Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

MULTIOBJECTIVE OPTIMIZATION OF THE VARIABILITY OF THE HIGH-PERFORMANCE LINPACK SOLVER

Tyler H. Chang

Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, IL 60439, USA

Jeffrey Larson

Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, IL 60439, USA

Layne T. Watson

Depts. of Computer Science, Mathematics, and Aerospace & Ocean Eng.
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061, USA

ABSTRACT

Variability in the execution time of computing tasks can cause load imbalance in high-performance computing (HPC) systems. When configuring system- and application-level parameters, engineers traditionally seek configurations that will maximize the mean computational throughput. In an HPC setting, however, high-throughput configurations that do not account for performance variability could result in poor load balancing. In order to determine the effects of performance variance on computationally expensive numerical simulations, the High-Performance LINPACK solver is optimized by using multiobjective optimization to maximize the mean and minimize the standard deviation of the computational throughput on the High-Performance LINPACK benchmark. We show that specific configurations of the solver can be used to control for variability at a small sacrifice in mean throughput. We also identify configurations that result in a relatively high mean throughput, but also result in a high throughput variability.

1 INTRODUCTION

Performance variability—the fluctuation or "jitter" in the observed performance of a computing system—is a well-known issue in both cloud (Uta et al. 2020) and high-performance computing (HPC) systems (Kramer and Ryan 2003). In practice, these fluctuations need not be normally distributed; they can follow a variety of distributions, including multimodal distributions (Xu et al. 2020). Accounting for such distributions often results in analysis that requires the use of nonparametric statistics (Lux et al. 2018). Performance variability can have many sources, including interference from system- and OS-level processes (De et al. 2008; Petrini et al. 2003), filesystem performance (Cao et al. 2017), and application-level parameters (Hammouda et al. 2015). When left unchecked, performance variability can result in poor load balancing (Dean and Barroso 2013) and a significant degradation in performance (Beckman et al. 2008; Chang et al. 2020). In this paper we investigate the effects of computational throughput variability of an algorithm for solving linear systems of equations on HPC systems.

Previous research has focused on modeling and analyzing I/O throughput variability (Cameron et al. 2019; Maricq et al. 2018) for memory-bound tasks. However, performance variability in compute-bound tasks, such as computationally expensive numerical simulations, is relatively unexplored. One dataset and thorough analyses of performance variability in this context is presented in Patki et al. (2019). Patki et al.

(2019) is actually similar to this paper in that they study the tradeoff between various features, including compute bound, but they just generate a data sample and use network analysis techniques on that dataset a posteriori, instead of attempting to optimize such features.

For a test problems, we consider the High-Performance LINPACK benchmark (HPLB) (Dongarra et al. 2003) problem. The TOP500 list (Strohmaier et al. 2019) defines the HPLB as the problem of solving a dense system of linear equations (of any size) using an LU factorization with partial pivoting. Submissions to the TOP500 list are allowed to present results of solving the HPLB of arbitrary size N and using any linear system solver, as long as the implementation has a floating-point operation count of $\frac{2}{3}N^3 + \mathcal{O}(N^2)$ and solves the problem in 64-bit precision. The software package `HPL` (Petitet et al. 2018) is a portable linear system solver that can utilize massive parallel resources to solve the HPLB. `HPL` uses a block-cyclic data distributing, recursive panel factoring, right-looking variant of the LU-decomposition algorithm.

`HPL` accepts an input file specifying numerous parameters to adjust the underlying algorithm. Properly identifying the parameters that maximize the observed throughput of `HPL` is a notoriously difficult task but has a significant impact on the observed computational throughput (Tan et al. 2009). Traditionally, `HPL` is optimized by experimenting with algorithmic parameters and following the recommendations of Petitet et al. (2018). When submitting performance results to the TOP500 list, researchers may run `HPL` many times with various settings and report only the maximum observed throughput. In previous research, genetic algorithms have been applied to the problem of optimizing `HPL`, treating this task as a black-box optimization problem (Dunlop et al. 2008).

In this paper we investigate the effects of throughput variability on performance when solving a dense linear system using `HPL`, and we model the inherent tradeoff between `HPL`'s mean throughput and throughput standard deviation. The results of this study may not be immediately applicable to the optimization of `HPL` for submissions to the TOP500 because such submissions do not need to account for variability in performance. In fact, the practice of accepting the maximum of all observed overall throughputs may implicitly encourage configurations with high variability. However, the fundamental algorithm that is implemented by `HPL` is similar to others, such as that used by the `PDGESV` driver from ScaLAPACK, which also uses parallel resources to solve a dense linear system of equations (Blackford et al. 1997). Therefore, as an example of a dense linear algebra workload, `HPL` can be used to gain insight into the effects of performance variability on parallel computational linear algebra tasks. A similar problem of interest is the usage of black-box optimization to empirically optimize the basic linear algebra subroutines (BLAS) for maximum performance on a given system (Whaley et al. 2001). However, previous research in tuning linear algebra libraries generally has not investigated the effects of performance variability.

In this paper, `HPL` is optimized on an HPC system by using a black-box *multiobjective* optimization algorithm to minimize throughput standard deviation and maximize the mean throughput simultaneously. This produces approximations to the *Pareto optimal* parameters for `HPL`, which produce throughput distributions along the tradeoff curve between mean throughput and throughput standard deviation. The goals of these experiments are to

- understand the shape of the tradeoff curve between throughput mean and standard deviation when solving large dense linear algebra problems on parallel resources and
- identify parameters for `HPL` that balance this tradeoff.

