ML-based Performance Portability for Time-Dependent Density Functional Theory in HPC Environments

Adrián P. Diéguez, Min Choi, Xinran Zhu, Bryan M. Wong and Khaled Z. Ibrahim

PMBS'22, held in conjunction with **SC’22**
Nov. 14, 2022

SC22
Dallas, TX | hpc accelerates.

Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Contributions

- ML methodology for performance portability
  - Transfer learning based on Bayesian optimization
  - Up to **46%** faster than conventional Bayesian optimization, up to **86%** faster than exhaustive search
  - Tested on a TDDFT workload, but with **broader applicability**
- DFTuning: a workflow for DFT performance portability
- Correlation metric for assessing the quality of Transfer Learning

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

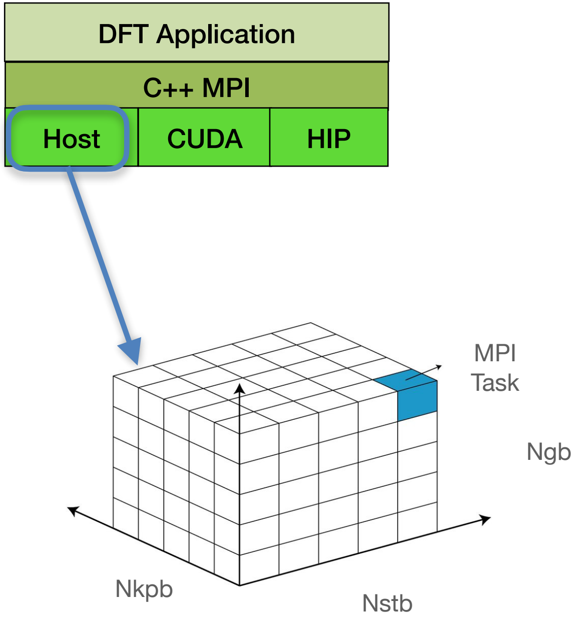
Motivation

- Density Functional Theory: a workhorse of chemistry and materials science.
- Objective: Target to new generations of DOE exascale machines.
  Performance portability challenge.

- The challenge is not new, but on the exascale era it is imperative to reduce the number of evaluations during the search.

Application Motivation

One application …



… different portability scenarios.

- *One Node of Cori (MPI)*
- *One Node of Perlmutter (MPI, CUDA)*
- *Multiple Nodes of Perlmutter (MPI, CUDA)*
- *One Node of Frontier (MPI, HIP)*

Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Autotuning

Auto-tuning can help with this:

- Empirical search
- Predictive search

Empirical_Search (shapes, strides):

$a \leftarrow TaskFeatures(shapes, strides)$
$c \leftarrow PlatformFeatures()$
$b^* \leftarrow \mathrm{argmin}_{b \in \mathcal{B}}\, MeasureTime(a, b, c)$
return b*

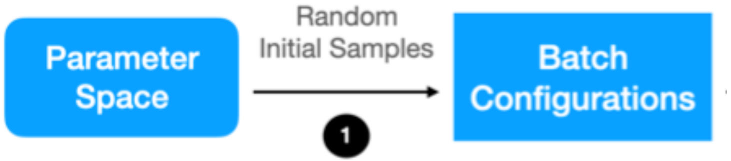
Predictive_Search (shapes, strides):

$a \leftarrow TaskFeatures(shapes, strides)$
$c \leftarrow PlatformFeatures()$
$f \leftarrow TimingModel()$
$b^* \leftarrow \mathrm{argmin}_{b \in \mathcal{B}}\, f(a, b, c)$
return b*

Empirical Search		Guarantees finding optimal	Very slow
Predictive Search	*Analytical Model*	Reduces the search time.	• Results depends on the quality of model • Complex
	Machine Learning	Reduces the search time. Black Box.	The search process is still infeasible

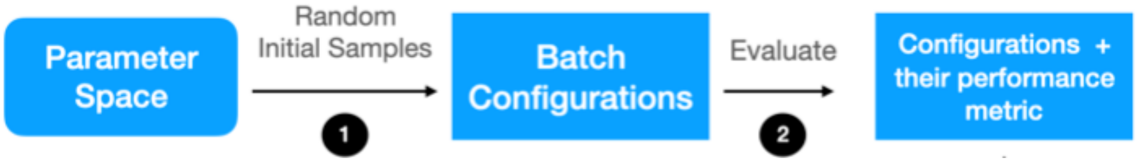
$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Bayesian optimization

