GPTuneBand: Multi-task and Multi-fidelity Autotuning for Large-scale High Performance Computing Applications

Xinran Zhu[†], Yang Liu[‡], Pieter Ghysels[‡], David Bindel[†], Xiaoye S. Li[‡]

Abstract

This work proposes a novel multi-task and multi-fidelity autotuning framework, GPTuneBand, for tuning large-scale expensive high performance computing (HPC) applications. GPTuneBand combines a multi-task Bayesian optimization algorithm with a multi-armed bandit strategy, well-suited for tuning expensive HPC applications such as numerical libraries, scientific simulation codes and machine learning (ML) models, particularly with a very limited tuning budget. Our numerical results show that compared to other state-of-the-art autotuners, which only allows single-task or single-fidelity tuning, GPTuneBand obtains significantly better performance for numerical libraries and simulation codes, and competitive validation accuracy for training ML models. When tuning the Hypre library with 12 parameters, GPTuneBand wins over its single-fidelity predecessor GPTune on 62.5% tasks, with a maximum speedup of 1.2x, and wins over a single-task, multi-fidelity tuner BOHB on 72.5% tasks. When tuning the MFEM library on large numbers of CPU cores, GPTuneBand obtains a 1.7x speedup when compared with the default code parameters.

1 Introduction

Autotuning aims at automatically finding code parameters that optimize the runtime, memory, communication cost, or accuracy for complex, black-box functions. Specifically, this technique can be applied to hyperparameter tuning of ML models such as neural networks and support vector machines [7, 50, 8, 54, 29, 36, 30, 16, 57], as well as tuning of performance-critical parameters for large-scale numerical libraries and first-principle simulation software packages [37, 35, 47]. Model-free optimization and Bayesian optimization (BO) are two main families of black-box tuning methods. Model-free optimization methods include global approaches such as simulated annealing [28], genetic algorithms [52] and particle swarm optimization [26], and local approaches such as Nelder–Mead simplex [41] and Orthogonal Search [11]. In contrast, Bayesian optimization methods [21, 49, 17] typically model the tuning objective as a Gaussian process (GP) [46], adaptively sample the objective to update the GP, and use the posterior GP as a surrogate model to search for new promising samples. When tuning expensive and noisy objectives, e.g., performance of large-scale HPC applications, both methods face significant challenges due to the high cost of reliable samples. Strategies to address these challenges include:

Multi-task tuning. In many tuning scenarios, there are correlated tuning tasks, and one can use such correlation to improve performance on each task. Multi-task tuning has been used for continuous sets of tasks [39, 38] and discrete tasks [31, 54, 44, 13, 35]. In the discrete case, one can use a multi-output GP model, such as the linear coregionalization model (LCM) [3, 22] or the intrinsic model of coregionalization (IMC) [3, 18, 9], to model performance across correlated tasks [54, 44, 13, 35]. Some works further extended acquisition functions in BO to multi-task settings [31, 54, 44, 13].

Multi-fidelity tuning. Many applications can execute at multiple fidelity levels, where the highest-fidelity runs correspond to the true, but most expensive objective function, and lower-fidelity ones are less accurate but computationally inexpensive. A multi-fidelity autotuner can run many low-fidelity evaluations (application runs) with smaller cost, quickly identify promising samples (parameter configurations), and then allocate more resources for high-fidelity evaluations (application runs) with those samples (parameter configurations). Some works focused on extending BO algorithms to multi-fidelity settings and adapted various acquisition functions in BO to multi-fidelity settings [19, 32, 23, 24, 45, 30, 57, 55]. Another line of work is to use multi-armed bandit strategy for multi-fidelity sampling to better balance low-fidelity evaluations and high-fidelity evaluations [33, 56].

To further improve tuner performance for large-scale HPC applications, we propose GPTuneBand, a publicly available autotuner allowing both multi-task and multi-fidelity tuning. GPTuneBand is specially designed for large-scale HPC applications, where application runs are expensive and limited to a relatively small number, but it

† Cornell University, Ithaca, NY 14850, USA (xz584, bindel@cornell.edu).

‡ Scalable Solvers Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA (liuyangzhuan, pghysels, xsli@lbl.gov).

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

is also well suited for tuning any applications that needs multi-task tuning and allows multi-fidelity evaluations.

Our main contributions are summarized below:

- GPTuneBand is an autotuning framework that allows both multi-task and multi-fidelity tuning. The key idea is to build LCM models across both tasks and fidelity levels to guide sampling and then use a multi-armed bandit strategy to select potentially strong candidates from lower-fidelity samples. Compared to neural network-based multi-task and multi-fidelity autotuners [56], our LCM-based lightweight autotuner requires much smaller numbers of samples to build informative models.
- GPTuneBand applies to HPC codes on a variety of shared-memory or distributed-memory, CPU or CPU-based machine. In addition, GPTuneBand itself is also distributed-memory parallel for the efficient modeling process.
- GPTuneBand is an extension of the publicly available multi-task, single-fidelity autotuning framework GPTune [35], and therefore it is also publicly available and inherits all salient features of GPTune, such as a historical database, crowd tuning, user-provided performance models, and reverse communication interface (RCI) to improve tuning efficiency.
- Our empirical study shows that GPTuneBand outperforms several other state-of-the-art autotuners on a wide range of large-scale scientific applications as well as ML models.

The paper is organized as follows. Section 2 discusses related state-of-the-art work as our reference algorithms. Section 3 introduces the notations of the general tuning problem, the multi-task tuning problem in the BO framework, and the multi-fidelity tuning problem in the multi-armed bandit framework. Section 4 describes the GPTuneBand algorithm with an illustrative example. Section 5 shows the empirical study of GPTuneBand on tuning various HPC and machine learning applications and compare with the other state-of-the-art autotuners.

2 Related work

In Section 1, we discussed related work in the general autotuning literature, and related work specifically in the multi-task tuning literature and multi-fidelity tuning literature. Here, we highlight some specific stat-of-the-art autotuners as the reference algorithms. *OpenTuner*[5] is one of the state-of-the-art model-free autotuners, which combines most of the model-free optimization algorithms mentioned in Section 1 and then solves a multi-armed bandit problem [25] to allocate workloads to different optimization algorithms. OpenTuner does not support multi-task or multi-fidelity tuning. *Hyperband* (HB) [33] is a model-free multi-fidelity tuner based on the bandit strategy and random sampling. It balances the number of objective evaluations and the fidelity by using the bandit strategy, and it will be described in detail in Section 3.2. *HpBandSter* (BOHB) [16] is a BO variant of HB. It differs from HB in that it builds Tree Parzen Estimator (TPE) [7] models to guide sampling. These tuners do not support multi-task tuning. Similarly, HB-ABLR [56] is a multi-task and multi-fidelity tuning algorithm that combines HB with adaptive Bayesian linear regression (ABLR) [42]. However, it relies on neural networks, and thus requires large number of evaluations, to build task correlation. *GPTune* [35] is a multi-task autotuning software package targeted at exascale applications, which relies on LCM surrogate models to efficiently learn task correlations. GPTune does not support multi-fidelity tuning, and it will be desribed in detail in Section 3.1. In addition, a multi-task search space refinement algorithm [43] is also developed.