Two experiments are performed. The first is a single-node analysis, where `HPL` is optimized for a fixed problem size on a single computing node. The second is a four-node study, where `HPL` is optimized on a significantly larger problem size with access to distributed computing resources. The second study requires much more computation time but is a more accurate representation of a large distributed simulation's workload.

Section 2 introduces further details on the configuration parameters for `HPL` and describes the black-box multiobjective optimization software package `VTMOP` that is used in this paper. Section 3 describes the single-node study, including how `HPL` is integrated into `VTMOP` and an analysis of the tradeoff curve

between mean throughput and throughput standard deviation that results from optimizing `HPL` using `VTMOP`. Section 4 describes the second experiment, which is a four-node variation of the experiments from Section 3. Section 5 summarizes our findings and briefly describes directions for future work.

2 BACKGROUND

This section presents further background on the parameters for `HPL` and introduces the multiobjective optimization software package `VTMOP`.

2.1 Tuning HPL

The driver for `HPL` solves a linear system $Az = b$, where $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$. This is done by decomposing $\Pi A = LU$, where L is lower triangular, U is upper triangular, and Π is a permutation matrix reflecting row-interchanges. The vector $y = L^{-1}\Pi b$ is computed during the factorization. Then the system $Uz = y$ is solved by using back-substitution. Before computing the decomposition, the parameter $EQUIL$ specifies whether A will be equilibrated so that its rows and columns have approximately equal magnitudes. Also, $ALIGN$ specifies the byte alignment for double precision numbers. To compute the decomposition, A is decomposed into $NB \times NB$ blocks, which are cyclically distributed over a two-dimensional grid of $P \times Q$ processors. The parameter $PMAP$ specifies whether these blocks are mapped in row- or column-major order.

The main loop of `HPL`'s algorithm works rightward from the leftmost columns of A. In each iteration, a panel of NB columns is factored by recursively dividing each panel into $NDIVS$ subpanels until each subpanel has a size of less than or equal to $NBMIN$. The parameters $RFACT$ and $PFACT$ specify the algorithm variants (right-looking, left-looking, or Crout's method) for recursively dividing each panel and solving each $NBMIN$-sized subpanel, respectively. After a panel has been factored, each processor must broadcast updates, using the broadcast topology $BCAST$. Six broadcast topologies are available, but the modified increasing ring is strongly recommended. The broadcasts are used to update the lower right submatrix. While the broadcast operation is being completed, it is possible to begin updating the next column of the submatrix using a lookahead pipe, whose depth is $DEPTH$. For each update, users have the choice between two update algorithms ($SWAP$): binary exchange or spread-roll. Binary exchange is preferred when the number of columns in the submatrix is small, and spread-roll is preferred when it is large; a mix of the two can be used when a threshold SN for swapping between the two algorithms is provided. Each panel's lower factors $L^{(1)}$ and the final upper factor U can be stored in either transposed or non-transposed format, as specified by $TL^{(1)}$ and TU, respectively. After completing the linear solve, the residual is checked against an error tolerance E to determine whether the HPLB has been solved to an acceptable precision.

The parameters covered in the preceding paragraphs are summarized in Table 1, along with their recommended values/ranges for achieving the maximum overall throughput (Petitet et al. 2018).

2.2 The Multiobjective Optimization Algorithm

Multiobjective optimization problems (MOPs) deal with conflicting objectives, whose tradeoffs must be balanced. In most cases, the solution to a MOP is a multidimensional tradeoff surface, called the Pareto front. In the standard formulation of a MOP, the goal is to minimize p real-valued cost functions $f_i : \mathscr{X} \to \mathbb{R}$, $i = 1, \ldots, p$, where $\mathscr{X} \subset \mathbb{R}^d$ is called the *feasible parameter space*. These cost functions f_i are conceptualized as a single vector-valued cost function $F : \mathbb{R}^d \to \mathbb{R}^p$, where $F(x) = (f_1(x), \ldots, f_p(x))$. The image $\mathscr{Y} = F(\mathscr{X})$ is called the *feasible objective space*.

For two points $X, Y \in \mathscr{Y}$, X dominates Y if X is componentwise less than or equal to Y and strictly less in at least one component. If $F(x^*)$ is nondominated for all $Y \in \mathscr{Y}$, then x^* is efficient, $F(x^*)$ is a point on the Pareto front, and the pair $(x^*, F(x^*))$ is said to be *Pareto optimal*. As $p = 2$ in our case, the

Table 1: Tuning Parameters for `HPL`

Parameter	Meaning	Value(s)	Recommendation
N	problem size	positive integer	80% of RAM
NB	block size	positive integer	$32, 33, \ldots, 256$
$PMAP$	process mapping	row- or col.-major	row-major
P	1st dim. of process grid	positive integer	$P = Q$
Q	2nd dim. of process grid	positive integer	$Q = P$
E	error tolerance	real number	16.0
$PFACT$	LU variant for $NBMIN$-sized panels	right, left, Crout	Crout
$NBMIN$	recursion stopping criterion	positive integer	4 or 8
$NDIVS$	divisions per level of recursion	positive integer	2
$RFACT$	LU variant for panel recursion	right, left, Crout	right
$BCAST$	broadcast topology	6 choices	mod. incr.-ring
$DEPTH$	lookahead pipe depth	nonnegative integer	1
$SWAP$	update algorithm	bin-ex, spread-roll, mix	mix
SN	swapping threshold (for mix)	positive integer	NB
$TL^{(1)}$	transpose $L^{(1)}$	Yes or No	Yes
TU	transpose U	Yes or No	Yes
$EQUIL$	equilibrate A	Yes or No	Yes
$ALIGN$	double alignment in bytes	positive integer	8

Pareto front is a one-dimensional curve. Further reading on MOPs and classical techniques for solving them can be found in the textbook of Ehrgott (2005).