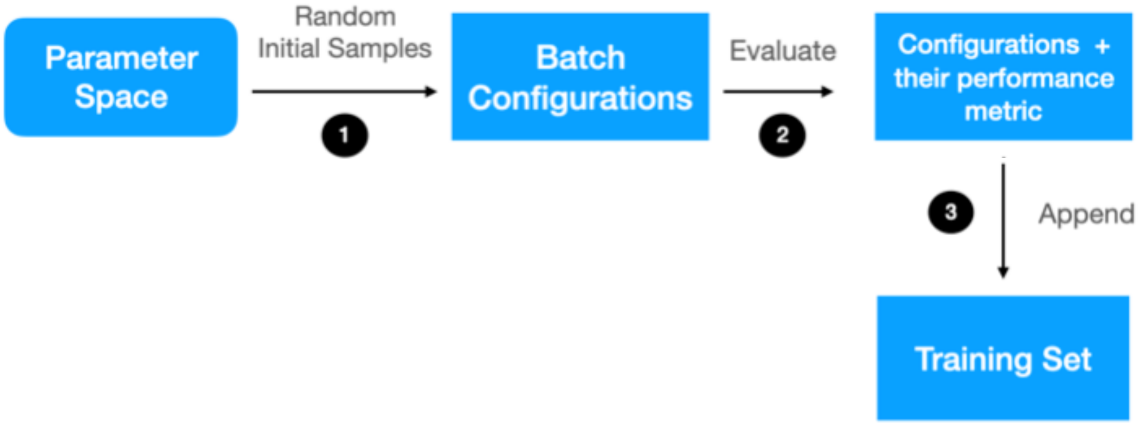


$$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$$

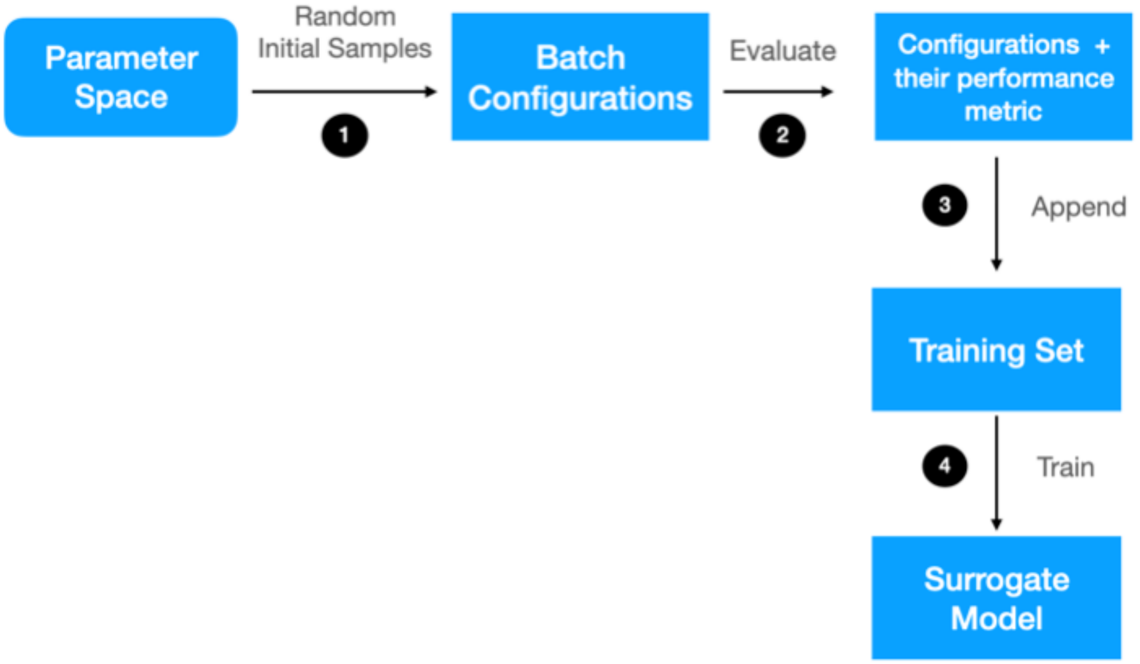
Bayesian optimization



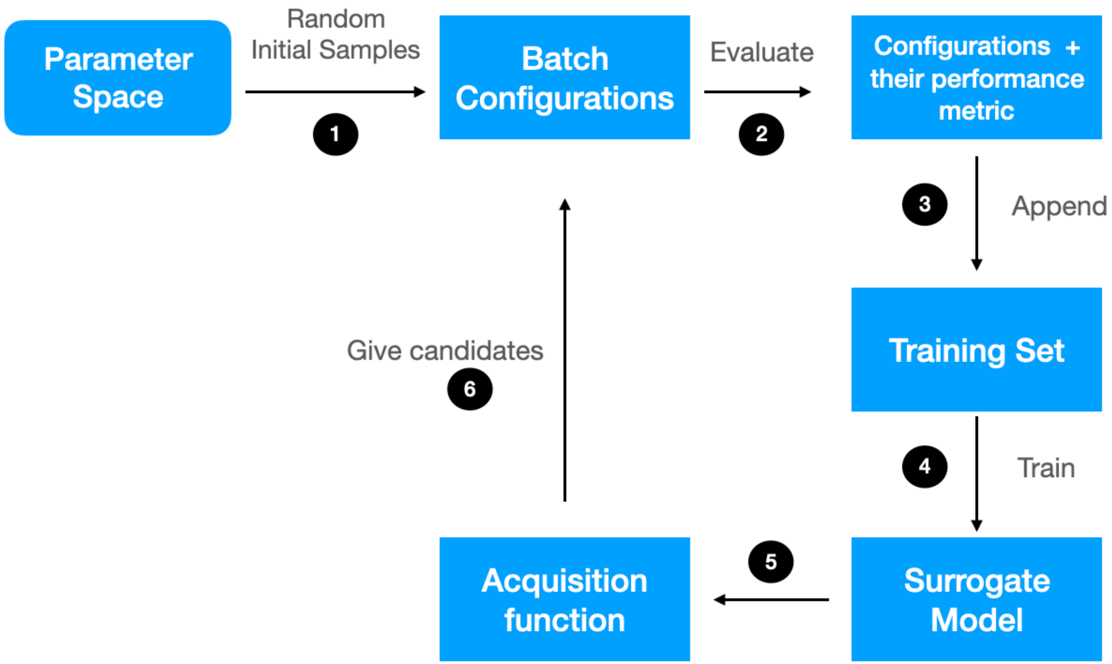
Bayesian optimization



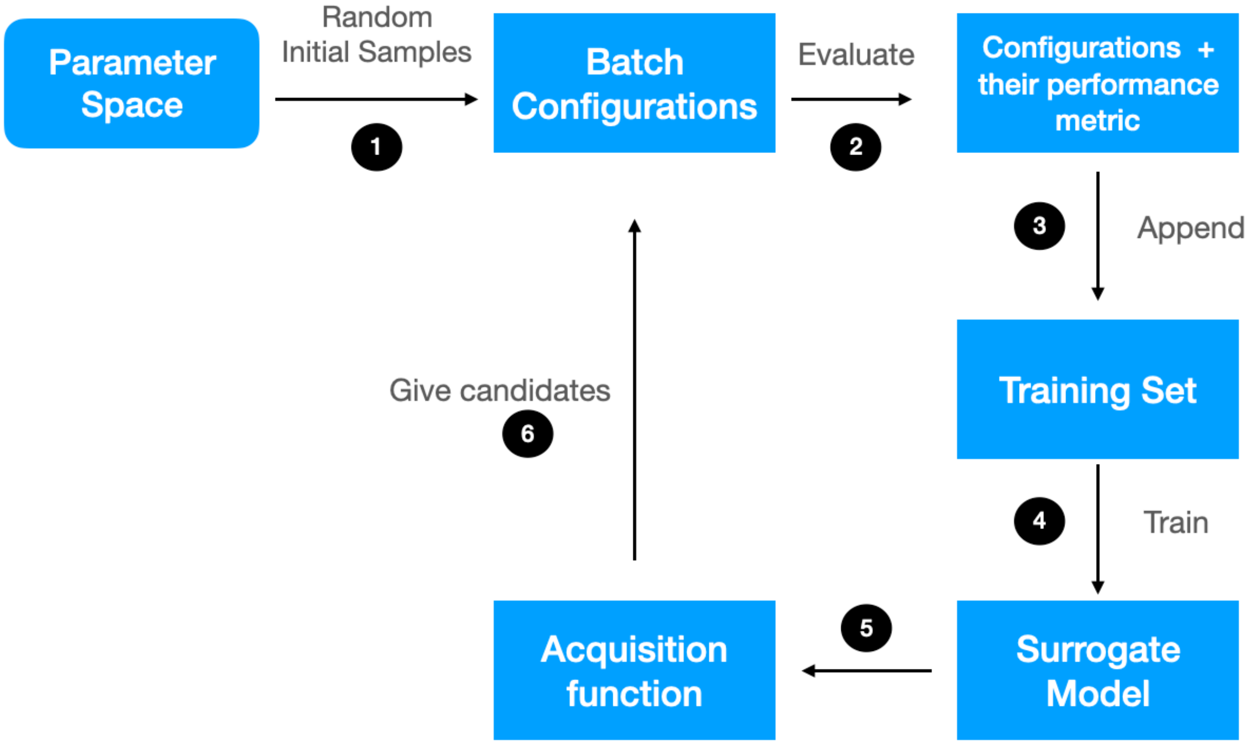
