Making Correctness Testing and Performance Autouning The Integral Parts of Software

X. Sherry Li
xsli@lbl.gov
Lawrence Berkeley National Laboratory

Collegeville Workshop on Sustainable Scientific Software (CW3S19),
July 22-24, 2019

Sustainability refers to the ability to maintain the scientifically useful capability of a software product over its intended life span, including understanding and modifying a software product's behavior to reflect new architectural advances.

Good methodologies to increase sustainability

- Automatic build system
  - Using CMake/Ctest increases build-test productivity and robustness, easier to manage dependencies on third-party software.
  - CMake supports builds for both Linux and Microsoft Windows.
- Open source repository
  - Svn → GitHub improved distributed contributions, making the code truly open-source.

Good methodologies to increase sustainability

- Automatic build system
  - Using CMake/Ctest increases build-test productivity and robustness, and easier to manage dependencies on third-party software
  - CMake supports builds for both Linux and Microsoft Windows.
- Open source repository
  - Svn → GitHub improved distributed contributions, making the code truly open-source.

- **Correctness testing**

- **Performance autotuning**

SuperLU numerical testing: Ax = b

- Regression test aims to provide coverage testing of all functionalities of the user-callable routines.
- Testing code structure:

```
For each I_1 ∈ set of valid values {
   For each I_2 ∈ set of valid values {
      ...
      For each I_q ∈ set of valid values {
         For each matrix type {
1.          Generate the input matrix A and right-hand side b;
2.          Call a user-callable routine with input values
            {I_1, I_2, ..., I_q};
3.          Compute the test metrics;
4.          Check whether each metric is smaller than a
            prescribed threshold;
         }
      }
      ...
   }
}
```

Matrix type	Description
0	sparse matrix `g10`
1	diagonal
2	upper triangular
3	lower triangular
4	random, $\kappa = 2$
5	first column zero
6	last column zero
7	last $n/2$ columns zero
8	random, $\kappa = \sqrt{0.1/\varepsilon}$
9	random, $\kappa = 0.1/\varepsilon$
10	scaled near underflow
11	scaled near overflow

Test Type	Test ratio	Routines
0	$\|\|LU - A\|\|/(n\|\|A\|\|\varepsilon)$	`dgstrf`
1	$\|\|b - Ax\|\|/(\|\|A\|\| \ \|\|x\|\|\varepsilon)$	`dgssv`, `dgssvx`
2	$\|\|x - x^*\|\|/(\|\|x^*\|\|\kappa\varepsilon)$	`dgssvx`
3	$\|\|x - x^*\|\|/(\|\|x^*\|\| \ FERR)$	`dgssvx`
4	$BERR/\varepsilon$	`dgssvx`

$$B'err = \max_i \frac{|r_i|}{(|A| \cdot |x| + |b|)_i}$$

$$F'err = \frac{\left\| x - x^* \right\|_\infty}{\left\| x^* \right\|_\infty}$$

Malloc/free balance check

- Debugging mode SUPERLU_MALLOC / SUPERLU_FREE

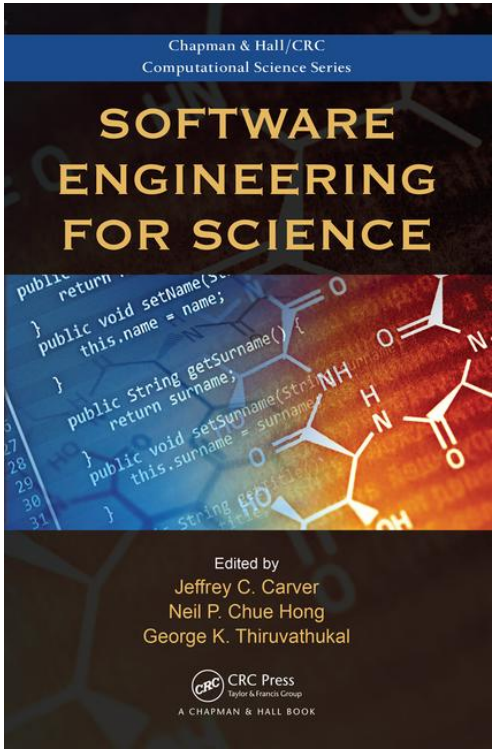
```
void *superlu_malloc(size_t size)
{
    char *buf;
    buf = (char *) malloc (size + 64);
    buf[0] = size;
    malloc_total += size;
    return (void *) (buf + 64);
}
```

```
void *superlu_free(void *addr)
{
    char *p = ((char *) addr) - 64;
    int n = ((size_t *) p)[0];
    malloc_total  -= n;
    free (p);
}
```

"Testing in Scientific Software: Impacts on Research Credibility, Development Productivity, Maturation, and Sustainability,"

Chapter in "Software Engineering for Science", Jeffrey Carver, Neil P. Chue Hong, George K. Thiruvathukal (editors), October 20, 2016, CRC Press.