The software package `VTMOP` is a Fortran 2008 implementation of the multiobjective optimization algorithm described by Deshpande et al. (2016). `VTMOP` produces a finite set of points approximating the Pareto front and an efficient set for a computationally expensive black-box MOP subject to lower/upper bound constraints. Tuning `HPL` is a black-box problem because we do not have access to partial derivatives of the objectives with respect to the parameters.

In this paper, we seek to maximize `HPL`'s mean throughput and minimize `HPL`'s throughput standard deviation. Section 3 shows how we account for the fact that many of the parameters that parameterize the problem are integer valued or categorical.

In each iteration of `VTMOP`, the first step is to identify an "isolated point" on the current approximation to the Pareto front. An isolation score is computed by considering the average distance from each known objective value that is currently nondominated to each of its neighboring objective points in a Delaunay graph. The preimage of the most isolated point is used as the center for a local trust region. Next, the package `VTDIRECT95` (He et al. 2009) is used to sample within the current trust region, and this data is used to fit p surrogates using the subroutine `LSHEP` from `SHEPPACK` (Thacker et al. 2010). Several weighted-sum scalarizations are applied to the p `LSHEP` surrogates and minimized by using the algorithm GPSMADS (Audet and Dennis, Jr. 2006). For a large budget, `VTMOP` converges to the true Pareto front if each component of F satisfies a Lipschitz condition.

3 TUNING HPL ON A SINGLE NODE WITH VTMOP

The single-node optimization of `HPL` takes place on an Intel Broadwell node of the HPC system `Bebop` at Argonne National Laboratory. Each Broadwell node is a 36-core Intel Xeon E5-2695v4 processor with 128 GB of DDR4 RAM. The `HPL` executable is built by using the Intel 17.0.4 C compiler and the corresponding Intel Parallel Studio implementation of MPI and Intel Math Kernel Library (MKL) implementation of the BLAS. `VTMOP` is built by using the Intel 17.0.4 Fortran compiler.

3.1 Integrating HPL as a VTMOP Objective Function

Not all of the parameters in Table 1 can be optimized by using VTMOP, and not all of these parameters are desirable to adjust: for example, adjusting the error tolerance E could allow for problems to be solved at undesirable precision. For the single-node study, HPL is optimized for the problem size $N = 10,000$, and the 6 integer-valued parameters NB, P, $NBMIN$, $NDIVS$, $DEPTH$, and SN are adjusted by VTMOP. Note that only one dimension (P) of the process grid is adjusted. We assume that the process grid's dimensions should match the number of available processors (36 for this study), so Q is inferred using $Q = \lceil 36/P \rceil$. The bound constraints for the six adjustable variables are summarized in Table 2. There are over 10^{11} possible combinations of these variables. All other parameters are fixed to the values recommended by Petitet et al. (2018) as shown in Table 2.

Table 2: Bounds for Adjustable Inputs When Tuning HPL

Parameter	Lower Bound	Upper Bound
NB	1	256
P	1	36
$NBMIN$	1	256
$NDIVS$	2	36
$DEPTH$	0	4
SN	1	256

Recall from Section 2 that we seek to minimize the throughput standard deviation and maximize the mean throughput as a function of the parameter configuration x. However, we do not have direct access to these two values. Instead, we can run HPL s times to evaluate the i.i.d. sequence of random variables $T_1(x), \ldots, T_s(x)$, each of which is a copy of the random variable $T(x)$, the observed throughput in Gflops when running HPL with configuration x. Using these values, we can estimate $\mathbb{E}[T(x)]$ and $\sqrt{Var(T(x))}$, the mean throughput and throughput standard deviation, respectively.

For a high-fidelity approximation to the throughput mean and standard deviation, both are computed after a sample of $s = 40$ runs of HPL, following the recommendation used by Cameron et al. (2019) when estimating the I/O throughput variance. On a single Broadwell node of Bebop, the total time for 40 runs of HPL with a problem size of $N = 10,000$ is more than a minute for all parameter configurations considered; certain suboptimal configurations can take much longer. The following steps are taken in order to prevent VTMOP from spending too much time performing 40 sample evaluations of parameters that are clearly suboptimal. For every parameter x, the mean and standard deviation estimates are computed after just $s = 5$ runs of HPL. Let $\hat{\mu}$ and $\hat{\sigma}$ be the early termination thresholds. If the five sample estimates for $\mathbb{E}[T(x)]$ and $\sqrt{Var(T(x))}$ with configuration x satisfy

$$\mathbb{E}[T(x)] < \hat{\mu} \quad \textbf{and} \quad \sqrt{Var(T(x))} > \hat{\sigma}, \tag{1}$$

then the run is aborted, and the five sample estimates are immediately returned.

