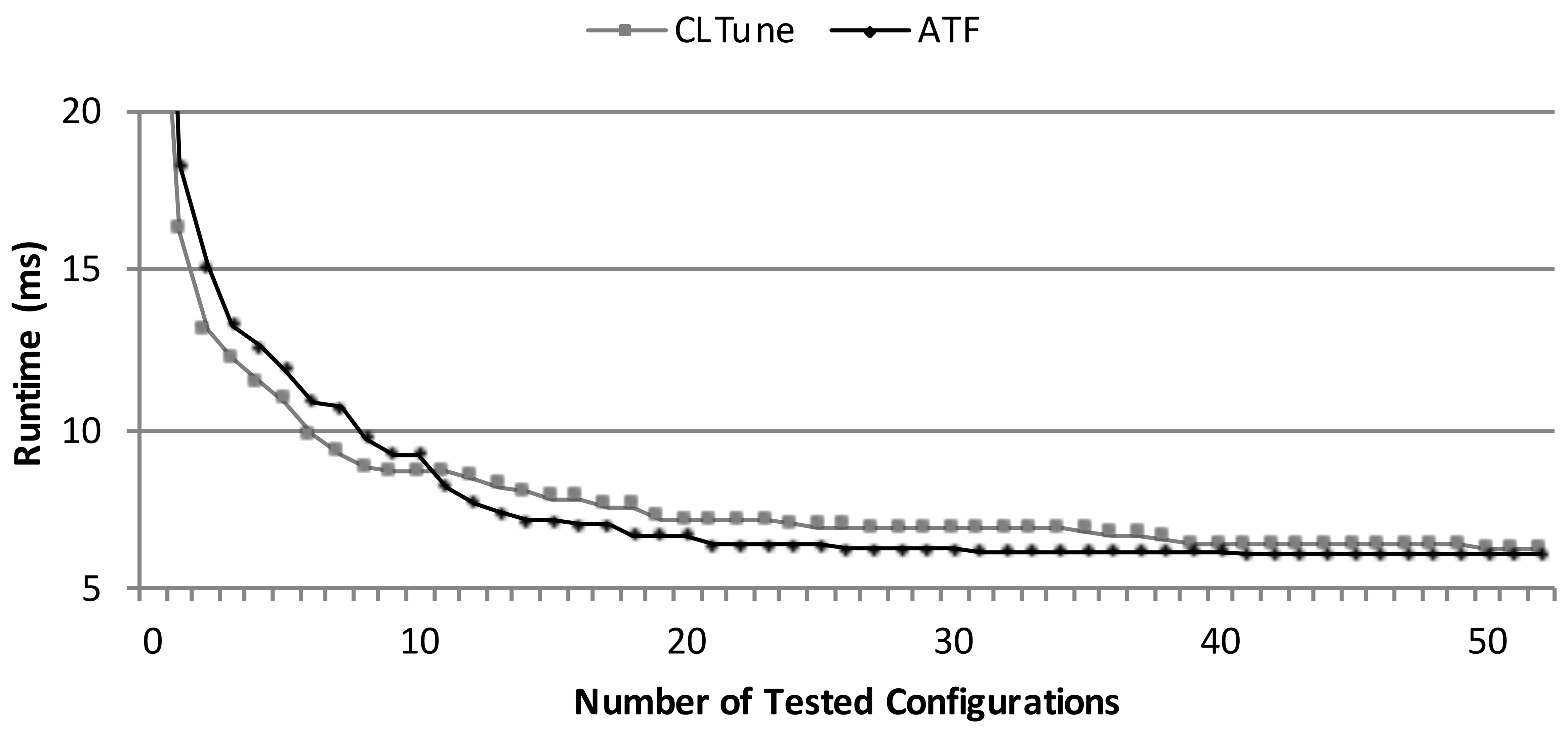
Auto-tunable implementations generated via our ***MDH code generation approach*** *[2]*

[2] Rasch, Schulze, Gorlatch. "Generating Portable High-Performance Code via Multi-Dimensional Homomorphisms", PACT'19