Algorithm 1 Bayesian optimization-based MLA

1: **Sampling:** Evaluate $y(x, t_i)$ at $n = n_{tot}/2$ initial random samples for each task $t_i \in \mathcal{T}$.
2: **while** $n < n_{tot}$ **do**
3: **Modeling:** Update hyperparameters of the LCM of $\{y(x, t_i)\}_{i \le m}$ using all available data.
4: **Search:** Search for an optimizer x_i^* for the EI of task $t_i \in \mathcal{T}$. Let $X^* = [x_1^*, x_2^*, \ldots, x_m^*]$.
5: Evaluate $y(x, t_i)$ for $t_i \in \mathcal{T}$ at the new tuning parameter configurations X^*.
6: $n \leftarrow n + 1$.
7: **end while**
8: Return the optimal tuning parameter configurations and objective function values for each task.

3 Background

The goal of tuning is to find the parameter configurations that optimize an application's performance such as runtime, accuracy, or memory use. The tuning objective is a function $f : \mathcal{X} \to \mathbb{R}$ over a d-dimensional tuning parameter space $\mathcal{X} \subset \mathbb{R}^d$ (discrete and categorical parameters are mapped to $\mathbb{R}$ if needed). We model observations of the objective, which may be noisy, by $y(x) = f(x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_{noise}^2)$. In the following, we describe the multi-task tuning algorithm of GPTune [35], and the multi-fidelity tuning algorithm of Hyperband [33], both of which are closely related to our GPTuneBand algorithm.

3.1 Multi-task tuning in the BO framework

In a multi-task setting, one has several tuning tasks

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

for an application, for example tuning the runtime of a linear algebra solver on several different matrix sizes. We parameterize tasks by $t \in \mathbb{T} \subset \mathbb{R}^\alpha$, where $\mathbb{T}$ is an α-dimensional task space. The objective can then be written as $f(x,t)$ with one task argument t. We denote by $\mathcal{T} = \{t_1, t_2, \ldots, t_m\} \subset \mathbb{T}$ a set of m tuning tasks.

The Efficient Global Optimization (EGO) [21] is a classical BO algorithm that builds a Gaussian process (GP) [46] as a surrogate model and then optimizes the Expected Improvement (EI) [40] acquisition function to determine the next sample. The multi-task BO algorithm used in GPTune, called the Multi-task Learning Autotuning (MLA) algorithm, extends EGO to the multi-task setting. MLA consists of three main phases: sampling, modeling and search [35].

Sampling. MLA evaluates a set of n initial samples for each task. We let $X_i \in \mathbb{R}^{d\times n}$ and $Y_i = [y_{i,1}, \ldots, y_{i,n}] \in \mathbb{R}^n$ denote initial samples and evaluated objective values for task t_i, and $X = [X_1, X_2, \ldots, X_m] \in \mathbb{R}^{d\times nm}$ and $Y = [Y_1; Y_2; \ldots; Y_m] \in \mathbb{R}^{mn}$ denote initial samples and evaluated objective values for all m tasks.

Modeling. MLA models the joint distribution across tasks by LCM [3, 22] which generalizes GPs to the multi-task setting. Specifically, LCM models objective $f(x,t_i)$ for each task $t_i \in \mathcal{T}$ by linear combinations of $Q \le m$ latent random functions:

$$f(x,t_i) = \sum_{q=1}^{Q} a_{i,q} u_q(x),$$

where $a_{i,q}$ are hyperparameters to learn and $u_q(x)$ are latent functions. Each latent function is an independent zero-mean GP with e.g., a squared exponential (SE) kernel

$$k_q(x,x') = \sigma_q^2 \exp\left(-\sum_{i=1}^{d} (x_i - x_i')^2 / l_i^q\right),$$

where σ_q^2 is the variance and l_i^q are length scales. Thus, the covariance matrix for all samples of all tasks, $\Sigma(X,X) \in \mathbb{R}^{mn\times mn}$, has entries

$$\Sigma(x_{i,j}, x_{i',j'}) = \sum_{q=1}^{Q} (a_{i,q}a_{i',q} + b_{i,q}\delta_{i,i'}) k_q(x_{i,j}, x_{i',j'}) + d_i \delta_{i,i'}\delta_{j,j'},$$

where $\delta_{i,j}$ is the Kronecker delta function, and $b_{i,q}$ and d_i are regularization parameters. The LCM model hyperparameters can then be estimated by maximizing the log-likelihood using gradient-based optimization methods such as L-BFGS [34].

Search. The LCM predicts the joint distribution at new points $[x_1^*, x_2^*, \ldots, x_m^*]$ with posterior mean $\mu^* = [\mu_1^*, \mu_2^*, \ldots, \mu_m^*]$ and posterior variance $\sigma^{*2} = [\sigma_1^{*2}, \sigma_2^{*2}, \ldots, \sigma_m^{*2}]$ as:

$$\mu^* = \Sigma(X^*, X)\Sigma(X,X)^{-1}Y,$$
$$\sigma^{*2} = \mathrm{diag}(\Sigma(X^*,X^*) - \Sigma(X^*,X)\Sigma(X,X)^{-1}\Sigma(X,X^*)),$$

which can be used to construct the EI acquisition function. The EI is then optimized to select the next point X^* for each task. We refer to the GPTune paper [35] for more details on the search phase.

One iteration finishes when a new sample X^* is evaluated and it repeats until the sample budget n_{tot} is exhausted. Algorithm 1 summarizes the MLA iterations.

Algorithm 2 Hyperband using SuccessiveHalving (SH) as a subroutine

Inputs: η, fidelity bounds $b_{\min}$ and $b_{\max}$

1: compute $s_{\max} = \left\lfloor \log_\eta \frac{b_{\max}}{b_{\min}} \right\rfloor$.
2: **for** $s \in \{s_{\max}, s_{\max}-1, \ldots, 0\}$ **do**
3: compute $N(s) = \left\lfloor \frac{s_{\max}+1}{s+1} \right\rfloor \eta^s$.
4: compute $B(s) = b_{\max}\eta^{-s}$.
5: generate $N(s)$ random samples at fidelity $B(s)$ and run SH on them.
6: **end for**

3.2 Multi-fidelity tuning in the multi-armed bandit framework

In a multi-fidelity setting, the application can be run with different fidelity parameters, such as the discretization level in a partial differential equation (PDE) solver, or the number of iterations in an iterative algorithm. Lower fidelity runs are less accurate, but are also less expensive. We parameterize fidelities by $b \in [b_{\min}, b_{\max}]$, where $b_{\min}, b_{\max}$ are given minimum and maximum fidelity levels. Here we assume the function evaluation time is linear with the fidelity b. In this way, the objective function can be augmented as $f(x,b)$ with one more fidelity input argument b. The tuning goal is to find tuning parameters x to optimize the objective at the highest fidelity, i.e., $f(x, b_{\max})$. Evaluations at a low fidelity $f(x,b)$ where $b < b_{\max}$ serve as approximations of $f(x, b_{\max})$.

Hyperband (HB) [33] (Algorithm 2) is a recent model-free multi-fidelity tuning algorithm. Based on bandit strategy, it collects sets of random samples at different fidelity levels and then calls successive halving (SH) [20] (Algorithm 3) to promote promising samples to higher fidelity evaluations. Given a fidelity bound $[b_{\min}, b_{\max}]$ and a "halving" factor η, Hyperband's multi-armed bandit strategy prescribes multiple brackets $\{0, 1, \ldots, s_{\max}\}$, where 0 corresponds to the highest fidelity and $s_{\max} = \lfloor \log_\eta b_{\max}/b_{\min} \rfloor$. Thus, the associated fidelity levels are $\{b_{\max} \simeq \eta^{s_{\max}} b_{\min}, \ldots, \eta b_{\min}, b_{\min}\}$, a

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

geometrically decreasing sequence. Here bracket s requires $N(s)$ initial samples at fidelity $B(s)$. Note that $b_{\min}, b_{\max}$ and η are designed for approximately equal evaluation costs across brackets. After a bracket s collects all initial samples, it evaluates all existing samples in the bracket, keeps the best η ones, and evaluates them at a fidelity η times larger. This process repeats until the highest fidelity $b_{\max}$ is reached in bracket s. Finally, Hyperband selects the best sample at the highest fidelity among all brackets.