In order to determine the values of $\hat{\mu}$ and $\hat{\sigma}$ to use in (1), HPL was run 40 times with the recommended settings of $NB = 128$, $P = 6$, $NBMIN = 8$, $NDIVS = 2$, $DEPTH = 1$, and $SN = 128$. The resulting estimates for throughput mean and standard deviation (rounded to three decimal places) are $\mathbb{E}[T(x)] = 613.041$ and $\sqrt{Var(T(x))} = 3.800$, and the early termination thresholds are assigned the arbitrary values $\hat{\mu} = 300$ and $\hat{\sigma} = 8$.

VTMOP can be initialized with a database of precomputed function evaluations. Before starting the single-node study, VTMOP is given an initial database containing the observed mean throughput and throughput standard deviation from running HPL with the recommended parameter evaluation. This inclusion serves as a sanity check since any parameter configurations whose result is dominated by the recommended parameters will not appear in the solution set.

As discussed in Section 2, VTMOP solves MOPs that are real-valued minimization problems. As posed above, HPL is an integer-valued stochastic min/max problem. The following adjustments to VTMOP are made for compatibility.

- The parameter space tolerance (an optional input to VTMOP) is set to 0.99. This prevents VTMOP from evaluating any two points in the parameter space whose Euclidean distance is less than or equal to 0.99 and prevents VTDIRECT95 from dividing any box whose diameter is less than 0.99.
- During the surrogate model optimization phase, the GPSMADS mesh size is restricted to an integer value. This ensures that each batch of candidate parameters values will be spaced on an integer lattice, offset by the position of the current trust region center.
- Before each set of parameters requested by VTMOP is evaluated (by running HPL), all of its components are "binned" by rounding to the nearest integer.
- Tuning HPL is posed as a MOP with the objective $F(x) = \left(-\mathbb{E}[T(x)], \sqrt{Var(T(x))}\right)$.
- All other parameters for VTMOP were given default values.

3.2 Results

Figure 1 shows an approximation to the tradeoff curve between the estimated mean throughput and throughput standard deviation based on 40 runs of HPL on a single node with $N = 10,000$. These results were attained by using VTMOP with a budget of 2,000 evaluations. Note that the scale of the mean throughput is two orders of magnitude larger than the scale of the throughput standard deviation.

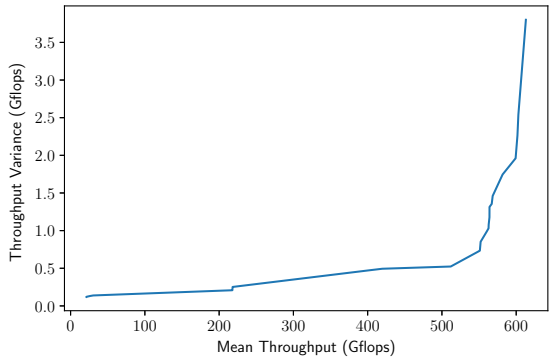

Figure 1: Tradeoff curve between mean throughput and throughput standard deviation in 40 runs of HPL when using the approximate Pareto optimal configurations on a single node with $N = 10,000$.

VTMOP found 24 approximate Pareto optimal configurations, as shown in Figure 1 and whose values are given in Table 3 listed in ascending order by mean throughput/throughput standard deviation. Note that the four configurations in Table 3 with the lowest overall mean/standard deviation have dismal throughputs and are probably not of interest to most readers. Also note that whenever $NBMIN \geq NB$, no recursion takes place, and therefore the variable $NDIVS$ is unused and meaningless.

For the 20 configurations that offer reasonable throughputs, the trends in Table 3 indicate that the "simple" variations of the HPL algorithm (i.e., no lookahead and no recursion) produce significantly lower standard deviations at a steep cost to the mean throughput. On the other hand, using lookahead and allowing for many levels of recursion (i.e., $NBMIN$ and $NDIVS$ small) result in maximal mean throughput, at the cost of increased variability. Each of these trends has several discontinuities, which could indicate that either the underlying Pareto front/efficient set is discontinuous or VTMOP did not fully converge in certain regions of the Pareto front with the given budget. With the exception of the four extremely low throughput settings, all of the approximate Pareto optimal configurations have block sizes that are close to the recommended

Table 3: Approximate Pareto Optimal Set for Single-Node Runs of `HPL` with $N = 10,000$

$\mathbb{E}[T(x)]$	$\sqrt{Var(T(x))}$	*NB*	*P*	*NBMIN*	*NDIVS*	*DEPTH*	*SN*
21.31	0.120	2	19	48	27	3	122
21.35	0.121	2	19	48	27	3	123
25.89	0.131	3	25	47	28	2	122
30.48	0.139	3	20	47	28	3	122
217.82	0.208	129	1	129	19	0	129
218.15	0.252	129	1	256	2	0	1
400.37	0.471	128	1	16	10	0	128
419.73	0.494	129	1	1	2	0	1
511.87	0.523	214	4	15	33	0	72
551.07	0.734	204	4	3	33	0	62
552.12	0.852	204	4	25	35	0	62
560.54	0.991	204	4	15	23	0	185
562.78	1.030	204	4	6	22	0	185
562.84	1.053	204	4	6	33	0	66
562.95	1.080	204	4	6	23	0	182
564.01	1.177	204	4	6	22	0	195
564.06	1.314	204	4	6	22	0	191
567.05	1.355	204	4	9	22	0	191
568.34	1.461	133	3	9	3	2	128
581.69	1.746	128	3	9	3	2	123
599.21	1.961	128	3	4	3	1	128
601.69	2.264	128	3	9	9	1	123
602.93	2.539	128	3	9	6	1	124
613.04	3.800	128	6	8	2	1	128

value of 128. The recommended configuration that was supplied in the initial database is Pareto optimal and achieves the highest mean throughput; it is the last row in Table 3.