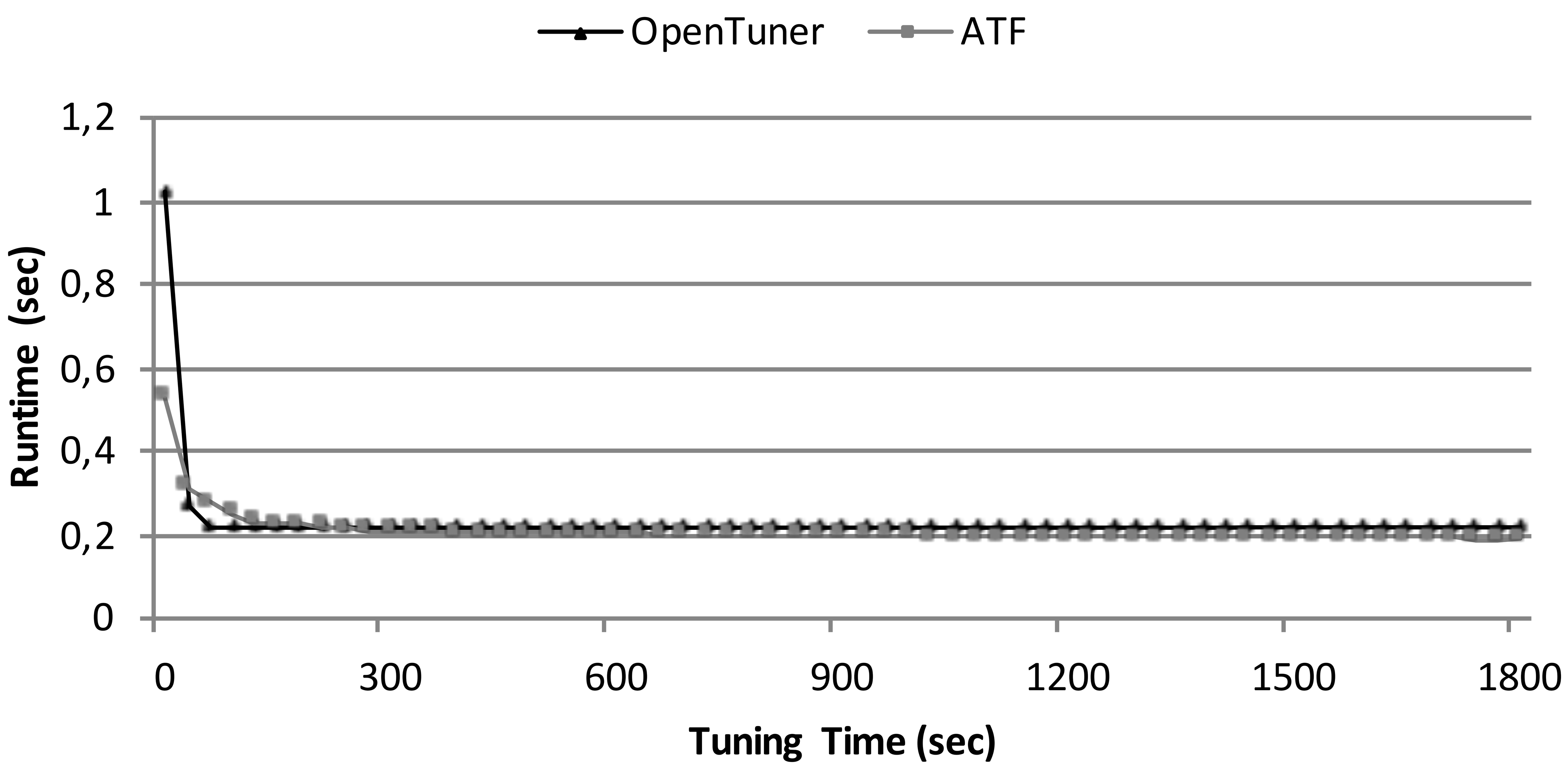
Experimental Evaluation

Highlights from CCPE'19

ATF for the favorable application classes of CLTune and OpenTuner:



2D Convolution (CLTune)



GCC Flags (OpenTuner)

ATF achieves tuning results of the same high quality as CLTune and OpenTuner for their favorable application classes.

