# Experimenting with GPTune for Optimizing Linear Algebra Computations

Makoto Morishita
morishita@hpc.itc.nagoya-u.ac.jp
Graduate School of Informatics, Nagoya University
Aichi-ken, Japan

Osni Marques
Yang Liu
Lawrence Berkeley National Laboratory
Berkeley, USA

Takahiro Katagiri
katagiri@cc.nagoya-u.ac.jp
Information Technology Center, Nagoya University
Aichi-ken, Japan

## 1 GPTUNE

GPTune[1] is an autotuning framework that solves an underlying black-box optimization problem, using surrogate modeling. GPTune uses Bayesian optimization based on Gaussian Process regression and supports advanced features such as multi-task learning, transfer learning, multi-fidelity/objective tuning, and parameter sensitivity analysis. GPTune targets the autotuning of HPC codes, in particular applications that are very expensive to evaluate.

## Problem description

- Input Space
  - This space defines the problems to be tuned. Every point in this space represents one instance of a problem.
- Parameter Space
  - This space defines the application parameters to be tuned. A point in this space represents a combination of the parameters. The tuner finds the best possible combination of parameters that minimizes the objective function associated with the application.
- Output Space
  - This space defines the objective of the application to be optimized. For example, this can be runtime, memory or energy consumption in HPC applications or prediction accuracy in machine learning applications.

## 2 EXPERIMENTS

| **Environment** (proxy for a distributed environment) | Name | OS | CPU | Memory |
|---|---|---|---|---|
| | MacBook Air (M1, 2020) | Ventura 13.4.1 | Apple M1 chip 8 cores | 8 [GB] |
| **Problem** $m$=$n$=1000 | Computation | | Tuning Parameters | |
| | **QR factorization** $A = QR$ <br> $A : A \in \mathbb{R}^{m \times n}$ <br> $Q$ : orthogonal matrix <br> $R$ : upper triangular matrix | | $nb$ | row block size |
| | | | $mb$ | column block size |
| | | | $p$ | number of processes |
| | | | $npernode$ | number of MPI processes per compute node |
| | **LU factorization** $A = LU$ <br> $A : A \in \mathbb{R}^{n \times n}$ <br> $L$ : lower triangular matrix <br> $U$ : upper triangular matrix | | $nb$ | row block size (= column block size) |
| | | | $p$ | row process grid |
| | | | $q$ | column process grid |

**Figure 1: Experimental environment**

In this work, we have focused on the autotuning of two algorithms implemented in ScaLAPACK[2]: QR and LU factorizations. In these cases, the tuning parameters are the block size (for cache memory optimization), and the process grid size for distributed computing.

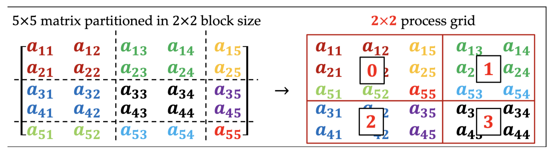


**Figure 2: Example of 2D block-cyclic distribution**

## 3 RESULT

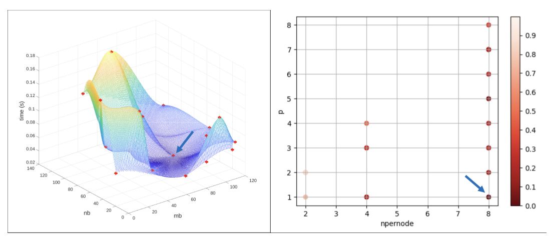


**Figure 3: Execution time for 4 tuning parameters Left: (mb, nb), Right: (npernode, p). The optimal configuration is given by mb=88, nb=64, npernode=8, p=1 (blue arrow).**

## 4 DISCUSSION

We have experimented with GPTune to find optimal parameters for QR and LU factorizations, therefore obtaining better performance. And we have learned how to select specific tuning parameters by changing the implementation of Space and Ojective Function in GPTune, and how to adapt the target application (in this case ScaLAPACK) to properly interface with GPTune.

## ACKNOWLEDGMENTS

This work was supported by JST SICORP Grant Number JPMJSC2201, Japan.

## REFERENCES

[1] GPTune-Dev. 2021. GPTune History Database. https://gptune.lbl.gov/.
[2] Berkeley; Univ. of Colorado Denver; Univ. of Tennessee; Univ. of California and NAG Ltd. 2022. ScaLAPACK-–Scalable Linear Algebra PACKage. https://www.netlib.org/scalapack/.