In order to further understand the throughput distributions that are shown in Figure 1 and Table 3, the 12 configuration with the highest mean throughputs were used for 100 runs of `HPL`, and the resulting histograms are shown in Figure 2. The distributions are ordered from left to right, top to bottom in ascending order by mean/standard deviation.

Figure 2 shows that each of the throughput distributions has a few outliers that are far **below** the median value. When running numerous numerical simulations in a batch (as described by Chang et al. 2020), the presence of a few "low" outliers could slow the entire computation (Dean and Barroso 2013). For a few of the distributions, there appears to be a cluster of lower throughput values, indicating a multimodal distribution. This mirrors the findings of Xu et al. (2020) for the distribution of I/O throughputs.

The findings in this section provide some insight into parameters for `HPL` that can be used to control the tradeoff between mean throughput and throughput variability. However, since `HPL` is intended for usage in a distributed memory environment, single-node runs are less interesting than multi-node runs. In the Section 4, this study is expanded to the multinode case.

4 TUNING HPL ON MULTIPLE NODES

In this section, `HPL` is optimized for four 36-core Intel Broadwell nodes, each of which is as described in Section 3. Thus, there are 144 total processors available and 512 GB of distributed RAM. In order to ensure that there is enough work for all 144 processors, the problem size considered in this section is increased

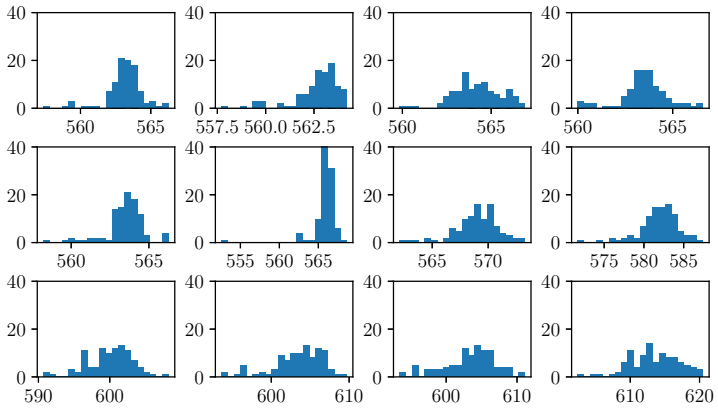

Figure 2: Histograms of observed throughputs when running `HPL` on a single node with $N = 10,000$, for the 12 highest mean throughput configurations. Distributions are ordered from left to right, top to bottom in ascending order by mean/standard deviation.

to $N = 20,000$. The operation count for `HPL` grows cubically with the problem size, so our doubling of N should result in each run of `HPL` requiring roughly eight times as many operations as the runs from Section 3.

The experiment is configured as in Section 3, with the following adjustments.

- Because evaluating the objectives is more expensive, a sample of only $s = 30$ runs of `HPL` is used to compute each configuration's mean and standard deviation, and `VTMOP` is given a budget of only 1,000 evaluations of F (half the budget from Section 3).
- Because the budget for evaluating F has been lowered, the variable NB is eliminated in order to reduce the size of the search space. Based on Petitet et al. (2018) and the results from Section 3, $NB = 128$ producing a five-variable problem with roughly 4×10^8 possible configurations.
- Because the processor count is now 144, the upper bound for P (as previously listed in Table 2) is increased to 144, and Q is inferred by using $Q = \lceil 144/P \rceil$.
- The five-sample thresholds in (1) are $\hat{\mu} = 1000$ and $\hat{\sigma} = 40$, based on 40 runs with the recommended settings of $P = 12$, $NBMIN = 8$, $NDIVS = 2$, $DEPTH = 1$, and $SN = 128$, which produced the estimates $\mathbb{E}[T(x)] = 2236.558$ and $\sqrt{Var(T(x))} = 21.362$. As in Section 3, `VTMOP` is initialized with these precomputed values in its database.

Figure 3 shows an approximation to the tradeoff curve between the estimated mean throughput and throughput standard deviation based on 30 runs of `HPL` on four nodes with $N = 20,000$. The shape of the tradeoff curve in Figure 3 is somewhat similar to the shape of the curve from Figure 1, but is less smooth, and the scale of the mean throughput is three orders of magnitude larger than the scale of the throughput standard deviation.

Again, 24 approximate Pareto optimal configurations are identified, as shown in Figure 3. Their values are given in Table 4, listed in ascending order by mean throughput/throughput standard deviation. Recall that $NBMIN = 128$ for all of these configurations, so only one of the configurations in Table 4 ($P = 8$, $NBMIN = 129$, $NDIVS = 20$, $DEPTH = 0$, $SN = 138$) results in no recursion, and every other configuration results in exactly one level of recursion.