Roscoe A. Bartlett, Anshu Dubey, Xiaoye S. Li, J. David Moulton, James M. Willenbring, and Ulrike Meier Yang (2016),



Autotuning method in xSDK4ECP project

- Develop an autotuning capability that learns optimal parameter selection and effectively replaces explicit human-based parameter selections.
- Collaborating with Y-Tune project led by Mary Hall, using complementary methods.

- Assumptions:
  - Execution of an ECP application corresponds to an expensive function evaluation.
  - Often, each application code solves one type of problems (e.g., certain PDE), with "similar" performance characteristics, even though sizes may vary.

- Optimization metrics: runtime, memory, energy, ⋯

Leverage statistical & machine learning

- Bayesian black-box optimization method based on multi-output Gaussian process.
- Specifically, use multitask and transfer learning to exploit the correlation among the multiple function evaluations at different parameters to build the learning model which can choose the parameter setting for the unseen task.

- Applications input the following to the tuner:
  - Entry function: F (p1, p2, …)
  - Exposing parameters
    - Integer: range
    - Real: range
    - Categorical
  - Constraints
- Autotuner calls entry function repeatedly with optimization algorithms to explore search space.

GPtune Python interface

Example: ScaLAPACK QR factorization

- Tasks: dimensions (m, n)
- 4 parameters: process grid (nproc = P*Q), Block size (mb, nb)

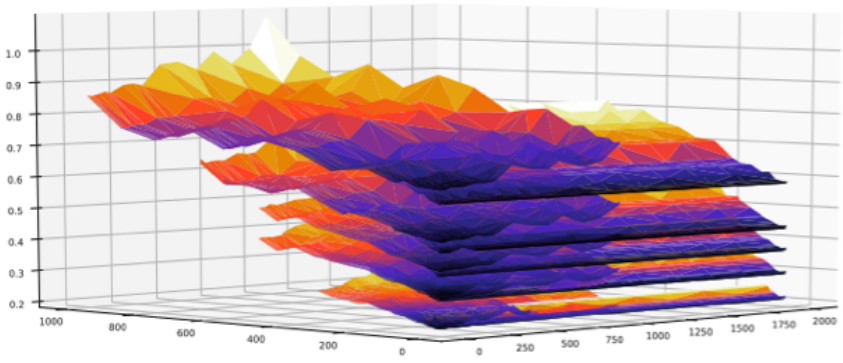
```
97      m     = Integer (name="m",     range=(1 , mmax))
98      n     = Integer (name="n",     range=(1 , nmax))
99      mb    = Integer (name="mb",    range=(1 , mmax))
100     nb    = Integer (name="nb",    range=(1 , nmax))
101     nproc = Integer (name="nproc", range=(nodes , nodes*cores))
102     p     = Integer (name="p",     range=(1 , nodes*cores))
103     r     = Real    (name="r",     range=(float("-Inf") , float("Inf")))
104
105     TS = Space(params=[m, n])
106     IS = Space(params=[mb, nb, nproc, p])
107     OS = Space(params=[r])
108
```

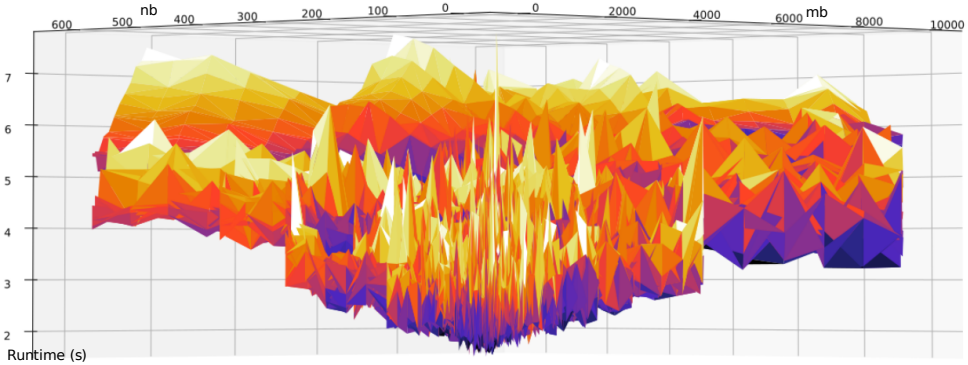
Python interface to define user function

```
109     def myobjfun(m, n, mb, nb, nproc, p):
110
111 #         return np.random.rand(1)
112         nth   = int(nodes * cores / nproc)
113         q     = int(nproc / p)
114
115 # [("fac", 'U10'), ("m", int), ("n", int), ("nodes", int), ("cores", int), ("mb", int), ("nb", int), ("r
116         params = [('QR', m, n, nodes, cores, mb, nb, nth, nproc, p, q, 1.)]
117
118         repeat = True
119 #         while (repeat):
120 #             try:
121         elapsedtime = pdqrdriver(params, niter = 3)
122         repeat = False
123 #             except:
124 #                 print("Error in call to ScaLAPACK with parameters ", params)
125 #                 pass
126
127         print(params, elapsedtime)
128
129         return elapsedtime
```