By using the multi-armed bandit strategy, Hyperband achieves good balance between the exploration via low fidelity sampling and the exploitation via high fidelity sampling. Instead of generating random samples for SH, HpBandSter (BOHB) [16] generates samples for SH using BO methods via TPE [7] models. However, neither HB nor BOHB supports a multi-task tuning setting. Moreover, even for single-task tuning, they do not exploit correlations among different fidelity levels during sample collection.

Algorithm 3 SuccessiveHalving

Inputs: $B(\cdot)$, s, η, a set C containing $N(s)$ configuration samples.

1: **for** $i \in \{s, s-1, \ldots, 1\}$ **do**
2: Evaluate all configuration samples from C at $B(i)$.
3: Select the top $\lfloor |C|/\eta \rfloor$ samples and form a new sample set C.
4: **end for**
5: Evaluate the current configuration sample set C at $B(0)$.
6: **Return** C containing selected samples at the highest fidelity.

4 The GPTuneBand algorithm

In this section, we introduce the proposed multi-task and multi-fidelity tuning algorithm GPTuneBand. Similar to HpBandSter (BOHB), GPTuneBand has three stages. First, the bandit strategy prescribes the total number of brackets, where bracket s requires $N(s)$ samples at starting fidelity $B(s)$. Second, BO algorithms are used to generate the $N(s)$ samples in each bracket. Third, each bracket runs the SH [20] algorithm (Algorithm 3) to find the best one(s) in each bracket.

We highlight several key differences between HpBandSter and GPTuneBand: (a) While HpBandSter build TPE models within each bracket to guide sampling, GPTuneBand uses LCMs across fidelities in different brackets to facilitate sampling. (b) GPTuneBand extends well to the multi-task setting by using LCMs across both fidelity brackets and tasks. (c) GPTuneBand can execute multiple passes to collect more samples and build more informative LCMs. Using correlations across fidelities and tasks, GPTuneBand needs significantly fewer total samples.

Since LCM is used across both brackets and tasks, we use "task" to refer to a user-specified tuning task, and "LCM-task" to refer to a sub-task used to construct LCMs. Specifically, an "LCM-task" is a bracket associated with a given fidelity level and a given tuning task. See Section 4.1 for more details and examples.

Algorithm 4 summarizes the GPTuneBand algorithm. The inputs are: a fidelity function $B(\cdot)$ mapping a bracket variable s to a fidelity level $B(s)$, a constant $s_{\max}$ determining the number of brackets, a constant "halving" factor η determining the number of samples of each bracket and the selection rate later in the SH run [20], and task parameters $\{t_i\}_{i=1}^m$ indicating tuning tasks. We explain Algorithm 4 in three stages:

Initialization stage. In this stage, we determine the number of samples $N(s)$ and the fidelity $B(s)$ for each bracket and each task. Given the constant $s_{\max}$, there will be $s_{\max}+1$ brackets indexed by bracket variables $s = 0, 1, \ldots, s_{\max}$. For bracket s, the number of samples $N(s)$ is geometrically-spaced:

$$N(s) = \lfloor (s_{\max}+1)/(s+1) \rfloor \eta^s.$$

The fidelity function $B(\cdot)$ maps a bracket variable s to a fidelity $B(s)$, satisfying $b_{\max} = B(0) > B(1) \cdots > B(s_{\max}) = b_{\min}$. In this way, different from Hyperband (Algorithm 2), we provide a more flexible way of bandit sampling, but it reduces to the Hyperband setup by setting $B(s) = b_{\max}\eta^{-s}$ and $s_{\max} = \lfloor \log_\eta(b_{\max}/b_{\min}) \rfloor$. In the remaining part of this paper, for clarity we use the same setup as Hyperband and we refer to one input tuple $[b_{\min}, b_{\max}, \eta]$ as one specific bandit structure.

Sampling stage. The next stage is to sample for all brackets of all tasks. The main for-loop in Algorithm 4 details the sampling approach. Specifically, it builds LCMs across both brackets and tasks to guide the sampling. We use the notation LCM(p, n) to represent one LCM with p LCM-tasks and n target samples for each LCM-task. In line 5 of Algorithm 4, "build or update LCM" works the same way as the MLA algorithm (Algorithm 1) iteratively build and update an LCM.

Selection stage. When a bracket of one task has finished the sampling stage, i.e., a bracket s reaches the target number of samples $N(s)$, it will be removed from the LCM and enter the selection stage. It then calls an SH [20] run to identify the best one(s) at the highest fidelity. At the end of the for-loop, after each bracket identifies the best ones, GPTuneBand then reports the best among them.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

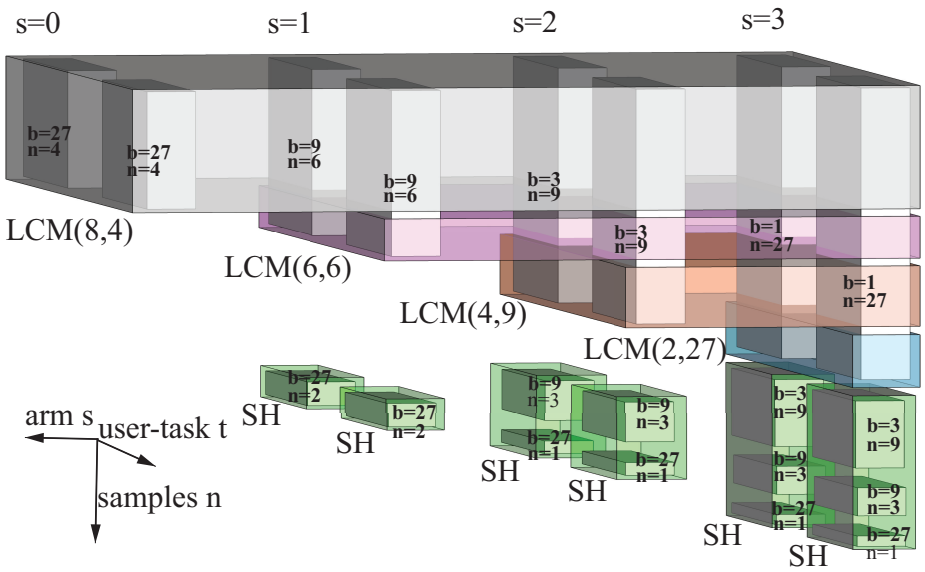

Figure 1: Illustrution of the proposed GPTuneBand algorithm with $b_{\min} = 1$, $b_{\max} = 27$, $\eta = 3$ and $m = 2$ tasks. Number of samples and corresponding fidelity are denoted by n and b respectively.