Bayesian optimization

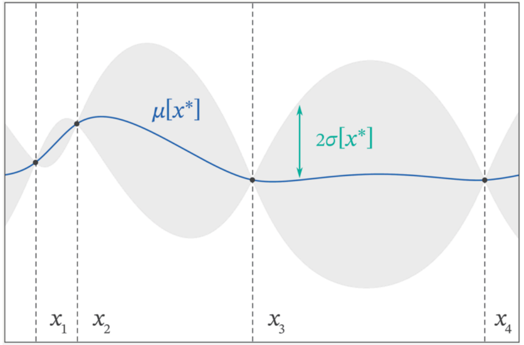


Bayesian optimization

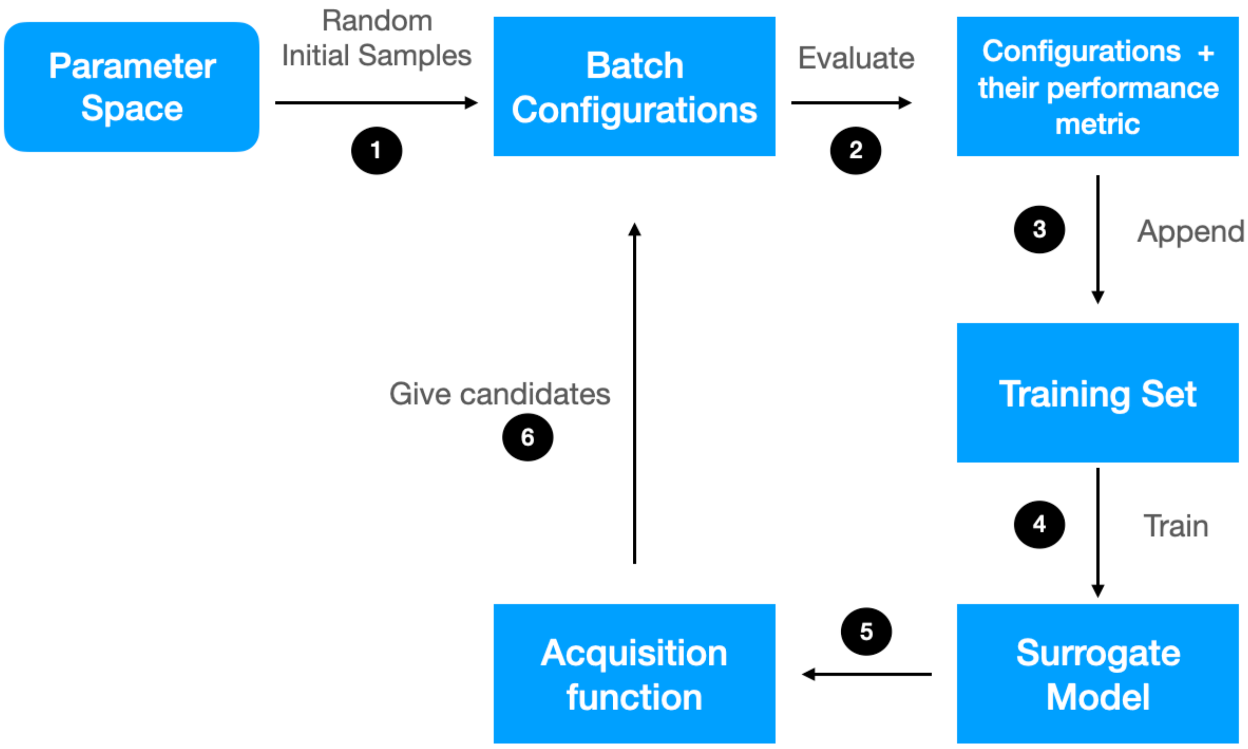


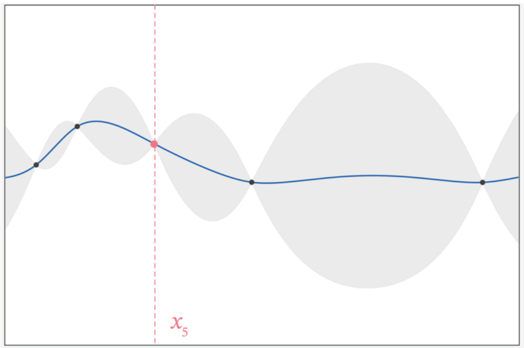
Bayesian optimization



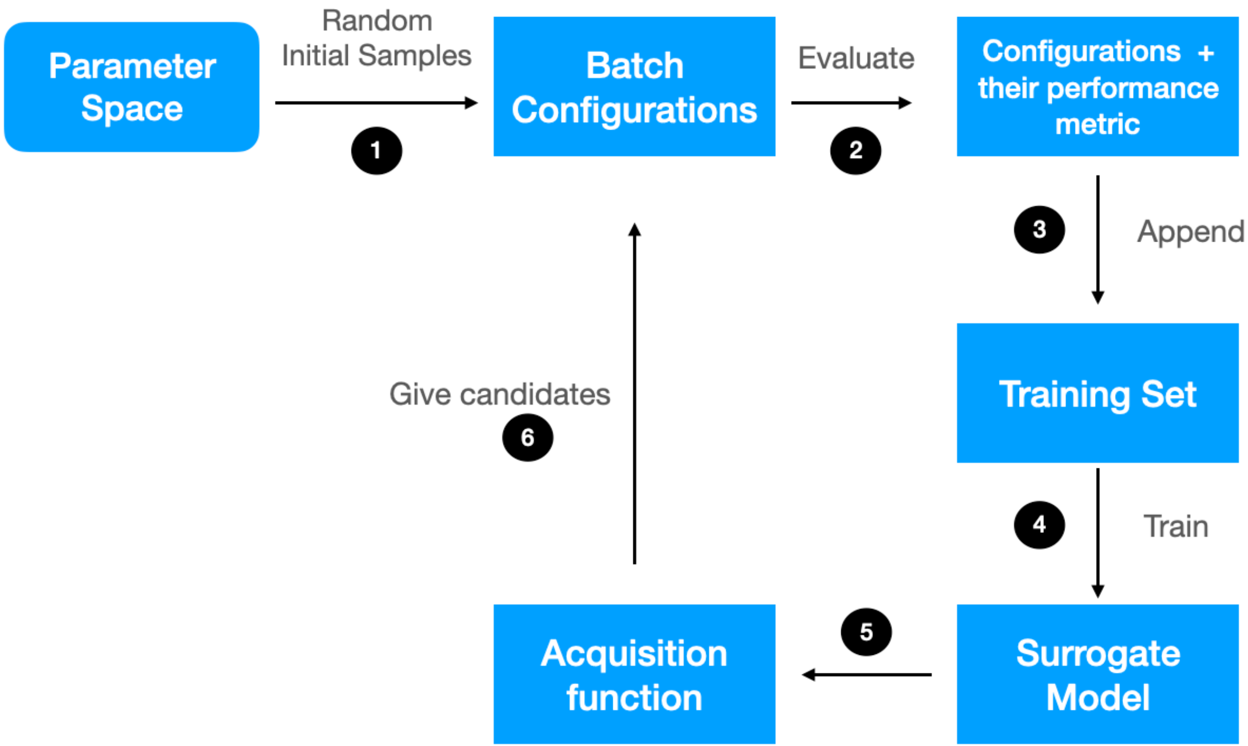