Example: ScaLAPACK QR factorization

- Semi-exhaustive search results:





1 node Edison, m = n = 2000
X-axis: MB, Y-axis: NB

Each layer is one (P,Q) configuration

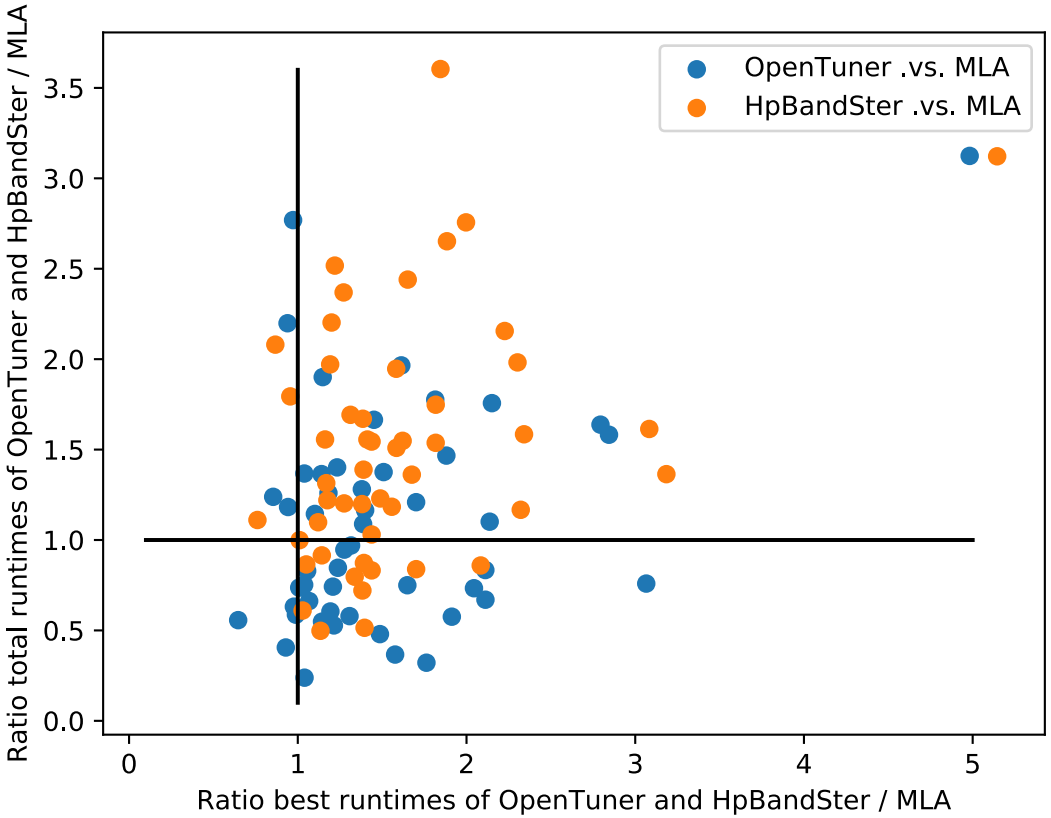
128 nodes Edison, m = n = 10,000
X-axis: MB, Y-axis: NB

Function non-smooth

Example: ScaLAPACK QR factorization

- Tasks: dimensions (m = n); parameters (mb, nb, nprocs, p).
- 50 tasks (dimension 1 − 20,000, following Latin Hypercube), 20 samples per task, 128 nodes of NERSC Edison
- Comparison between MLA and OpenTuner, HpBandSter

- MLA finds better parameters in 42 cases comp. to OpenTuner, in 47 cases comp. to HyBandSter
- Average 1.5x improvement



On-going work

- More examples:
  - SuperLU sparse direct solver: 7 parameters
    - For a set of un-related matrices chosen from SuiteSparse, GPtune is similar to OpenTuner
    - For a set of matrices in the same family, GPtune is better
  - Hypre AMG preconditioner to GMRES
    - 3D Poisson, const. coeff. Isotropic: 12 parameters

- Consider to incorporate algorithmic and hardware performance models in the autotuner.

- Plan to release GPtune end of FY19.

Thank You

Multitask and transfer learning

- Example: Co-Kriging exploits correlation of sampled executions in parameter space.

- Red / Purple curves represent cheap / expensive variables.
- Only sample 4 points with expensive variables
- Use the knowledge of 11 points of cheap samples.
- Co-Kriging model (Orange) is more accurate than Kriging (Green) to predict expensive curve