Experimental Evaluation

ATF has also been successfully used for further application classes:

	Domains	Applications
1	Compiler Optimizations	GCC Flags [CCPE'19], SIMD Vectorization [ICS'19]
2	Data Mining	Probabilistic Record Linkage [PACT'19]
3	Quantum Chemistry	CCSD(T) [PACT'19]
4	Deep Learning	MCC [SC'21], BLAS [PACT'19]
5	DSL Compiler Optimizations	Lift [CGO'18], MDH [PACT'19]
6	Polyhedral Compilation	PPCG [CGO'18], Pluto [*WIP*]
7	Signal Processing	FFT [FHPNC'19@ICFP]
8	Stencil Computations	Conv2D, Jacobi 2D/3D, … [CGO'18]

ATF — User Interface

ATF's user interface compared to popular auto-tuning approaches:

	Domain-specific approaches	OpenTuner	CLTune	ATF
Arbitrary Programming Language		✓		✓
Arbitrary Application Domain		✓	✓	✓
Arbitrary Tuning Objective	(✓)	✓		✓
Arbitrary Search Technique	(✓)	✓	✓	✓
Interdependent Parameters	✓		✓	✓
Automatic Cost Function Generation	✓		✓	✓

ATF's user interface combines major advantages of existing auto-tuning approaches

ATF — User Interface

ATF's interface types:

Offline Auto-Tuning

```
#atf::tp name        NUM_WG_1
         range       interval<int>( 1,N_1 )

#atf::tp name        NUM_WI_1
         range       interval<int>( 1,N_1 )

#atf::tp name        LM_SIZE_1
         range       interval<int>( 1,N_1 )
         constraint  LM_SIZE_1 <= N_1

#atf::tp name        PM_SIZE_1
         range       interval<int>( 1,N_1 )
         constraint  PM_SIZE_1 <= LM_SIZE_1

#atf::run_script      "./run.sh"
#atf::compile_script "./compile.sh"

#atf::search_technique auc_bandit
#atf::search_technique duration<minutes>( 10 )

// program code
```

ATF Tuning Annotations

DSL-based Interface

Online Auto-Tuning

ATF C++ Interface

...

ATF Python Interface (WIP)

GPL-based Interfaces

ATF — User Interface

ATF's ***DSL-based*** user interface [3] (***offline tuning***):

Tuning Parameters without interdependencies

```
#atf::tp name       NUM_WG_1
         range      interval<int>( 1,N_1 )

#atf::tp name       NUM_WI_1
         range      interval<int>( 1,N_1 )
```

Name

Range (either interval or set)

Tuning Parameters with interdependencies

```
#atf::tp name       LM_SIZE_1
         range      interval<int>( 1,N_1 )
         constraint LM_SIZE_1 <= N_1

#atf::tp name       PM_SIZE_1
         range      interval<int>( 1,N_1 )
         constraint PM_SIZE_1 <= LM_SIZE_1
```

Name

Range (either interval or set)

Constraint

Cost Function specification

```
#atf::run_script     "./run.sh"
#atf::compile_script "./compile.sh"
#atf::cost_file      "./cost.txt"
```

Search Technique & Abort Condition

```
#atf::search_technique auc_bandit
#atf::search_technique duration<minutes>( 10 )

// program code
```

Program Code annotated with *tuning directives*

[3] Rasch, Gorlatch. "ATF: A Generic, Directive-Based Auto-Tuning Framework", CCPE'19

ATF — User Interface

ATF supports different *search techniques* & *abort conditions [3]:*

ATF provides pre-implemented search techniques:

1. `exhaustive`
2. `random_search`
3. `simulated_annealing`
4. `differential_evolution`
5. `particle_swarm`
6. `pattern_search`
7. `torczon`

} Basic Techniques

1. `round_robin`
2. `auc_bandit`

} Meta Techniques

ATF provides various abort conditions, e.g.:

- `duration<D>(t)`: stops tuning after time interval `t`; here, `D` is an `std::chrono::duration` (sec, min, etc.)
- `cost(c)`: stops tuning when a configuration with a cost lower or equal than `c` has been found;
- `speedup<D>(s,t)`: stops tuning when within last time interval `t` cost could not be lowered by a factor `>=s`;
- …

Further search techniques and abort conditions can be easily added to ATF

[3] Rasch, Gorlatch. "ATF: A Generic, Directive-Based Auto-Tuning Framework", CCPE'19

ATF — User Interface

ATF automatically generates the *cost function* for the user [3]:

```
#atf::run_script      "./run.sh"
#atf::compile_script  "./compile.sh"
#atf::cost_file       "./cost.txt"
```

Script for running program to tune
Script for compiling program (optional)
Cost File (optional)

```
#atf::tp name       NUM_WG_1
         range      interval<int>( 1,N_1 )

#atf::tp name       NUM_WI_1
         range      interval<int>( 1,N_1 )

#atf::tp name       LM_SIZE_1
         range      interval<int>( 1,N_1 )
         constraint LM_SIZE_1 <= N_1

#atf::tp name       PM_SIZE_1
         range      interval<int>( 1,N_1 )
         constraint PM_SIZE_1 <= LM_SIZE_1

#atf::run_script     "./run.sh"
#atf::compile_script "./compile.sh"
#atf::cost_file      "./cost.txt"

#atf::search_technique auc_bandit
#atf::search_technique duration<minutes>( 10 )

// program code
```

ATF program
(from slide 22)

ATF automatically generates the cost function for *compiling & running & measuring* the program to tune

OpenCL

```
#atf::opencl::platform_id 0
#atf::opencl::device_id   1

#atf::opencl::input scalar<int>( N )
#atf::opencl::input scalar<float>()
#atf::opencl::input buffer<float>( N )
...

#atf::opencl::global_size N/WPT
#atf::opencl::local_size LS
```

NVIDIA CUDA

```
#atf::cuda::device_id 1

#atf::cuda::input scalar<int>( N )
#atf::cuda::input scalar<float>()
#atf::cuda::input buffer<float>( N )
...

#atf::cuda::grid_dim  (N/WPT)/LS
#atf::cuda::block_dim LS
```

ATF provides convenience features for automatically generating OpenCL and CUDA host code

[3] Rasch, Gorlatch. "ATF: A Generic, Directive-Based Auto-Tuning Framework", CCPE'19

ATF — User Interface

Side note — observation:

```
global_size ( ((1 + ((kSizeM - 1) / MWG))*MWG * MDIMCD) / MWG ,
              ((1 + ((kSizeN - 1) / NWG))*NWG * NDIMCD) / NWG )
local_size  ( MDIMCD , NDIMCD )
```

ATF: Global & Local size

ATF more expressive for OpenCL than CLTune

Exactly as originally intended for CLTune's GEMM in [Matsumoto et al., 2014]

```
// Default values for global & local size
auto id = tuner.AddKernel(gemm_fast, "gemm_fast", {kSizeM, kSizeN}, {1, 1});

// ...

// Sets constraints
tuner.AddConstraint(id, DividesM, {"MWG"}); // MWG has to divide M
tuner.AddConstraint(id, DividesN, {"NWG"}); // NWG has to divide N

// ...

// Modifies the thread-sizes (both global and local) based on the parameters
tuner.MulLocalSize(id, {"MDIMC", "NDIMC"});
tuner.MulGlobalSize(id, {"MDIMC", "NDIMC"});
tuner.DivGlobalSize(id, {"MWG", "NWG"});
```

workaround (not required in ATF)

CLTune: Global & Local size

ATF is able to express exactly CLTune GEMM's global & local sizes, thereby avoiding additional constraints (speedups >1,37x for DL sizes)

ATF — User Interface

ATF's ***DSL-based*** user interface [3] (*offline tuning*):

```
{
  "tuning_parameters" :
  [
    {
      "name"   : "NUM_WG_1",
      "range" : "interval<int>( 1,1000 )"
    },
    {
      "name"   : "NUM_WI_1",
      "range" : "interval<int>( 1,1000 )"
    }
    ,
    {
      "name"   : "LM_SIZE_1",
      "range" : "interval<int>( 1,1000 )",
      "constraint" : "LM_SIZE_1 <= N_1"
    },
    {
      "name"   : "PM_SIZE_1",
      "range" : "interval<int>( 1,1000 )",
      "constraint" : "PM_SIZE_1 <= LM_SIZE_1"
    }
  ],

  "program_source" : "./program.cl",
  "compile_script" : "./compile_script.sh",
  "run_script"     : "./run.bin",
  "cost_file"      : "./cost.txt",

  "search_technique" : "auc_bandit",
  "abort_condition"  : "duration<minutes>(30)",
}
```

Tuning Parameters without interdependencies

Tuning Parameters with interdependencies

Cost Function specification

Search Technique & *Abort Condition*

Name

Range (either `interval` or `set`)

Constraint

JSON file format as alternative to directives

[3] Rasch, Gorlatch. "ATF: A Generic, Directive-Based Auto-Tuning Framework", CCPE'19

