Experimenting with GPTune for Optimizing Linear Algebra Computations

Makoto Morishita[1], Osni Marques[2], Yang Liu[2], Takahiro Katagiri[1]
[1]Nagoya University, [2]Lawrence Berkeley National Laboratory

ABSTRACT

In High Performance Computing (HPC), software often has many parameters that impact its performance. However, it is difficult to determine optimal values for such parameters in an impromptu way. The automatic tuning – autotuning – of parameters is therefore an area of great interest. The purpose of this work is to understand the methodology of GPTune, which is an autotuning framework developed by DOE's Exascale Computing Project, and use the framework in a set of applications of interest.

GPTune

GPTune 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

❑**Spaces**

1. Input space: This space defines the problems to be tuned. Every point in this space represents one instance of a problem.

2. 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.

```
parameter_space = Space([Real(0., 1., transform="normalize", name="x")])
```

3. 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.

❑**Objective Function**

The user need to define a (Python) function representing the objective function to be optimized

```
def objectives(point):
    x = point['x']
    # call HPC code with parameter x
      …
    # get the function value f
      …
    return [f]
```

EXPERIMENTS

In this work, we have focused on the autotuning of two algorithms implemented in ScaLAPACK: QR and LU factorizations. The former is provided with the GPTune repository, while the latter was implemented during my visit. In these cases, the tuning parameters are the block size (for cache memory optimization), and the process grid size for distributed computing.

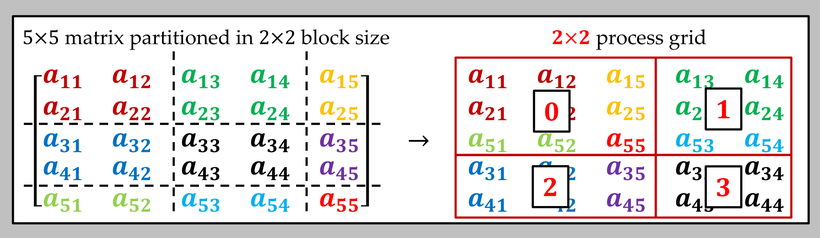


Figure 1. Example of 2D block-cyclic distribution

HARDWARE

	Name	OS	CPU	Memory
Environment (proxy for a distributed environment)	MacBook Air (M1, 2020)	Ventura 13.4.1	Apple M1 chip 8 cores	8 [GB]

	Computation	Tuning Parameters	
Problem m=n=1000	**QR factorization** $A = QR$ $A : A \in \mathbb{R}^{m \times n}$, Q : orthogonal matrix, 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 $A = LU$ $A : A \in \mathbb{R}^{n \times n}$, L : lower triangular matrix, U : upper triangular matrix	nb	row block size (= column block size)
		p	row process grid
		q	column process grid