Unlike in Section 3, the recommended configuration of $P = 12$, $NBMIN = 8$, $NDIVS = 2$, $DEPTH = 1$, and $SN = 128$, which produced the observations $\mathbb{E}[T(x)] = 2236.558$ and $\sqrt{Var(T(x))} = 21.362$, is **not** one of the approximate Pareto optimal configurations listed in Table 4. Notice that the recommended configuration's mean throughput is relatively close to the fastest mean throughputs in Table 4, but the

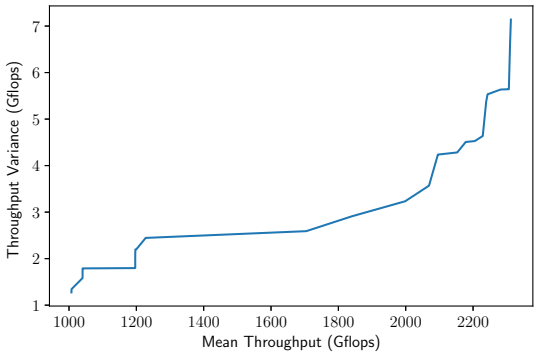

Figure 3: Tradeoff curve between mean throughput and throughput standard deviation in 30 runs of `HPL` when using the approximate Pareto optimal configurations on four nodes with $N = 20,000$.

standard deviation of the recommended configuration is about three times the largest standard deviation in Table 4 and more than four times that of configurations with similar mean throughputs.

The values of $DEPTH$ and SN that produce the highest mean throughputs are identical or very similar to their recommendations. The value of P that produces the highest mean throughput is $P = 9$ (which implies $Q = 16$). Although the recommended setting used in this experiment was $P = 12$, the recommendation of

Table 4: Approximate Pareto Optimal Set for Four-Node Runs of `HPL` with $N = 20,000$, $NB = 128$

$\mathbb{E}[T(x)]$	$\sqrt{Var(T(x))}$	P	$NBMIN$	$NDIVS$	$DEPTH$	SN
1007.1933	1.2741281	3	123	26	3	123
1007.4800	1.3432847	3	123	26	3	118
1040.2800	1.5831875	3	117	31	3	123
1040.3267	1.7903830	3	117	31	3	118
1196.7100	1.7968075	3	117	21	2	123
1196.7167	2.0589725	3	117	21	2	121
1197.0267	2.1971664	3	117	23	2	123
1199.9100	2.2000549	3	116	21	2	123
1227.7600	2.4454885	3	112	31	2	123
1704.5900	2.5906130	8	129	20	0	138
1840.7733	2.9114439	6	117	26	3	121
1998.3433	3.2330922	7	117	26	3	123
2069.1700	3.5682653	9	124	21	3	134
2095.6033	4.2372310	9	114	21	3	134
2153.0533	4.2825011	6	123	21	1	127
2178.2900	4.5079508	6	117	21	1	121
2205.2133	4.5274438	6	112	15	1	121
2228.8300	4.6360767	9	112	21	2	123
2238.9800	5.3728821	9	107	15	2	123
2243.0733	5.5329068	7	114	21	1	123
2281.7467	5.6351565	9	119	23	1	129
2306.3233	5.6427973	9	114	23	1	123
2309.9733	6.6279416	9	107	21	1	123
2312.3500	7.1394364	9	107	26	1	118

Table 5: Additional Evaluations of `HPL` with $N = 20,000$, $NB = 128$

$\mathbb{E}[T(x)]$	$\sqrt{Var(T(x))}$	P	$NBMIN$	$NDIVS$	$DEPTH$	SN
2330.69	15.178	9	4	2	1	128
2319.27	14.011	9	8	2	1	128
2323.22	14.302	9	4	16	1	128
2310.85	13.302	9	8	32	1	128

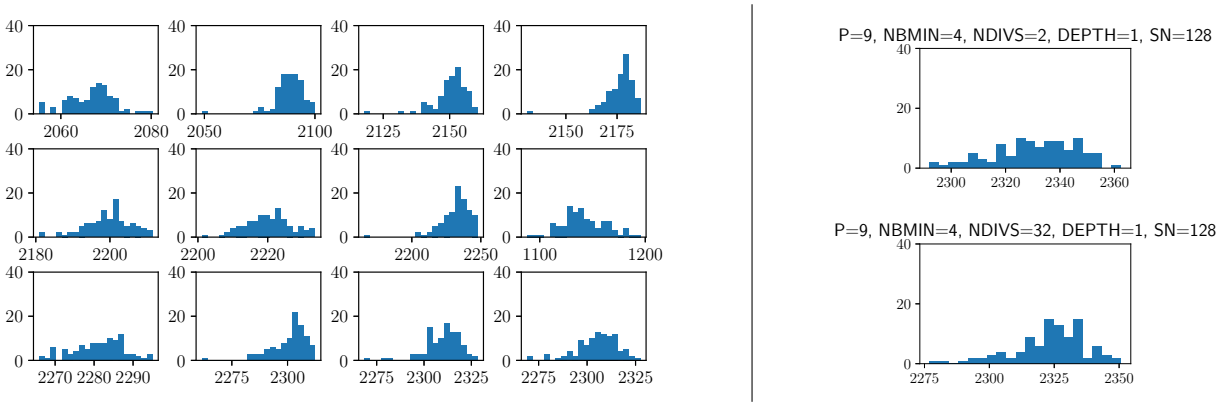

Figure 4: Histograms of observed throughputs when running `HPL` on four nodes with $N = 20,000$, $NB = 128$. The throughput distributions for the 12 highest mean throughput configurations are from Table 4 (left) and the two nondominated configurations are from Table 5 (right).

Petitet et al. (2018) is actually to use either $P = Q$ **or** P slightly less than Q, so $P = 9$ is still in line with this recommendation. However, the values $NBMIN > 100$ and $NDIVS > 20$ seem to strongly contrast with the recommendations of Petitet et al. (2018) that $NBMIN = 4$ or 8 and $NDIVS = 2$. On closer inspection, with a fixed block size of $NB = 128$, using $NDIVS = 21,\ldots,26$ (as in the highest mean configurations in Table 4) results in an effective minimum block size of between 4 and 6 after a single level of recursion. This *partially* agrees with the recommendations of Petitet et al. (2018), who recommend a similar minimum block size of after many more levels of recursion.