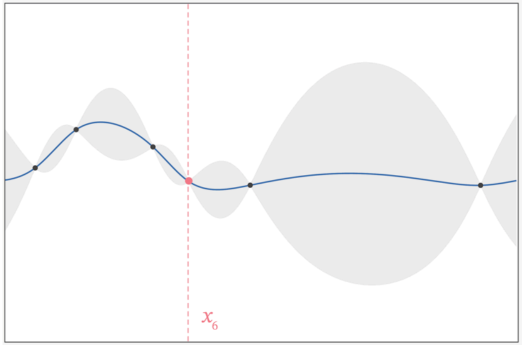
Bayesian optimization



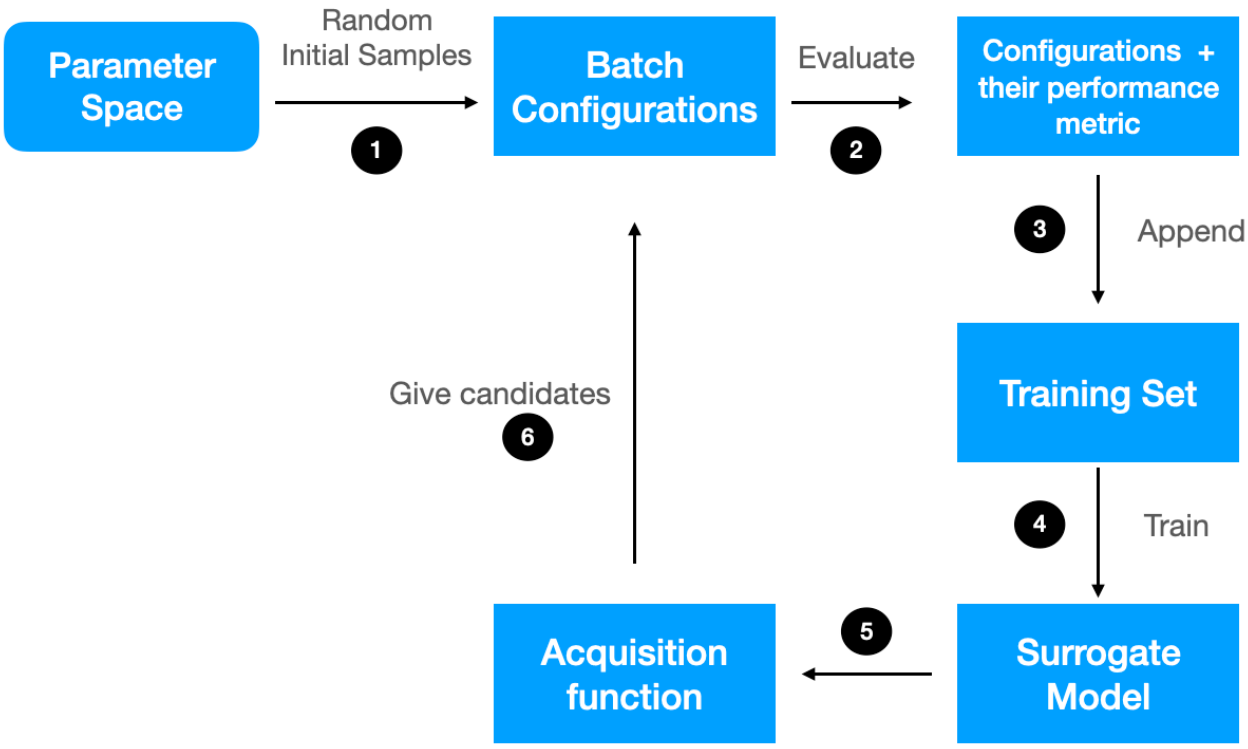


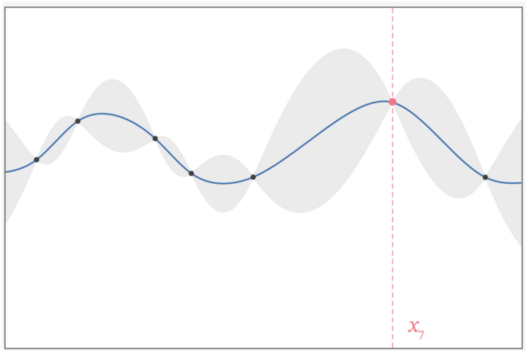
Bayesian optimization





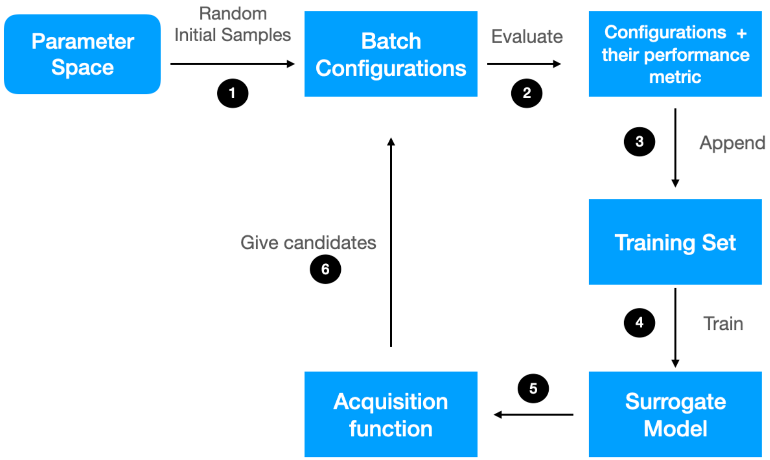
Bayesian optimization





Transfer Learning (I)