4.1 Illustrative example of GPTuneBand

In this section, we use a graphical example in Figure 1 to illustrate Algorithm 4. This example uses the $b_{\min} = 1$, $b_{\max} = 27$, $\eta = 3$ bandit structure and $m = 2$ tasks. Each solid vertical block represents one set of n samples at fidelity level b for one task. Each transparent horizontal block corresponds to one LCM (in gray, pink, orange and blue). Each transparent vertical block represents one SH run (in green).

From line 1 of Algorithm 4, for each of the two tasks, it generates 4 brackets $s = \{0, \ldots, 3\}$, with fidelities $B(0) = 27, B(1) = 9, B(2) = 3, B(3) = 1$, and number of samples $N(0) = 4, N(1) = 6, N(2) = 9, N(3) = 27$ respectively. The main for-loop of Algorithm 4 contains the sampling stage (LCM sampling) and the selection stage (SH runs). We explain the LCM sampling and the SH runs separately even though they are done concurrently in Algorithm 4.

For the LCM sampling (Algorithm 4, lines 4 and 5), GPTuneBand first builds an LCM with $m(s_{\max}+1) = 8$ LCM-tasks and 4 samples each, using samples from all brackets and tasks. Next, bracket $s = 0$ is removed from LCM sampling and starts an SH run (Algorithm 4, line 6). Without bracket $s = 0$, an LCM with $ms_{\max} = 6$ LCM-tasks and 6 samples each is then updated, which only requires 2 more samples per LCM-task from 2 additional MLA search iterations (same as the MLA's search phase in line 4 of Algorithm 1) as there exist 4 samples per LCM-task from the previous LCM sampling. This LCM sampling process continues until all the brackets and tasks have collected the target number of samples.

For the SH runs (Algorithm 4, line 6), it performs 6 SH runs in total, 3 brackets $s = 1, 2, 3$ multiplied with 2 tasks $t = 1, 2$ (bracket $s = 0$ requires no SH runs since it collects samples at the highest fidelity $b_{\max}$ directly). For one task, each SH starts using $N(s)$ samples at the starting fidelity $B(s)$, then boosts the fidelity for the top $1/\eta = 1/3$ configurations until the highest fidelity $b_{\max}$ is reached.

Algorithm 4 and Figure 1 only show one pass of GPTuneBand, while in fact GPTuneBand can do multi-pass tuning, with number of passes $n_p > 1$. In the first

Algorithm 4 GPTuneBand Algorithm (one pass)

Inputs: fidelity function $B(\cdot)$, $s_{\max}$, η, tasks: $t_1, \ldots, t_m$

1: Compute the starting number of samples $N(s)$ for each bracket s:
2: $N(s) = \left\lceil \frac{s_{\max}+1}{s+1} \right\rceil \eta^s$ for $s = 0, 1, \ldots, s_{\max}$.
3: **for** $s \in \{0, \ldots, s_{\max}\}$ **do**
4: build the current LCM-task set $T = \{T_{ij} : i = 1, \ldots, m, j = s, \ldots, s_{\max}\}$, where an LCM-task T_{ij} means task t_i with starting fidelity $B(j)$.
5: build or update LCM$(|T|, N(s))^{\dagger}$, where $|T| = m(s_{\max} - s + 1)$, until $N(s)$ configuration samples for each task $T_{ij} \in T$ are obtained.
6: run sucessive halving (SH) on each task T_{ij} in bracket s, i.e. on $T^s = \{T_{ij} : i = 1, \ldots, m, j = s\}$
7: **end for**
8: **Return** Configuration with the optimal objective value for each task t_i.

† LCM(p, n) means an LCM model with p LCM-tasks, with n target samples for each LCM-task.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

pass, each LCM(p, n) generates n new samples per LCM-task from scratch. In the following passes, however, each LCM(p, n) generates n new samples per LCM-task on top of all the historical data from previous passes, i.e., data generated by previous LCM sampling and SH runs. Therefore, more passes n_p could significantly improve the model quality of each LCM.

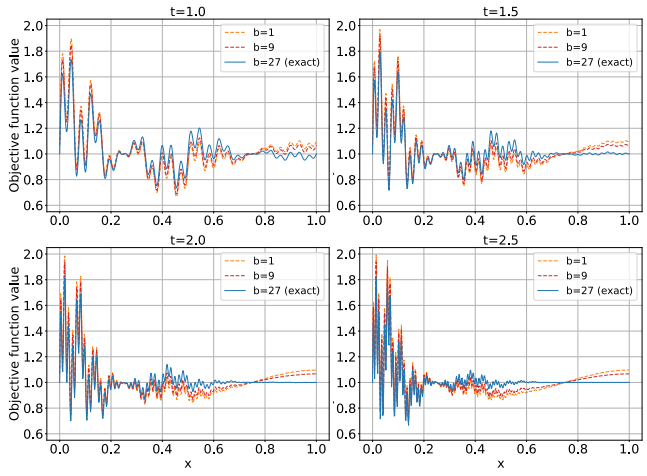

Figure 2: Example of multi-fidelity demo objective function for $t \in [1, 1.5, 2, 2.5]$ with fidelity level $b \in [1, 9, 27]$ and $b_{\max} = 27$.

5 Experiments

We run multi-task and multi-fidelity (MTMF) tuning to evaluate the tuning performance of GPTuneBand on one synthetic function and four real numerical applications: `hypre` [15], fast kernel ridge regression (KRR), `NIMROD` [51], `MFEM` [4], and one machine learning model: graph convolutional network (GCN). Experiments are run on the Cori machine at NERSC[1]: a Cray XC40 system with 2388 Haswell nodes, each of which consists of two 16-core Intel Xeon E5-2698v3 processors. Codes are available online [1], and more experiment details can be found in the supplement[2].

We compare with other state-of-the-art tuners: GPTune [35], HpBandSter (BOHB) [16], TPE [7] and OpenTuner (OT) [5]. GPTune supports multi-task and single-fidelity tuning; BOHB supports single-task and multi-fidelity tuning; TPE is the single-fidelity version of BOHB, i.e., a single-task BO algorithm based on TPE models; OpenTuner supports single-task and single-fidelity tuning.

[1] `http://www.nersc.gov/users/computational-systems/cori/`

[2] `https://drive.google.com/file/d/1naLrGS73gBaDOQ83wpg7FzBHnb0JNCXJ/view?usp=sharing`

To compare multi-fidelity tuners GPTuneBand and BOHB, we use the same multi-armed bandit structure in both cases. To compare with single-fidelity tuners (GPTune, OpenTuner and TPE), which always run the application at the highest fidelity, we use the same, normalized, total evaluation cost. Specifically, let the highest fidelity evaluation have unit cost, an evaluation at fidelity b would have normalized cost $b/b_{\max}$. We ensure the total sum of these normalized costs over all samples at all fidelities for one task remains the same across all autotuners.