Based on these results, one reasonable hypothesis is that the settings of $P = 9$, $NBMIN \in [4,8]$, $DEPTH = 1$, and $SN = 128$ produce the highest mean throughputs, and a large number of divisions per level (resulting in very few levels of recursion before achieving $NBMIN$) produces low variance. In order to check this hypothesis, 4 additional configurations are evaluated with a budget of 100, whose results are given in Table 5. Of the configurations in Table 5, the two configurations with $NB = 4$ are nondominated with respect to other configurations in Table 4, exhibiting significantly higher mean throughputs than the configurations found by `VTMOP` at the cost of a steep increase in the throughput standard deviation. These results somewhat support the hypothesis that the number of levels of recursion is correlated with higher throughput standard deviation. However, the last few entries of Table 4 offer significantly lower standard deviations, which suggest that the configurations $NDIVS = 21,\ldots,26$ and $SN = 118,\ldots,123$ may further affect the mean throughput and throughput standard deviation.

Figure 4 (left) shows histograms of observed throughputs in 100 runs of `HPL` with the 12 configurations from Table 4 with the highest mean throughput, listed from left to right, top to bottom in ascending order by mean/standard deviation. Figure 4 (right) shows histograms of observed throughputs in 100 runs of `HPL` with the two nondominated configurations from Table 5. The maximum throughput distribution (from Table 5) has a flatter shape, associated with hits having higher standard deviation, while other high-throughput configurations are either normally distributed or left-skewed.

5 CONCLUSIONS AND FUTURE WORK

In this paper, the software package `VTMOP` was used to optimize the `HPL` solver for the HPLB.

- The performance variability in `HPL` can be meaningfully controlled by adjusting configuration parameters and sacrificing mean throughput.
- The number of levels of recursion used by `HPL` is a factor that significantly contributes to throughput variability but does not explain all throughput variability.
- Configurations for `HPL` that are nearly optimal for maximizing mean throughput (such as the recommended setting used in Section 4) could result in a throughput standard deviation that is many times higher than the throughput standard deviation for configurations that are Pareto optimal and achieve comparable mean throughput.

Additionally, the techniques that are introduced in this paper can be repurposed or generalized for studying performance tradeoffs in different types of problems. A similar framework could be used to study the tradeoff between mean throughput and throughput variability for I/O bound tasks, for example, by considering the `IOzone` benchmark used by Cameron et al. (2019). It would also be interesting to repeat this study for solving large sparse linear systems, which appear in many real-world numerical simulations (e.g., finite element methods). Furthermore, it would be interesting to investigate the impact of variability when tuning linear algebra libraries. For example, the ScaLAPACK linear system driver `PDGESV` is extremely similar to `HPL`.

ACKNOWLEDGMENT

This work was supported by the U.S. Dept. of Energy (DOE) through the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two DOE organizations, the Office of Science and the National Nuclear Security Administration. This work was also supported by the National Science Foundation under Grant No. CNS-1838271 and by the DOE, Office of Science Graduate Student Research (SCGSR) program. The SCGSR program is administered by the Oak Ridge Institute for Science and Education (ORISE), which is managed by ORAU under contract number DE-SC0014664. All opinions in this paper are the authors' and do not necessarily reflect the policies and views of the DOE, ORAU, or ORISE. The authors gratefully acknowledge the computing resources provided on Bebop, an HPC system operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

REFERENCES

Audet, C., and J. E. Dennis, Jr.. 2006. "Mesh Adaptive Direct Search Algorithms for Constrained Optimization". *SIAM Journal on Optimization* 17(1):188–217.

Beckman, P., K. Iskra, K. Yoshii, S. Coghlan, and A. Nataraj. 2008. "Benchmarking the Effects of Operating System Interference on Extreme-Scale Parallel Machines". *Cluster Computing* 11(1):3–16.

Blackford, L. S., J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, and K. Stanley. 1997. *ScaLAPACK Users' Guide*, Volume 4. Philidelphia, PA: Society for Industrial and Applied Mathematics.

Cameron, K. W., A. Anwar, Y. Cheng, L. Xu, B. Li, U. Ananth, J. Bernard, C. Jearls, T. Lux, Y. Hong, L. T. Watson, and A. R. Butt. 2019. "MOANA: Modeling and Analyzing I/O Variability in Parallel System Experimental Design". *IEEE Transactions on Parallel and Distributed Systems* 30(8):1843–1856.

Cao, Z., V. Tarasov, H. P. Raman, D. Hildebrand, and E. Zadok. 2017. "On the Performance Variation in Modern Storage Stacks". In *Proc. 15th USENIX Conference on File and Storage Technologies (FAST '17)*, 329–344. Santa Clara, CA: USENIX Association.

Chang, T. H., J. Larson, L. T. Watson, and T. C. H. Lux. 2020. "Managing Computationally Expensive Blackbox Multiobjective Optimization Problems with libEnsemble". In *Proc. 2020 Spring Simulation Conference (SpringSim '20)*, Number 31, 1–12. Fairfax, VA: Society for Modeling and Simulation International.