ATF — User Interface

ATF's ***GPL-based*** user interface [4] (***online tuning***), for **C++**:

```
int main()
{
  const std::string saxpy = /* path to kernel of Listing */;
  const int         N     = /* an arbitrary input size   */;

  // Step 1: Generate the Search Space
  auto WPT = atf::tp( "WPT"                         ,
                      atf::interval<size_t>( 1,N ) ,
                      atf::divides( N )            );
  auto LS  = atf::tp( "LS"                          ,
                      atf::interval<size_t>( 1,N ) ,
                      atf::divides( N/WPT )        );

  // Step 2: Implement a Cost Function
  auto saxpy_kernel = atf::opencl::kernel< atf::scalar<int>   ,  // N
                                           atf::scalar<float> ,  // a
                                           atf::buffer<float> ,  // x
                                           atf::buffer<float> >  // y
                                          ( saxpy_kernel_as_string, "saxpy" );  // kernel code & name

  auto cf_saxpy = atf::opencl::cost_function( saxpy_kernel ).platform_id( 0 )                  // OpenCL platform id
                                                            .device_id( 0 )                    // OpenCL device id
                                                            .inputs( atf::scalar<int>( N )   , // N
                                                                     atf::scalar<float>()    , // a
                                                                     atf::buffer<float>( N ) , // x
                                                                     atf::buffer<float>( N ) ) // y
                                                            .global_size( N/WPT )              // OpenCL global size
                                                            .local_size( LS )                  // OpenCL local size

  // Step 3: Explore the Search Space
  auto tuning_result = atf::tuner().tuning_parameters( WPT,LS )
                                   .search_technique( atf::auc_bandit() )
                                   .tune( cf_saxpy , atf::evaluations(50) );
}
```

Analogous to ATF's DSL-based interface

[4] Rasch, Haidl, Gorlatch. "ATF: A Generic Auto-Tuning Framework", HPCC'17

ATF — User Interface

ATF's ***GPL-based*** user interface (***online tuning***), for **Python**:

```python
import atf

# kernel code as string
saxpy_kernel_as_string = """
__kernel void saxpy( const int N, const float a, const __global float* x, __global float* y )
{
    for( int w = 0 ; w < WPT ; ++w )
    {
        const int index = w * get_global_size(0) + get_global_id(0);
        y[ index ] += a * x[ index ];
    }
}
"""

# input size
N = 1000

# Step 1: Generate the Search Space
WPT = atf.tp( "WPT", atf.interval( atf.size_t , 1,N ), atf.divides( N )     )
LS  = atf.tp( "LS" , atf.interval( atf.size_t , 1,N ), atf.divides( N/WPT ) )

# Step 2: Implement a Cost Function
saxpy_kernel = atf.opencl.kernel( types=[ atf.scalar(atf.int)   ,     # N
                                          atf.scalar(atf.float) ,     # a
                                          atf.buffer(atf.float) ,     # x
                                          atf.buffer(atf.float) ],    # y
                                  kernel_code=saxpy_kernel_as_string,
                                  kernel_name="saxpy" )

cf_saxpy = atf.opencl.cost_function(
    kernel=saxpy_kernel,
    platform_id=0,                            # OpenCL platform id
    device_id=0,                              # OpenCL device id
    inputs=[ atf.scalar( atf.int , N )    ,   # N
             atf.scalar( atf.float )      ,   # a
             atf.buffer( atf.float , N ) ,    # x
             atf.buffer( atf.float , N ) ],   # y
    global_size=N/WPT,                        # OpenCL global size
    local_size=LS                             # OpenCL local size
)

# Step 3: Explore the Search Space
tuning_result = atf.tuner(
    tuning_parameters=[ WPT, LS ],
    search_technique=atf.auc_bandit,
).tune( cf_saxpy, atf.evaluations(50) )
```

Analogous to ATF's DSL-based interface

Work-in-Progress results from Bachelor Thesis:
"Migration of Auto-Tuning Framework from C++ to Python",
Waldemar Gorous, 2019