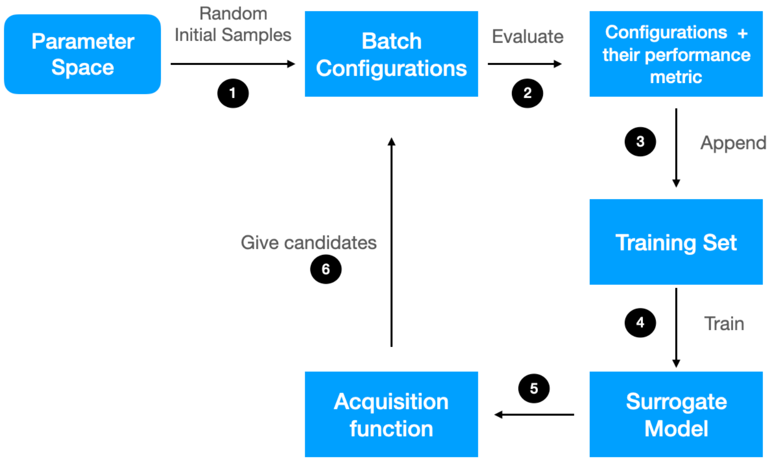
Running the search on Cori:



63 sample evaluations

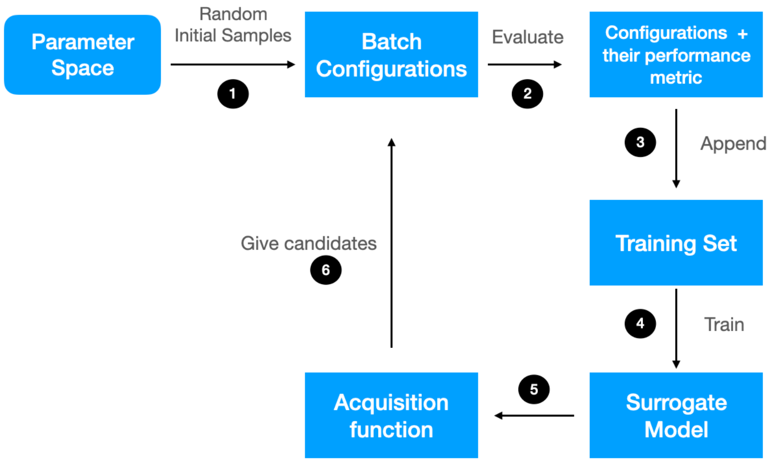
Transfer Learning (II)

Running the search on Cori:



63 sample evaluations

Running the search on Perlmutter:

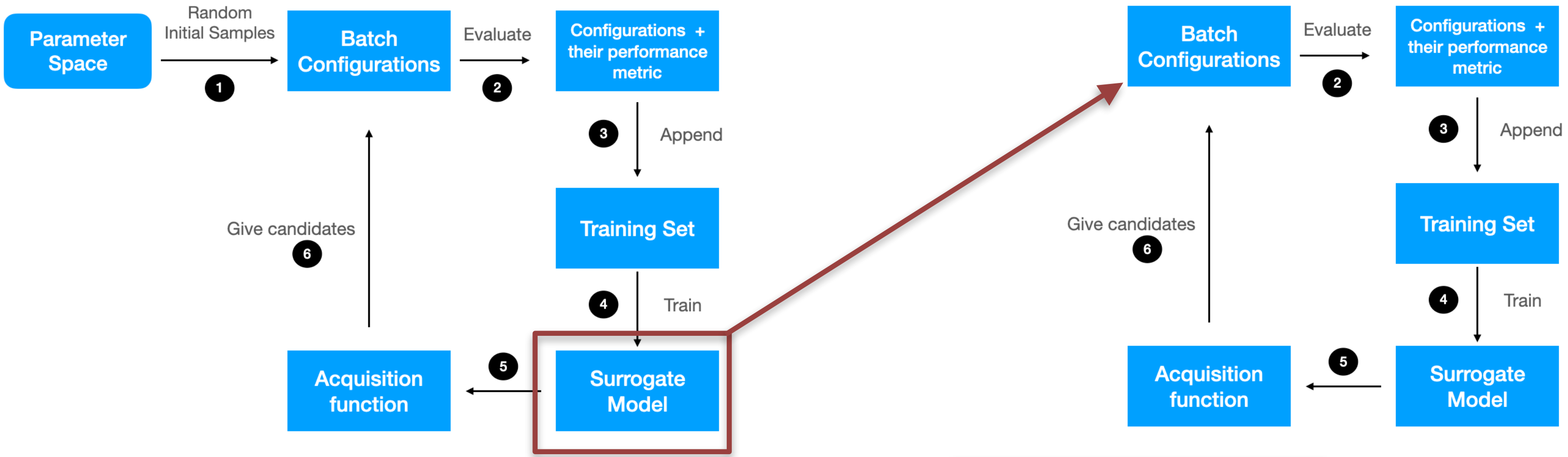


71 sample evaluations

Transfer Learning (and III)

Running the search on Cori:

Running the search on Perlmutter:



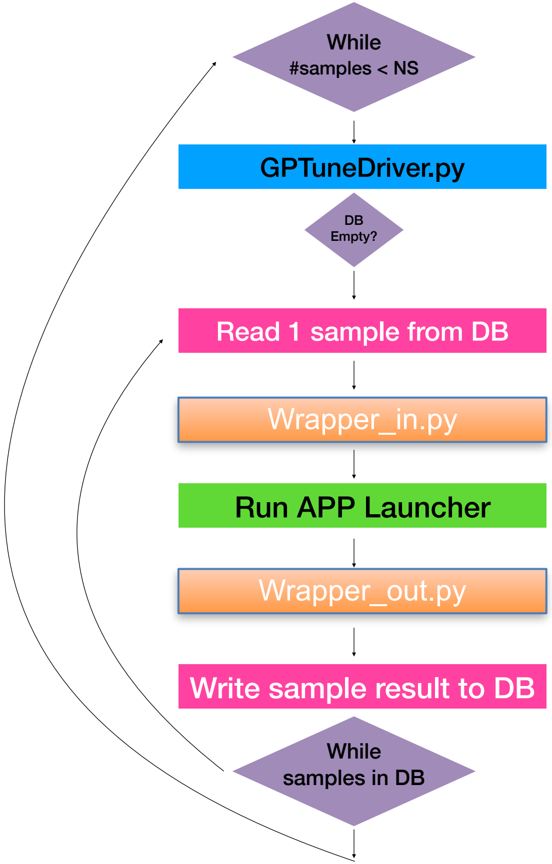
63 sample evaluations

30 sample evaluations

Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

DFTuning: Decoupling workflow from GPTune



1- Portability Support:

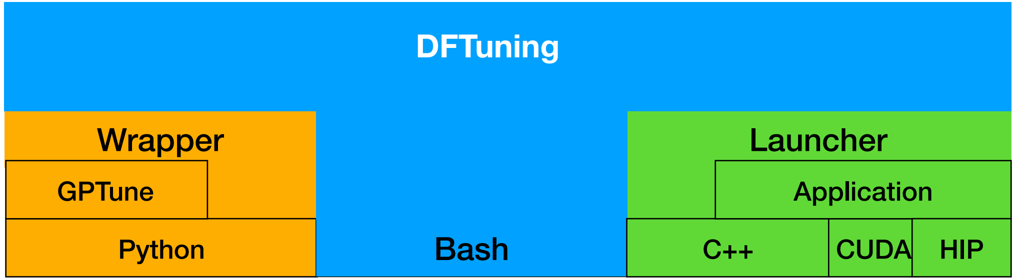
- Supports Transfer Learning for learning tuning parameters for a new input **on the same platform**

2- Convergence:

- Insufficient converge criteria

3- Search Efficiency:

- Initial samples evaluated sequentially
- Acquisition function provides 1 candidate at a time
- MPI spawning based on OpenMPI