Performance metrics. In multi-task settings, we define three metrics to evaluate the tuning performance of an autotuner. The **absolute performance** is the final optimal objective values found by an autotuner, while the **relative performance** is the ratio of the optimal objective values found by an autotuner to the optimal among all autotuners. Specifically, let $\{A_i\}_{i=1}^5$ be 5 autotuners. With tasks $\{t_j\}_{j=1}^m$, the absolute performance of autotuner A_i is defined as

$$R^{A_i} = \left[o_1^{A_i}, o_2^{A_i}, \ldots, o_m^{A_i}\right],$$

and its relative performance is then

$$\tilde{R}^{A_i} = \left[o_1^{A_i}/o_1^*, o_2^{A_i}/o_2^*, \ldots, o_m^{A_i}/o_m^*\right],$$

where $o_j^* = \min_{1 \le i \le 5}\{o_j^{A_i}\}$ is the best objective value of task t_j among all autotuners. Therefore, the closer each entry of $\tilde{R}^{A_i}$ is to 1, the better the relative performance autotuner A_i shows. For tuning with a relatively large number of tasks, we evaluate tuning performance by comparing the distribution of $\tilde{R}$ over tasks $\{t_j\}_{j=1}^m$ and use $\bar{R} = (\text{mean}(\tilde{R}) + \text{median}(\tilde{R}))/2$ as a metric to evaluate the relative performance, the closer to 1 the better. In addition, for tuning with a small number of tasks, it is more straightforward to evaluate the **tuning history**, which is an array of historical best objective values found by an autotuner during objective evaluations on one task. We plot and compare the tuning history (on the highest fidelity) versus thus-far the total normalized evaluation cost (using all fidelities).

5.1 Synthetic function

We first run MTMF tuning on a 1D demo function. The true objective is

$$y(x, t) = 1 + e^{-(x+1)^{t+1}} \cos(2\pi x) \sum_{i=1}^{3} \sin\left(2\pi x (t+2)^i\right),$$

where $x \in [0, 1]$ and $t > 0$. This function is highly non-convex, representing a very hard problem for black-box optimization. The tuning goal is to minimize $y(x, t)$ for multiple tasks t. To enable multi-fidelity evaluations, we add noise to the true objective, with a constant $e = 0.1$

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

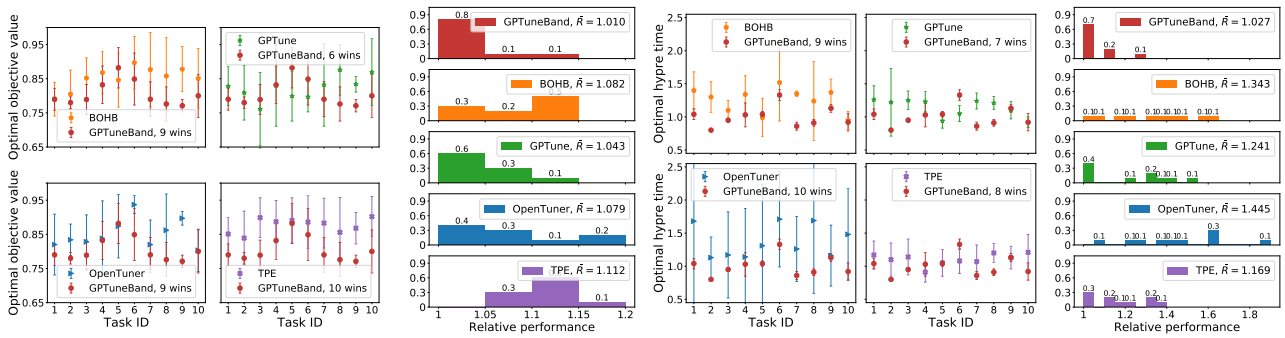

Figure 3: Detailed tuning results: 1) pair-wise comparison of absolute performance metric R^{A_i} and 2) histogram plot of relative performance metric $\tilde{R}^{A_i}$, where $i = 1, ..., 5$ correspond to five autotuners. Left: 10 demo tasks corresponding to ID 1 of Table 1. Right: 10 `hypre` tasks corresponding to ID 3 of Table 2.

ID	Task range	Bandit structure $[b_{\min}, b_{\max}, \eta]$	GPTuneBand	BOHB	GPTune	OpenTuner	TPE
1	[1, 1.45]	[1, 8, 2]	**1.010**	1.082	1.043	1.079	1.112
2	[1, 1.45]	[1, 16, 2]	**1.018**	1.064	1.039	1.054	1.094
3	[1, 1.45]	[1, 9, 3]	**1.022**	1.088	1.023	1.092	1.114
4	[1, 1.45]	[1, 27, 3]	**1.017**	**1.017**	1.050	1.060	1.051
5	[1, 1.45]	[1, 16, 4]	**1.016**	1.025	1.030	1.086	1.084
6	[1, 5.5]	[1, 27, 3]	**1.013**	1.090	1.107	1.131	1.090

Table 1: MTMF tuning of 10 demo tasks with different task ranges and different multi-armed bandit structures. The relative performance metric $\bar{R}$ of each autotuner is shown, the closer to 1 the better.

controlling the maximum noise level:

$$\tilde{y}(x, t, b) = y(x, t)\,(1 + e\cos(ax)(1 - b/b_{\max}))\,.$$

Hence, $\tilde{y}(x, t, b_{\max}) = y(x, t)$ is an exact evaluation, while $\tilde{y}(x, t, b_{\min})$ approximates $y(x, t)$ with the least accuracy. Figure 2 shows plots of demo function for $t \in [1, 1.5, 2, 2.5]$ with different fidelity levels $b \in [1, 9, 27]$. Specifically, $b = b_{\max} = 27$ provides exact evaluation of the demo function $y(x, t)$, while $b = b_{\min} = 1$ and $b = 9$ provide approximate evaluations with different accuracy.

Using this demo, we study how different bandit structures affect the performance of GPTuneBand. We select 10 tasks uniformly from a range of t and different bandit structures. Each experiment uses $n_p = 1$ pass and is repeated for 5 runs. Table 1 compares the relative performance, where GPTuneBand shows the best relative performance for all experiments (rows). As for the absolute performance, averaging over all 6 experiments, GPTuneBand wins over BOHB on 73% tasks, over GPTune on 67% tasks, over OT on 85% tasks, over TPE on 95% tasks. For more details, see Figure 3 (Left) for the pair-wise comparison of absolute performance R^{A_i} and the histogram of relative performance $\tilde{R}^{A_i}$ for ID 1 in Table 1. We conclude that, for this example, GPTuneBand performs the best and is not sensitive to the choice of bandit structure. Therefore, in the following, we mainly use the [1, 8, 2] and [1, 27, 3] structures, which work the best here.

5.2 HPC application: Hypre

The package `hypre` [15] contains several families of parallel algebraic multigrid preconditioners and solvers for large-scale sparse linear systems. Here we tune the runtime of GMRES with the BoomerAMG preconditioner for solving the convection-diffusion equation on structured 3D grids. Task parameters are defined by the a, c coefficients in the convection-diffusion equation: $-c\Delta u + a\nabla \cdot u = f$. There are 12 tuning parameters of integer, real and categorical types, for example processor topology parameters, total number of MPIs, the AMG strength threshold, the type of the parallel coarsening algorithm and the type of parallel interpolation operator. Details on tuning parameters can be found at [1].

The fidelity level is defined by the discretization with k^3 grid points, where k ranges from $k_{\min} = 10$ to $k_{\max} = 100$. Given the $\mathcal{O}(k^3)$ computational complexity of the algebraic multigrid algorithm, the fidelity mapping from b to k is a linear function interpolating $[b_{\min}, k^3_{\min}]$ and $[b_{\max}, k^3_{\max}]$. Experiments are run on 2 NERSC Cori nodes, with 5 repeated runs and $n_p = 1$ pass. Each highest-fidelity evaluation takes 5 seconds on average, depending on the value of the tuning parameter.