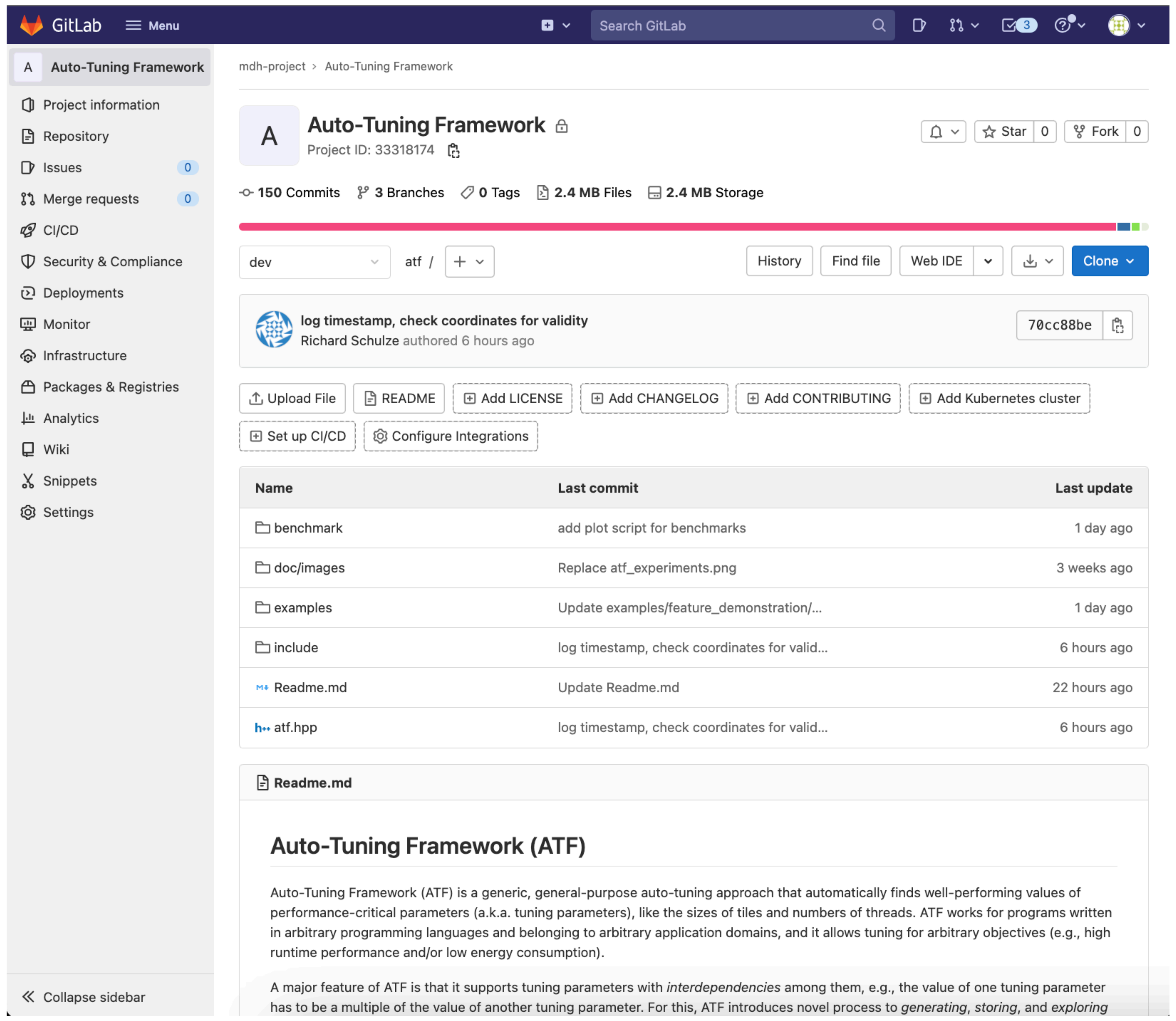
Summary

<u>ATF's **main contribution** is efficiently handling **tuning parameters with interdependencies** among them:</u>

- ATF introduces novel processes to <u>*generating*</u> & <u>*storing*</u> & <u>*exploring*</u> the search spaces of interdependent parameters, based on its *parameter constraints* and the *chain-of-trees search space representation*;

- ATF's user interface is <u>arguably simpler to use</u> than CLTune & OpenTuner [CCPE'19, HPCC'17].

- *ATF does not contribution to search techniques (in contrast to: **KT**, **KTT**, **HyperMapper — next talk**, …)*



<u>ATF available on GitLab for all workshop attendees:</u>

`https://gitlab.com/mdh-project/atf`

ATF — Current State & Future Work

Current State:

- ATF's DSL-based user interface currently not maintained (we use its C++ interface)
- ATF's implementation is a proof of concept (error handling can be improved, etc)
- …

Future Work:

- getting the best of both worlds:
  1. *efficient search techniques,* as in **KT**, **KTT**, **HyperMapper**, etc;
  2. *efficient search space generation & storing & exploring,* as in **ATF**.
- using different search techniques for search space parts with different characteristics (as in Pfaffe et. al [ICS'19], **HyperMapper**, etc): *categorical*, *ordinal*, *real*, etc.
- improving multi-objective auto-tuning toward *Pareto optimality* (as in **HyperMapper**)
- …

Thanks for listening!

Richard Schulze
r.schulze@wwu.de

Ari Rasch
a.rasch@wwu.de

Questions?