QR factorization

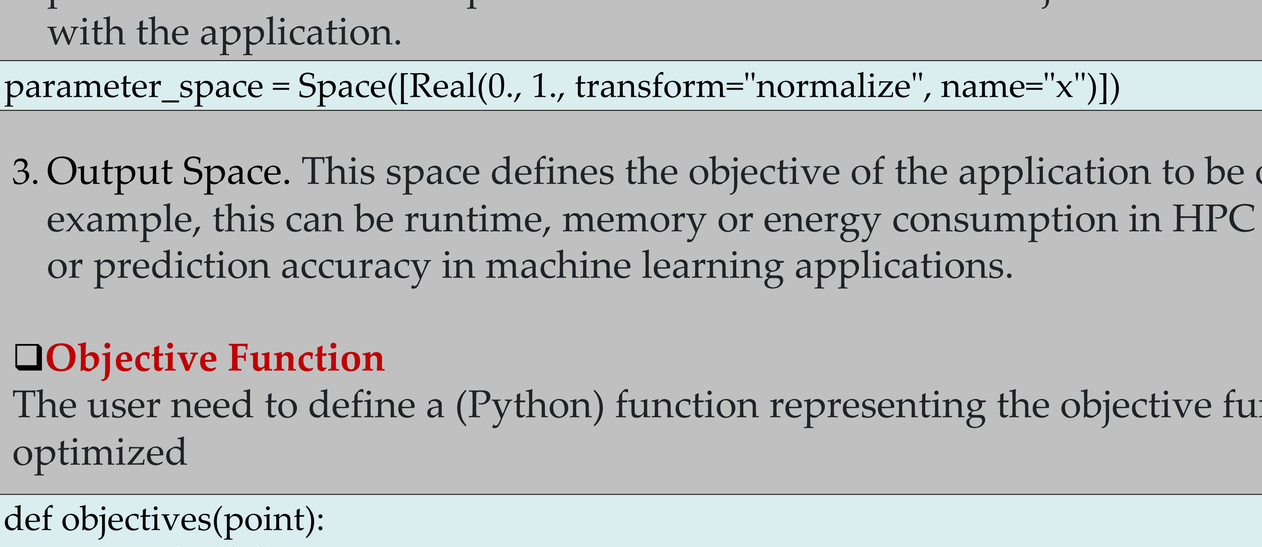


Figure 2. Execution time for 4 tuning parameters
Left: (mb, nb), Right: (npernode, p). The optimal configuration is given by mb=88, nb=64, npernode=8, p=1 (blue arrow).

LU factorization

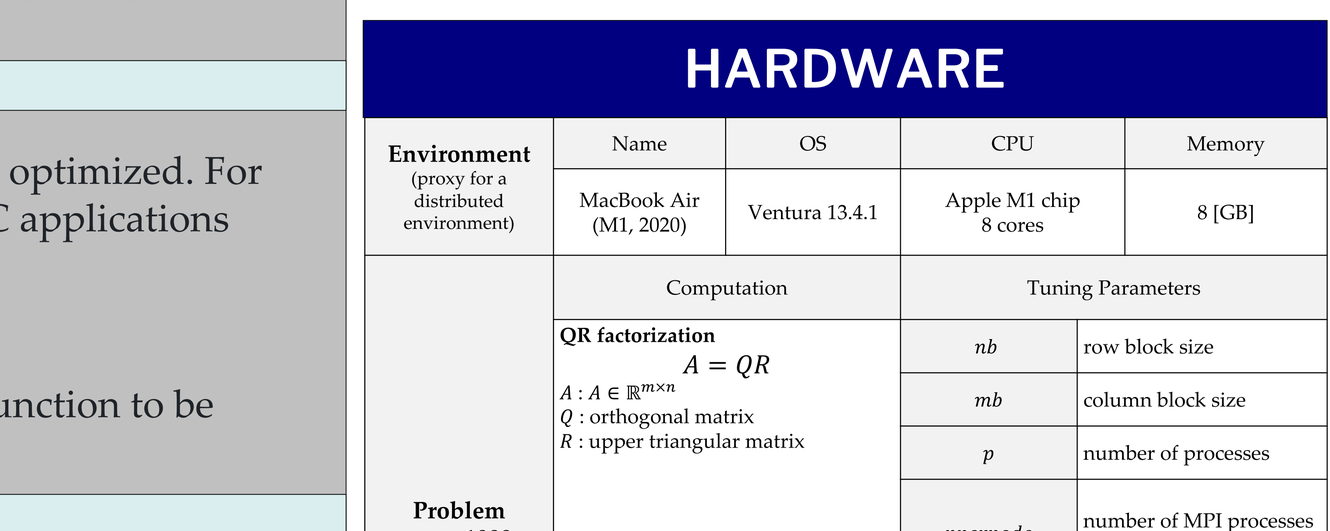
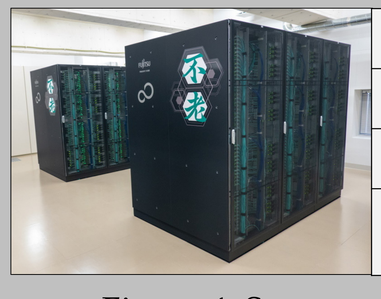


Figure 3. Execution time for 3 tuning parameters
Left: (nb, p), Right: (nb, q). The optimal configuration is given by nb=32, p=1, q=3 (blue arrow).

FUTURE WORK

- Install GPTune on the Supercomputer Flow Type I to examine its behavior on a Fugaku-type machine (arm architecture).
- Apply GPTune to applications of interest, including Fast Fourier Transform (e.g. FFTX, which is the exascale follow-on to the FFTW open source discrete FFT package).



Theoretical Performance	7.782 [TFLOPS]
Memory	72 [TiB] (32[GiB] / Node)
Number of Nodes	2,304 Nodes (110,592 cores)
CPU	A64FX (Armv8.2-A + SVE) 48 cores, 2.2 [GHz]

Figure 4. Supercomputer Flow Type I
(FUJITSU Supercomputer PRIMEHPC FX1000)

DISCUSSION

① We have experimented with GPTune to find optimal parameters for QR and LU factorizations, therefore obtaining better performance.
② 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.