Using `hypre`, we study how the multi-task setting affects the performance of GPTuneBand. We randomly

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

ID	Task groups	Bandit structure $[b_{\min}, b_{\max}, \eta]$	GPTuneBand	BOHB	GPTune	OpenTuner	TPE
1	10	[1, 8, 2]	1.20	1.25	**1.02**	1.49	1.23
2	10	[1, 27, 3]	1.11	1.16	**1.01**	1.35	1.10
3	5+5	[1, 27, 3]	**1.03**	1.34	1.24	1.46	1.17
4	3+2+3+2	[1, 27, 3]	**1.03**	1.27	1.13	1.53	1.19
5	3+3+4	[1, 27, 3]	1.09	1.29	**1.08**	1.39	1.14
6	3+3+4	[1, 8, 2]	**1.10**	1.19	**1.10**	1.34	1.13

Table 2: MTMF tuning of 10 `hypre` tasks with different task groups: $m_1 + m_2$ represents one m_1-task tuning and one m_2-task tuning. Each task group performs an independent GPTune/GPTuneBand multi-task tuning. The relative performance metric $\bar{R}$ of each autotuner is shown, the closer to 1 the better.

select 10 tasks with $a, c \in [0, 1]$ and break tasks into different groups for multi-task tuning. This task group division only affects multi-task tuners GPTune and GPTuneBand, which perform an independent multi-task tuning per group.

A comparison of relative performance is summarized in Table 2. From Table 2, GPTuneBand outperforms almost all tuners but GPTune in 10-task tuning (ID 1&2). However, if the same 10 tasks are divided into smaller multi-task groups (ID 3,4,5,6) with each group containing tasks of similar coefficients a, c, GPTuneBand then performs the best or nearly best. As for the absolute performance, with smaller groups, on average GPTuneBand wins over BOHB on 72.5% tasks, over GPTune on 62.5% tasks, over OT on 85% tasks, and over TPE on 65% tasks. For more details, see Figure 3 (Right) for the pair-wise comparison of absolute performance R^{A_i} and the histogram plot of relative performance $\tilde{R}^{A_i}$ for ID 3 in Table 2. On `hypre`, BOHB, however, cannot even outperform its single-fidelity version TPE, whereas GPTuneBand shows effective multi-fidelity tuning.

For the observation that smaller groups improve GPTuneBand to outperform GPTune, one intuitive reason is that, for the same set of tasks, GPTuneBand builds more and larger size of LCMs than GPTune, which is more difficult to fit due to more LCM hyperparameters, especially for large task counts.

5.3 HPC-ML application: linear-complexity kernel ridge regression (KRR)

We run MTMF tuning of the KRR validation accuracy on datasets from the public repository [14]. We consider two tasks/datasets SUSY [6] and Occupancy [10], with $N = 10K$ and $N = 8K$ training data respectively. We use a linear-complexity (in N) KRR algorithm using the distributed-memory numerical software `STRUMPACK`[53] with the HSS compression algorithm. The tuning setting is similar to Algorithm 5 of [12] with two real-valued tuning parameters: the length scale h in the Gaussian kernel and the regularization parameter λ in ridge regression. Here we use the 5 autotuners to tune h and λ simultaneously without any grid search as in [12].

Given the linear-complexity of HSS-enhanced KRR, we define the fidelity by the percentage p of total training data, where $p_{\min} = 0.5$ to $p_{\max} = 1$. Therefore, the fidelity level is linear with evaluation cost. With the validation set fixed to 1K, a low-fidelity evaluation uses partial training data, while a highest-fidelity evaluation uses all training data. Given a bandit structure $[b_{\min}, b_{\max}, \eta]$, the fidelity mapping from b to p is a linear function interpolating $[b_{\min}, p_{\min}]$ and $[b_{\max}, p_{\max}]$. We use the $[b_{\min}, b_{\max}, \eta] = [1, 27, 3]$ bandit structure with $n_p = 3$ passes. Each highest-fidelity evaluation requires about 5 seconds on 2 NERSC Cori nodes.

A comparison of the tuning history is summarized in Figure 4. Tuning histories are averaged over 5 runs, with the shaded area being the standard error. We note that, on average, GPTuneBand outperforms its single-fidelity counterpart GPTune. For the first task on the left, all tuners, except TPE, find a good enough objective value fairly quickly and they also reach very similar final results. TPE performs the worst on the first task. For the second task, though all tuners reach similar final results, earlier tuning performance differs a lot. Noticeably, GPTuneBand outperforms all tuners, except only TPE; BOHB cannot even outperform its single-fidelity counterpart TPE and showed very large performance variances. Generally, on the two tasks among all tuners, GPTuneBand performs stably well with small variances, and consistently outperforms its single-task counterpart GPTune.

5.4 ML application: graph convolutional network (GCN)

We run MTMF tuning on another ML application using GPUs. The tuning goal is the validation accuracy of a graph convolutional network (GCN) [27] for semi-supervised graph node classification. We tune four GCN hyperparameters[3]: number of hidden units, initial learn-

[3]We use the GCN implementation in `https://github.com/tkipf/pygcn`.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

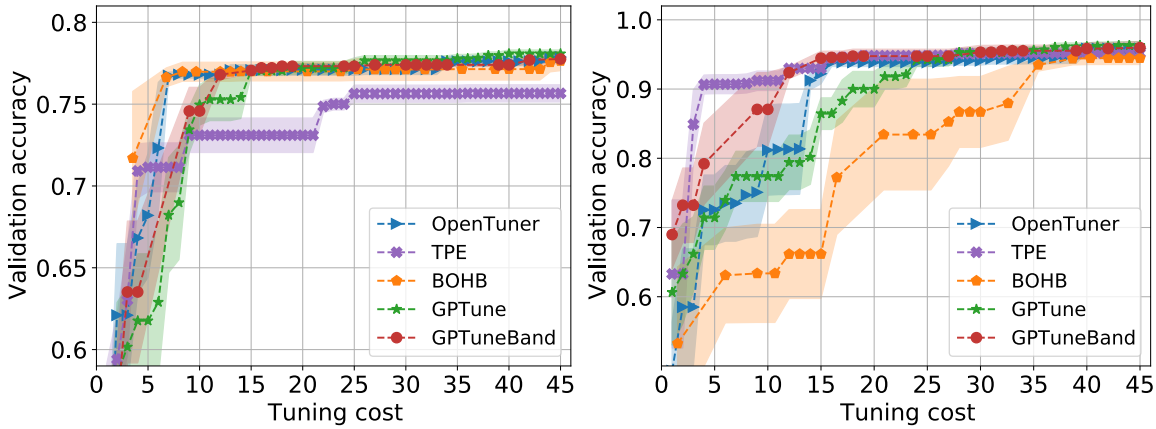

Figure 4: Tuning history of KRR on 2 tasks/datasets: SUSY (Left) and Occupancy (Right). Averaged over 5 runs. Plots are the historically best function values at the highest fidelity versus thus-far the total normalized evaluation cost using all fidelities.

ing rate, dropout rate for all layers, and weight decay (the L2 regularization factor in the GCN loss). We set up two graph datasets Cora [48] and Citeseer [48] as two tasks. The fidelity is defined by the number of training epochs K ranging from $K_{\min} = 100$ to $K_{\max} = 500$. Given a bandit structure $[b_{\min}, b_{\max}, \eta]$, the fidelity mapping from b to K is a linear function interpolating $[b_{\min}, K_{\min}]$ and $[b_{\max}, K_{\max}]$. We use the $[b_{\min}, b_{\max}, \eta] = [1, 27, 3]$ bandit structure with $n_p = 4$ passes. Each evaluation at highest fidelity requires several seconds using 1 NERSC Cori GPU (NVIDIA Tesla V100 GPU).