De, P., R. Kothari, and V. Mann. 2008. "A Trace-Driven Emulation Framework to Predict Scalability of Large Clusters in Presence of OS Jitter". In *Proc. 2008 IEEE International Conference on Cluster Computing*, 232–241. Tsukuba, Japan: Institute of Electrical and Electronics Engineers.

Dean, J., and L. A. Barroso. 2013. "The Tail at Scale". *Communications of the ACM* 56(2):74–80.

Deshpande, S., L. T. Watson, and R. A. Canfield. 2016. "Multiobjective Optimization Using an Adaptive Weighting Scheme". *Optimization Methods and Software* 31(1):110–133.

Dongarra, J. J., P. Luszczek, and A. Petitet. 2003. "The LINPACK Benchmark: Past, Present, and Future". *Concurrency and Computation: Practice and Experience* 15(9):803–820.

Dunlop, D., S. Varrette, and P. Bouvry. 2008. "On the Use of a Genetic Algorithm in High Performance Computing Benchmark Tuning". In *Proc. 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 105–113. Edinburgh, UK: Institute of Electrical and Electronics Engineers.

Ehrgott, M. 2005. *Multicriteria Optimization*. 2nd ed. Lecture Notes in Economics and Mathematical Systems Series. Heidelberg, Germany: Springer.

Hammouda, A., A. R. Siegel, and S. F. Siegel. 2015. "Noise-Tolerant Explicit Stencil Computations for Nonuniform Process Execution Rates". *ACM Transactions on Parallel Computing* 2(1):7:1–7:33.

He, J., L. T. Watson, and M. Sosonkina. 2009. "Algorithm 897: VTDIRECT95: Serial and Parallel Codes for the Global Optimization Algorithm DIRECT". *ACM Transactions on Mathematical Software* 36(3):17:1–17:24.

Kramer, W. T., and C. Ryan. 2003. "Performance Variability of Highly Parallel Architectures". In *Proc. International Conference on Computational Science (ICCS 2003)*, 560–569. St. Petersburg, Russia: Springer.

Lux, T. C. H., L. T. Watson, T. H. Chang, J. Bernard, B. Li, X. Yu, L. Xu, G. Back, A. R. Butt, K. W. Cameron, and Y. Hong. 2018. "Nonparametric Distribution Models for Predicting and Managing Computational Performance Variability". In *Proc. IEEE SoutheastCon 2018*, 1–7. St. Petersburg, FL: Institute of Electrical and Electronics Engineers.

Maricq, A., D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci. 2018. "Taming Performance Variability". In *Proc. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 409–425. Carlsbad, CA: USENIX Association.

Patki, T., J. J. Thiagarajan, A. Ayala, and T. Z. Islam. 2019. "Performance Optimality or Reproducibility: That is the Question". In *Proc. The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)*, 1–30. Denver, CO: Association for Computing Machinery.

Petitet, A., R. C. Whaley, J. Dongarra, and A. Cleary. 2018. *HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. Version 2.3. https://www.netlib.org/benchmark/hpl, accessed 5^{th} April, 2020.

Petrini, F., D. J. Kerbyson, and S. Pakin. 2003. "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q". In *Proc. 2003 ACM/IEEE Conference on Supercomputing (SC '03)*, 55–55. Phoenix, AZ: Association for Computing Machinery.

Strohmaier, E., J. Dongarra, H. Simon, and M. Meuer. 2019, November. *The Top 500 List*. https://www.top500.org, accessed 18^{th} April, 2020.

Tan, T. Z., R. S. M. Goh, V. March, and S. See. 2009. "Data Mining Analysis to Validate Performance Tuning Practices for HPL". In *Proc. 2009 IEEE International Conference on Cluster Computing and Workshops*, 1–8. New Orleans, LA: Institute of Electrical and Electronics Engineers.

Thacker, W. I., J. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, and M. W. Berry. 2010. "Algorithm 905: SHEPPACK: Modified Shepard Algorithm for Interpolation of Scattered Multivariate Data". *ACM Transactions on Mathematical Software* 37(3):34:1–34:20.

Uta, A., A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeyer, C. Maltzahn, R. Ricci, and A. Iosup. 2020. "Is Big Data Performance Reproducible in Modern Cloud Networks?". In *Proc. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*, 513–527. Santa Clara, CA: USENIX Association.

Whaley, R. C., A. Petitet, and J. J. Dongarra. 2001. "Automated Empirical Optimizations of Software and the ATLAS Project". *Parallel Computing* 27(1–2):3–35.

Xu, L., Y. Wang, T. Lux, T. Chang, J. Bernard, B. Li, Y. Hong, K. Cameron, and L. Watson. 2020. "Modeling I/O Performance Variability in High-Performance Computing Systems Using Mixture Distributions". *Journal of Parallel and Distributed Computing* 139:87–89.

AUTHOR BIOGRAPHIES

TYLER H. CHANG (Ph.D., Virginia Tech, 2020) is a postdoc studying multiobjective optimization at Argonne National Laboratory. He is interested in numerical analysis, algorithms, and parallel computing. His email is tchang@anl.gov.

JEFFREY LARSON (Ph.D., University of Colorado Denver, 2012) is a computational mathematician at Argonne National Laboratory. He studies algorithms for optimizing computationally expensive functions. His email is jmlarson@anl.gov.

LAYNE T. WATSON (Ph.D., Michigan, 1974) is a professor at Virginia Tech. He has interests in numerical analysis, mathematical programming, bioinformatics, and data science. His email is ltw@cs.vt.edu.