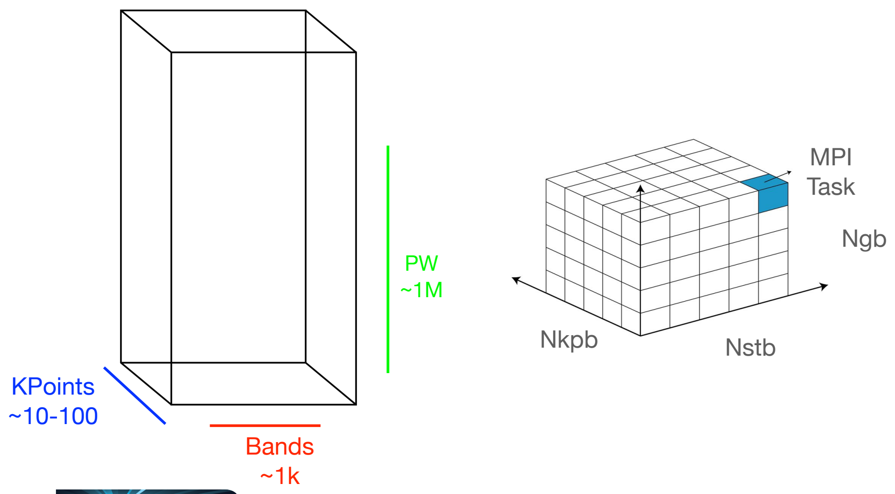
Outline

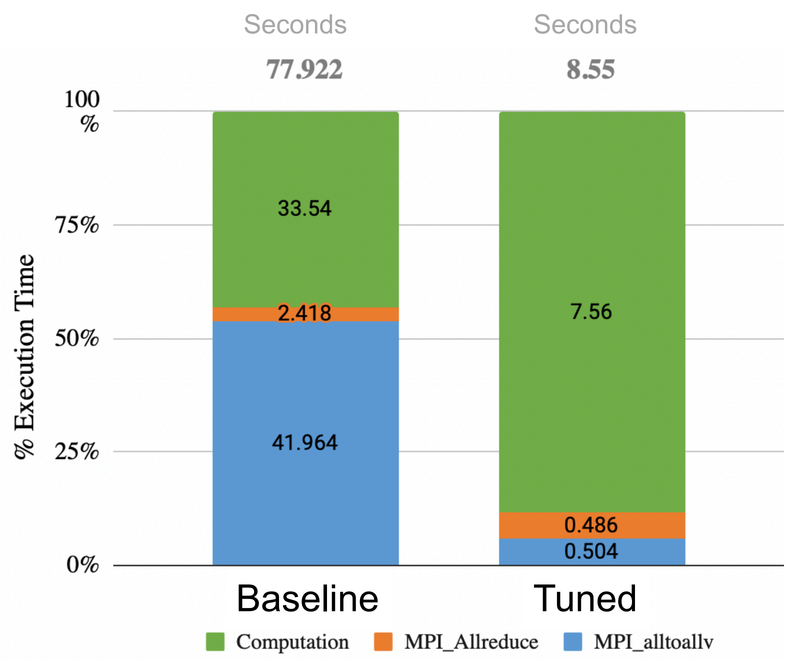
- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

RT-TDDFT MiniApp Analysis

- RT-TDDFT MiniApp using QBox framework
- Tuning parameters define the MPI grid dimensionality
- Wide range of exec. times depending on tuning parameters
- Communication bounded





Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

ML Methodology

- **Task/Input description** (argument shapes, data layout)
- **Implementation description** (auto-tuning parameters)
- **Platform description** (capabilities, micro-benchmarks)

Given an *input task* and a *platform*: find an optimal *tuning configuration*

Predictive_Search (shapes, strides):

$a \leftarrow TaskFeatures(shapes, strides)$

$c \leftarrow PlatformFeatures()$

$\boxed{\begin{array}{l} f \leftarrow TimingModel() \\ b^* \leftarrow \operatorname{argmin}_{b \in \mathcal{B}} f(a, b, c) \end{array}}$

return b*

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

ML Methodology

- **Task/Input description** (argument shapes, data layout)
- **Implementation description** (auto-tuning parameters)
- **Platform description** (capabilities, micro-benchmarks)

Given an *input task* and a *platform*: find an optimal *tuning configuration*

Input Parameters

C	The name of the input task. In this case 'Si_222'

Task Description

Performance Parameters

sp	Number of ranks working on Spin dimension. It can be 1 or 2
kp	Number of ranks on the KPoint dimension. Any power-of-2 number from 1 to cores*nodes
sb	Number of ranks on the Band dimension. Any power-of-2 number from 1 to cores*nodes
$ranks$	Total number of ranks to be used. Any power-of-2 number from 2 to cores*nodes

Implementation Description

Constants

$cores$	Number of cores per node in the target platform, which is a power-of-two number.
$nodes$	Number of allowed nodes to use in the target platform.

Platform Description
and they are **constants**.

Constraints

Constraint1	$ranks \leq cores \times nodes$
Constraint2	$sp \times kb \times sb \leq ranks$

BERKELEY LAB
Bringing Science Solutions to the World
U.S. DEPARTMENT OF ENERGY
Office of Science

ML Methodology

- **Task/Input description** (argument shapes, data layout)
- **Implementation description** (auto-tuning parameters)
- **Platform description** (capabilities, micro-benchmarks)

Given an *input task* and a *platform*: find an optimal *tuning configuration*

What if a new *input task* on the same platform?

Run a new search from scratch

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

ML Methodology

- **Task/Input description** (argument shapes, data layout)
- **Implementation description** (auto-tuning parameters)
- **Platform description** (capabilities, micro-benchmarks)

Given an *input task* and a *platform*: find an optimal *tuning configuration*

Knowledge from the *source* task is shared during the learning of the *new* task

What if a new *input task* on the <u>same</u> platform?

Run a new search from scratch
- or -
Using the Transfer Learning Autotuning feature of GPTune

If *new* and *source* tasks are similar,
the model will learn correlations among them,
<u>accelerating the search or</u>
<u>finding better optimal values</u>.

ML Methodology

- **Task/Input description** (argument shapes, data layout)
- **Implementation description** (auto-tuning parameters)
- **Platform description** (capabilities, micro-benchmarks)

Given an *input task* and a *platform*: find an optimal *tuning configuration*

But **this is not performance portability** between platforms.

What if a new *input task* on the same platform?

Run a new search from scratch

- or -

Using the Transfer Learning Autotuning feature of GPTune

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Using BO+TL search for performance portability

Platform Description

Task Description

Implementation Description

Input Parameters	
bandwidth	Peak theoretical bandwidth of the target platform in GB/s. From 1 to 1000.
cores	Number of cores per node in the target platform, which is a power-of-two number. From 1 to 256.
nodes	Number of allowed nodes to use in the target platform. From 1 to 3.
C	The name of the input task. In this case 'Si_222'
Performance Parameters	
sp	Number of ranks working on Spin dimension. It can be 1 or 2
kp	Number of ranks on the KPoint dimension. Any power-of-2 number from 1 to cores*nodes
sb	Number of ranks on the Band dimension. Any power-of-2 number from 1 to cores*nodes
$ranks$	Total number of ranks to be used. Any power-of-2 number from 2 to cores*nodes
Constraints	
Constraint1	$ranks \leq cores \times nodes$
Constraint2	$sp \times kb \times sb \leq ranks$

- Embedding the Platform description parameters as Input Parameter variables (not constants anymore).
- Modifying the metadata to enable the execution.
- Choosing Platform features that can explain the objective function: communication-bounded

Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Test Platforms at NERSC

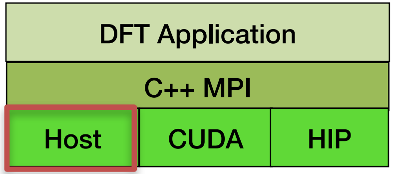
	Cori	Perlmutter
CPU	2x Intel Xeon E5-2698 v3 2.3 GHz	AMD EPYC 7763 2450 GHz
# cores per Node	32	64
Node Memory	128 GB DDR4 2133 MHz	256 GB DDR4 3200 MHz
Node Peak Mem. Bandwidth	$<$ 110 GB/s	204.8 GB/s
Interconnection	Cray Aries DragonFly	HPE Cray Slingshot 11

- *Si_222* input task: 8 k-points, 4 bands, 40K PWs.

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Portability scenarios

One application …



… different portability scenarios.

- One Node of Cori (MPI)
- One Node of Perlmutter (MPI) ← Cross-Platform
- Multiple Nodes of Perlmutter (MPI) ← Intra-Platform

Transfer-Learning Results

(*sp, kp, sb, ranks*)

Scenario	*Found at*	*Execut. Eval.*	*Optimal Config.*	*Eval. Time*
BO Perlmutter, 1 node	#47	70	$(1, 8, 4, 64)$	1.36s

An exhaustive search would explore 204 valid combinations.

A reduction of **66%** in the number of evaluations.

$\mathrm{cov}(u_i(z), u_j(z')) = k_j(z, z')$

Transfer-Learning Results: Intra-Platform

(*sp, kp, sb, ranks*)

Scenario	*Found at*	*Execut. Eval.*	*Optimal Config.*	*Eval. Time*
BO Perlmutter, 1 node	#47	70	$(1, 8, 4, 64)$	1.36s
Intra-Platform Portability				
Without TL	#43	70	$(1, 8, 2, 128)$	1.12s
With TL	#40	60	$(1, 8, 4, 128)$	0.91s

Now using 2 Nodes of Perlmutter.

Exhaustive search would explore 285 combinations

From 70 to 60 : **14.3%** reduction
From 285 to 60: **79%** reduction

Transfer-Learning Results: Cross-Platform

(*sp, kp, sb, ranks*)

Scenario	*Found at*	*Execut. Eval.*	*Optimal Config.*	*Eval. Time*
BO Perlmutter, 1 node	#47	70	$(1, 8, 4, 64)$	1.36s
Intra-Platform Portability				
Without TL	#43	70	$(1, 8, 2, 128)$	1.12s
With TL	#40	60	$(1, 8, 4, 128)$	0.91s
Cross-Platform Portability				
Without TL	#7	40	$(1, 8, 2, 32)$	6.76s
With TL	#40	60	$(1, 8, 4, 32)$	4.68s

Now moving to 1 Node in Cori:
92 valid combinations

Without TL stops earlier, but optimal is worse.

Assessing quality with metrics

A metric for measuring correlation among tasks in Transfer learning:

- Magnitudes near 1 indicate high correlation, close to 0 mean no relation between tasks.

- Intra-Platform results: **0.9498**
- Cross-Platform results: **0.834**

Platform	$(sp, kp, sb, ranks)$	*Efficiency*	*Evaluations Executed*	
Perlmutter, 1 node	$(1, 8, 4, 64)$	100%	70	
Without transfer learning				
Perlmutter, 2 nodes	$(1, 8, 2, 128)$	81.39%	70	Φ = 0.7492
Cori, 1 node	$(1, 8, 2, 32)$	69.23%	40	
With transfer learning				
Perlmutter, 2 nodes	$(1, 8, 4, 128)$	100%	60	Φ = 1
Cori, 1 node	$(1, 8, 4, 32)$	100%	60	

$\text{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Outline

- Contributions & Motivation
- Some ML Concepts
- DFTuning
- The RT-TDDFT Mini-App
- ML Methodology
- Experimental Results
- Conclusions

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

More details in the paper…

- Conversion-stopping criteria
- How to re-use DFTuning for more applications
- Avoid OpenMPI nested parallelism (MPI spawning)
- Evaluate in parallel the initial candidates
- Enabling compile-time performance parameters
- Correlation metric for tasks in the TL learned model
- Pennycook metric calculation
- …

$$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$$

Conclusions

- We present a novel ML-based Autotuning Methodology, based on BO + TL, for addressing the performance portability problem in TDDFT workload.
- The methodology has a broader applicability to other apps and tuning frameworks.
- We show promising results with TL on performance portability:
  - Saves up to **46.7%** of app evaluations compared to a BO search in NERSC platforms.
  - We demonstrate with a new metric why TL worked here.
  - Pennycook performance portability metric shows the highest portability.

$\mathrm{cov}(u_q(x), u_q(x')) = k_q(x, x')$

Future work

- Target GPU platforms.
- Study the importance of the Task/Platform parameters.
- Focus on large scale executions.

Thank you

aperezdieguez@lbl.gov