C Configuration Space and Tuning Library (CCS)

Brice Videau

Argonne National Laboratory
Section 1

Context
Auto-Tuning as a Service

BOAST: Metaprogramming and Tuning Framework (2011):

- Computing kernel oriented
- Users could define parametrized code variants
- BOAST could
  - Generate a kernel variant based on a given configuration;
  - Build and link the kernel into a plugin;
  - Load the plugin and evaluate it into BOAST memory space.
- Defined an API to plug autotuning frameworks:
  - Autotuning as a service;
  - BOAST asks for a configuration, and give results back;
  - Repeat until satisfied/finished.

Very performant performance evaluation/validation, as kernel calls can reuse input data, and noise is kept minimal.
Problematic

- At the time no auto-tuning framework offered such an interface.
- Fast forward 10 years, no auto-tuning framework offers such an interface:
  - This is a lie, some python frameworks offer an ask and tell interface;
  - Auto-tuning frameworks consider themselves orchestrator of the process.
- How can performant online auto-tuning be enabled?
  - JIT, compiler code generation autotuning;
  - Performance critical, autotuning driven by the application;

Create a low level API to describe autotuning problems and their associated tuners.
Section 2

C Configuration Space and Tuning Library
Objective

Goal

- Create a dedicated low level API and library to:
  - Describe hyperparameters, configuration spaces, and tuners with ask and tell interface;
  - Provide expected functionalities: constrained values, sampling, etc…
  - Bridge autotuning frameworks and applications with auto-tuning needs.
- Demonstrate capabilities

Constraints

- Performant: limit undue overheads
- Easy to bind in high level languages
- High level languages tuners must be able to be invoked from C/C++
- Co-design with PROTEAS-TUNE ECP project and Kokkos tools
- Drop in replacement for ConfigSpace (Python AutoML package https://automl.github.io/ConfigSpace/master/)
Design Philosophy

- Greatly inspired by ConfigSpace
- C API, with reference counted objects
- Keep objects immutable when possible to limit the need for synchronization
- Limit memory allocation to what is strictly necessary internally, users should manage their memory
- Decouple statistical aspects from problem descriptions
- Flexibility when it doesn’t harm performance
State of the Project

Nearing CCS first release:

**CCS supports**
- Numerical, Categorical, and Ordinal **hyperparameters**;
- **Random number generators** (GLS based);
- Uniform, Normal, Roulette, Multivariate, and Mixture **distributions**;
- **Configuration spaces** with constraints (inactive hyperparameters and forbidden clauses) and sampling facilities;
- **Objective spaces** with a rich expression based objective description;
- **Expressions** to describe constraints and objectives;
- **Tuners** with ask and tell interface to optimize configuration spaces.

**Interoperability**
- Python and Ruby bindings, with python and ruby tuners demonstrated running from C;
- Integrated into ytopt and DeepHyper tools;
- Kokkos connector for autotuning of Kokkos regions.
Example

```python
import cconfigspace as ccs
p1 = ccs.CategoricalHyperparameter(name = 'p1', values = [' ', ' #pragma omp #P2', '#pragma omp target teams distribute #P2', '#pragma omp target teams distribute #P4', '#pragma omp #P3'])
p2 = ccs.CategoricalHyperparameter(name = 'p2', values = [' ', 'parallel for #P3', 'parallel for #P5', 'parallel for #P6'])
p3 = ccs.CategoricalHyperparameter(name = 'p3', values = [' ', 'simd'])
p4 = ccs.CategoricalHyperparameter(name = 'p4', values = [' ', 'dist_schedule(static)', 'dist_schedule(static, #P8)'])
p5 = ccs.CategoricalHyperparameter(name = 'p5', values = [' ', 'schedule(#P7,#P8)', 'schedule(#P7)'])
p6 = ccs.CategoricalHyperparameter(name = 'p6', values = [' ', 'numthreads(#P9)'])
p7 = ccs.OrdinalHyperparameter(name = 'p7', values = ['static', 'dynamic'])
p8 = ccs.OrdinalHyperparameter(name = 'p8', values = ['1', '8', '16'])
p9 = ccs.OrdinalHyperparameter(name = 'p9', values = ['1', '8', '16'])

cs = ccs.ConfigurationSpace(name = "omp")
cs.add_hyperparameters([p1, p2, p3, p4, p5, p6, p7, p8, p9])

cs.set_condition(p2, "p1 # [' #pragma omp #P2', '#pragma omp target teams distribute #P2']")
cs.set_condition(p4, "p1 == '#pragma omp target teams distribute #P4'")
cs.set_condition(p3, "p1 == '#pragma omp #P3' || p2 == 'parallel for #P3'")
cs.set_condition(p5, "p2 == 'parallel for #P5'")
cs.set_condition(p6, "p2 == 'parallel for #P6'")
cs.set_condition(p7, "p5 # ['schedule(#P7)', 'schedule(#P7,#P8)']")
cs.set_condition(p8, "p4 == 'dist_schedule(static, #P8)' || p5 == 'schedule(#P7,#P8)'")
cs.set_condition(p9, "p6 == 'numthreads(#P9)'")

cs.add_forbidden_clauses(['p1 == '#pragma omp #P2' && p2 == ', ', 'p1 == '#pragma omp #P3' && p3 == ', '])
samples = cs.samples(100)
```
Section 3

Conclusion and Future Work
Conclusion and Future Work

CCS is nearing it’s first release:

- Find it on GitHub: https://github.com/argonne-lcf/CCS
- Already largely usable, easy prototyping in high level languages
- First release will be packaged in spack

Future works:

- Serialization of tuners / configuration space
- Tree generative configuration spaces

Collaboration opportunities:

- CCS is a co-design effort: still time to stir development to suit your use-case
- Integrate CCS in runtimes to optimize tunables
Acknowledgement

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.