A comparison of the tuning history is summarized in Figure 5. Tuning histories are averaged over 10 runs, with the shaded area being the standard error. We note that, on average, GPTuneBand not only finds the best final objective value, but also optimizes the performance faster than other tuners. For this example, GPTuneBand shows efficient multi-task and multi-fidelity tuning.

5.5 HPC Application: NIMROD

In addition, we run single-task, multi-fidelity tuning on a large-scale HPC application `NIMROD` for modeling reactor-scale tokamaks. `NIMROD` is a time-marching code for modeling extended magnetohydrodynamic equations, whose most computationally expensive component is solving multiple nonsymmetric sparse linear systems using `SuperLU_DIST` at each time step.

In this experiment, we consider single-task tuning of the marching time by fixing a geometry model and discretization with 16 NERSC Cori nodes and a total of 512 MPIs. We pick four integer tuning parameters affecting computation granularity of matrix assembly in `NIMROD` and matrix factorization in `SuperLU_DIST`. The fidelity level is defined by the number of time steps t, where $t_{\min} = 3$ to $t_{\max} = 30$. Given that the runtime of one function evaluation is proportional to t, we map a fidelity level b to number of time steps t by linearly interpolating between $[b_{\min}, t_{\min}]$ and $[b_{\max}, t_{\max}]$.

For fast and reliable evaluations of the tuning objective, we use the so-called reverse communication interface (RCI, see Section 3.2.2 of [2]). However, other autotuners (OpenTuner, BOHB or TPE) do not support RCI, and therefore we only apply GPTune and GPTuneBand to tune `NIMROD`. Tuning histories are plotted in Figure 6 (left). GPTuneBand is able to find better runtime faster than GPTune, and the best runtime is about 7% faster than the default parameter configuration (the first sample in Figure 6).

5.6 HPC Application: MFEM

We also run single-task and multi-fidelity tuning on another large-scale HPC application `MFEM`. `MFEM` [4] is a high-performance scalable finite element discretization library, containing a collection of discretization algorithms, linear system solvers, and application drivers. Here we focus on using first order Nédélec elements on tetrahedral mesh to discretize a unit cube for solving highly indefinite Maxwell equations. Once the linear system is constructed, we solve it with the multi-frontal sparse solver `STRUMPACK`[53] with block low-rank (BLR) compression algorithms.

In this experiment, we consider single-task tuning of total `MFEM` runtime using 16 NERSC Cori nodes. There are three tuning parameters indicating the number of OpenMP threads per MPI, cutoff size of BLR compressed fronts, and leaf sizes in BLR compression. The fidelity level is defined by the mesh resolution on the geometry. Specifically, we use the $[b_{\min}, b_{\max}, \eta] = [1, 8, 8]$ bandit structure, corresponding to linear systems of size $N = 1872064$ (high fidelity) and $N = 238688$ (low fidelity).

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

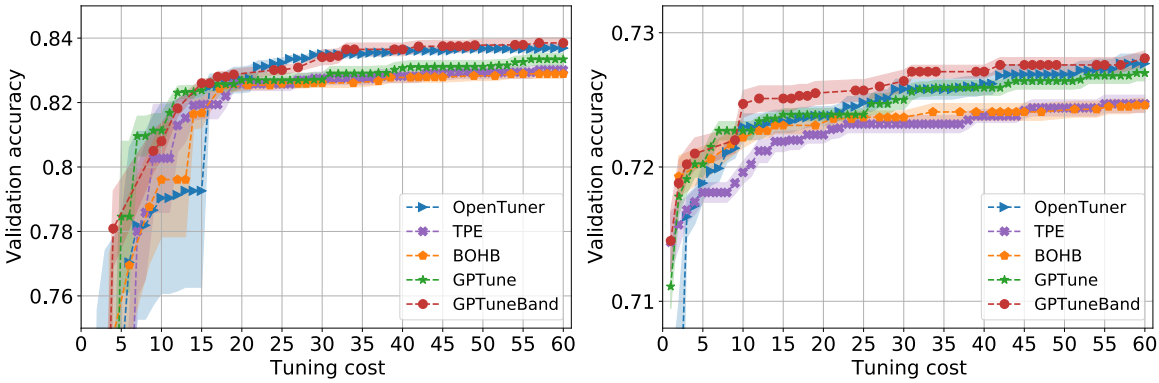

Figure 5: Tuning history of GCN on 2 tasks/datasets Cora (Left) and Citeseer (Right). Averaged over 10 runs. Plots are the historically best function values at the highest fidelity versus thus-far the total normalized evaluation cost using all fidelities.

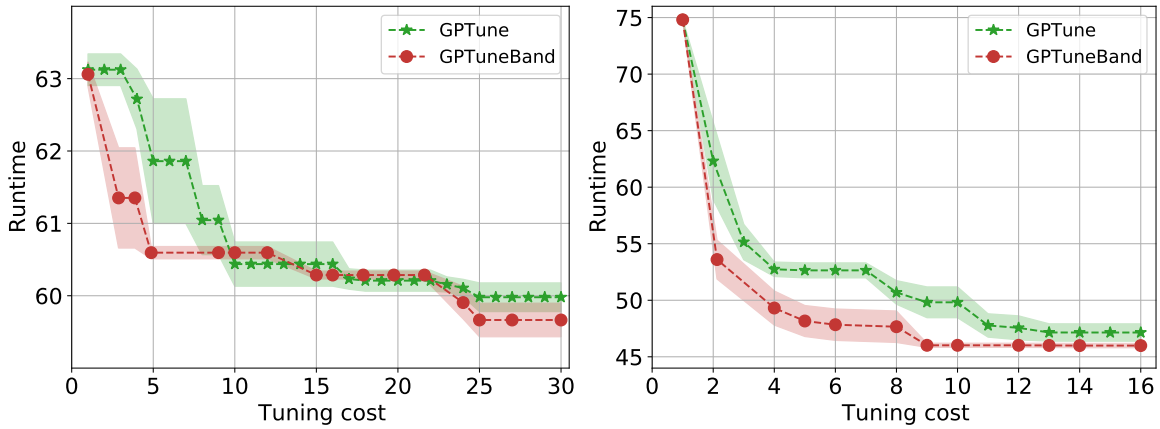

Figure 6: (Left) Tuning history of NIMROD: single task, bandit structure $[b_{\min}, b_{\max}, \eta] = [1, 8, 2]$. Averaged over 4 runs with the shaded area being the standard error. (Right) Tuning history of FMEM: single task, bandit structure $[b_{\min}, b_{\max}, \eta] = [1, 8, 8]$. Averaged over 7 runs with the shaded area being the standard error.

Same as `NIMROD`, we only use the RCI-enabled GPTune and GPTuneBand to tune `MFEM`. Tuning results are plotted in Figure 6 (right). GPTuneBand is able to find better runtime faster than GPTune, and the best runtime is about 1.7x (1.7 times) faster than the default parameter configuration (the first sample in Figure 6).

6 Conclusion

We developed GPTuneBand, a gernal tuning framework that supports both multi-task and multi-fidelity (MTMF) tuning, well-suited for tuning large-scale HPC applications when the number of application runs is limited. GPTuneBand outperforms other state-of-the-art autotuners on the MTMF tuning of a wide range of numerical software packages and machine learning methods. GPTuneBand is implemented within the framework of GPTune, which is a recently developed, publicly available, multi-task autotuner. Therefore, GPTuneBand essentially extends the GPTune tuning framework to a multi-fidelity setting, inherits all features of GPTune, and is also publicly available. Limitations, and therefore future developing directions of GPTuneBand may include: 1) a more rigorous investigation on task correlations, 2) an improvement of GPTuneBand on multi-task tuning for a large number of tasks, 3) a relaxation of the multi-task modeling, to allow different number of samples for each task in the LCM modeling.

Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and by the National Science Foundation under CCF-1934985 and the Simons foundation. We used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

References

[1] GPTune. https://github.com/gptune/GPTune.

[2] GPTune user guide. https://gptune.lbl.gov/documentation/gptune-user-guide.

[3] Mauricio A. Alvarez, Lorenzo Rosasco, and Neil D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4(3):195–266, 2012.

[4] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. MFEM: A modular finite element methods library. *Computers and Mathematics with Applications*, 81:42–74, 2021.

[5] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 303–315, 2014.

[6] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5:4308, 2014.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, page 2546–2554. Curran Associates, Inc., 2011.

[8] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 115–123. PMLR, 2013.

[9] Edwin V Bonilla, Kian Chai, and Christopher Williams. Multi-task Gaussian process prediction. In *Advances in Neural Information Processing Systems*, volume 20, pages 153–160. Curran Associates, Inc., 2008.

[10] Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39, 2016.

[11] Timothy M Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the twenty-seventh annual symposium on Computational geometry (SOCG)*, pages 1–10, 2011.

[12] Gustavo Chávez, Yang Liu, Pieter Ghysels, Xiaoye Sherry Li, and Elizaveta Rebrova. Scalable and memory-efficient kernel ridge regression. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 956–965, 2020.

[13] Sihui Dai, Jialin Song, and Yisong Yue. Multi-task Bayesian optimization via Gaussian process upper confidence bound. In *ICML 2020 Workshop on Real World Experiment Design and Active Learning*, 2020.

[14] Dheeru Dua and Casey Graff. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science, 2017.

[15] Robert D. Falgout and Ulrike Meier Yang. hypre: A library of high performance preconditioners. In *International Conference on Computational Science (ICCS)*, pages 632–641. Springer Berlin Heidelberg, 2002.

[16] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1437–1446. PMLR, 2018.

[17] Peter I Frazier. Bayesian optimization. In *Recent Advances in Optimization and Modeling of Contemporary Problems*, pages 255–278. INFORMS, 2018.

[18] Pierre Goovaerts et al. *Geostatistics for natural resources evaluation*. Oxford University Press on Demand, 1997.

[19] Deng Huang, Theodore T Allen, William I Notz, and R Allen Miller. Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32(5):369–382, 2006.

[20] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 240–248. PMLR, 2016.

[21] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[22] Andre G Journel and Charles J Huijbregts. *Mining geostatistics*. Academic Press, 1976.

[23] Kirthevasan Kandasamy, Gautam Dasarathy, Junier Oliva, Jeff Schneider, and Barnabás Póczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In *Advances in Neural Information Processing Systems*, volume 29, page 1000–1008. Curran Associates, Inc., 2016.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

[24] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, volume 70, pages 1799–1808. PMLR, 2017.

[25] Michael N Katehakis and Arthur F Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[26] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the International Conference on Neural Networks (ICNN)*, volume 4, pages 1942–1948. IEEE, 1995.

[27] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[28] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[29] Aaron Klein, Simon Bartels, Stefan Falkner, Philipp Hennig, and Frank Hutter. Towards efficient Bayesian optimization for big data. In *NIPS 2015 Bayesian Optimization Workshop*, 2015.

[30] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, Frank Hutter, et al. Fast Bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics*, 11(2):4945–4968, 2017.

[31] Andreas Krause and Cheng Ong. Contextual Gaussian process bandit optimization. In *Advances in Neural Information Processing Systems*, volume 24, pages 2447–2455. Curran Associates, Inc., 2011.

[32] Rémi Lam, Douglas L Allaire, and Karen E Willcox. Multifidelity optimization using statistical surrogate modeling for non-hierarchical information sources. In *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 0143, 2015.

[33] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[34] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[35] Yang Liu, Wissam M Sid-Lakhdar, Osni Marques, Xinran Zhu, Chang Meng, James W Demmel, and Xiaoye S Li. GPTune: Multitask learning for autotuning exascale applications. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 234–246, 2021.

[36] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65. PMLR, 2016.

[37] Harshitha Menon, Abhinav Bhatele, and Todd Gamblin. Auto-tuning parameter choices in HPC applications using Bayesian optimization. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 831–840. IEEE, 2020.

[38] Jan Hendrik Metzen. Minimum regret search for single- and multi-task optimization. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 192–200. PMLR, 2016.

[39] Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. Bayesian optimization for contextual policy search. In *Proceedings of the Second Machine Learning in Planning and Control of Robot Motion Workshop*. IROS Hamburg, 2015.

[40] Jonas Močkus. On Bayesian methods for seeking the extremum. In *Proceedings of the IFIP Technical Conference*, pages 400–404. Springer, 1975.

[41] John A Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[42] Valerio Perrone, Rodolphe Jenatton, Matthias Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6846–6856, 2018.

[43] Valerio Perrone, Huibin Shen, Matthias W Seeger, Cedric Archambeau, and Rodolphe Jenatton. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. *Advances in Neural Information Processing Systems*, 32:12771–12781, 2019.

[44] Matthias Poloczek, Jialei Wang, and Peter I Frazier. Warm starting Bayesian optimization. In *2016 Winter Simulation Conference (WSC)*, pages 770–781. IEEE, 2016.

[45] Matthias Poloczek, Jialei Wang, and Peter I Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems*, volume 30, pages 4291–4301. Curran Associates, Inc., 2017.

[46] Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[47] Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, and Devesh Tiwari. *Bliss: Auto-Tuning Complex Applications Using a Pool of Diverse Lightweight Learning Models*, page 1280–1295. Association for Computing Machinery, New York, NY, USA, 2021.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

[48] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[49] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. In *Proceedings of the IEEE*, volume 104, pages 148–175. IEEE, 2016.

[50] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, page 2951–2959. Curran Associates, Inc., 2012.

[51] C.R. Sovinec, A.H. Glasser, T.A. Gianakon, D.C. Barnes, R.A. Nebel, S.E. Kruger, S.J. Plimpton, A. Tarditi, M.S. Chu, and the NIMROD Team. Nonlinear magnetohydrodynamics with high-order finite elements. *Journal of Computational Physics*, 195:355, 2004.

[52] Mandavilli Srinivas and Lalit M Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[53] STRUMPACK – STRUctured Matrix PACKage, version 5.1.1, 2021.

[54] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 26, pages 2004–2012. Curran Associates, Inc., 2013.

[55] Shion Takeno, Hitoshi Fukuoka, Yuhki Tsukada, Toshiyuki Koyama, Motoki Shiga, Ichiro Takeuchi, and Masayuki Karasuyama. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 9334–9345. PMLR, 2020.

[56] Lazar Valkov, Rodolphe Jenatton, Fela Winkelmolen, and Cédric Archambeau. A simple transfer-learning extension of hyperband. In *NIPS Workshop on Meta-Learning*, 2018.

[57] Jian Wu, Saul Toscano-Palmerin, Peter I Frazier, and Andrew Gordon Wilson. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*, volume 115, pages 788–798. PMLR, 2020.

Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited