2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) | 979-8-3503-3763-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/PDP59025.2023.00011

An Auto-Tuning Method for High-Bandwidth Low-Latency Approximate Interconnection Networks

Shoichi Hirasawa, Michihiro Koibuchi
Information Systems Architecture Science Research Division
National Institute of Informatics
{hirasawa, koibuchi}@nii.ac.jp

***Abstract*—The next-generation interconnection networks, such as 400 GbE specification, impose Forwarding Error Correction (FEC) operation, such as RS-FEC (544,514), to incoming packets at every switch. The significant FEC latency increases the end-to-end communication latency that degrades the application performance in parallel computers. To resolve the FEC latency problem, a prior work presented error-prone high-bandwidth low-latency networks that do not perform the FEC operation. They enable high-bandwidth approximate data transfer and low-bandwidth perfect data transfer to support various kinds of parallel applications subject to different levels of probability of bit-flip occurrence. As the number of approximate data transfers increases, the parallel applications can obtain a significant speedup of their execution at the expense of the moderate degraded quality of results (QoRs). However, it is difficult for users to identify whether each communication should be approximate or not, so as to obtain the shortest execution time with enough QoRs for a given parallel application. In this study, we apply an auto-tuning framework for approximate interconnection networks; it automatically identifies whether each communication should be approximate data transfer or not, by attempting thousands executions of a given parallel application. An auto-tuning attempts a large number of program executions by varying the possible communication parameters to find out the best execution configuration of the program. The multiple executions would generate different positions of bit flips on communication data that may provide different qualities of results even if the same parameters are taken. Although this uncertainty introduces difficulties in the optimization of the auto-tuning, many offline trials lead to a high probability of the program's success execution. Evaluation results show that high-performance MPI applications with our auto-tuning method result in 1.30 average performance improvement on error-prone high-performance approximate networks.**

***Keywords*-Auto tuning, interconnection network, approximate computing, bit flips.**

I. Introduction

Existing interconnection networks usually have error detection and correction mechanisms, such as the re-transmission with cyclic redundancy check (CRC), to achieve high-reliable communication on parallel computers. While existing optical interconnection networks use low-level modulation, i.e., on/off keying, the next-generation interconnection networks, such as the use of 800Gbps-specification link, will use multi-level modulation for high-bandwidth communication [1]. The multi-level modulation, such as Pulse Amplitude Modulation (PAM) and Quadrature Amplitude Modulation (QAM), occurs frequent bit flips with non-negligible probability due to its smaller design margin. There is a strong need for better error detection and correction ability in the next-generation interconnection networks, such as Forward Error Correction (FEC).

There are various kinds of FEC algorithms, hard decision (HD) block codes, hard decision concatenated codes, and soft-decision iterative codes, at the expense of largely buffering transferred data [2]. It is reported that 25 Gbps high-reliability high-end optical interconnects require at least 84 ns for storing 2,112-bit encoding blocks used in Base-R FEC computation [3], thus expecting about 100 ns FEC overhead at every switch. Although the 25 Gigabit Ethernet Consortium releases Low-latency FEC specifications for 50, 100, and 200 Gb/s Ethernet, it cuts the FEC latency only in half.

The IEEE 400 Gb/s Ethernet Study Group estimates up to 100 ns latency for $4\times$RS–FEC on 10×40 Gbps lanes [4]. Since the current switch delay ranges from 40 ns [5] to 100 ns in InfiniBand QDR, the per-hop delay will increase to 3.5-times (40 ns to 140 ns) due to FEC computation.

To resolve the FEC latency problem, a prior work proposed error-prone high-bandwidth, low-latency approximate interconnection networks that do not use the FEC operation [6]. The approximate networks follow the term "approximate computing" that exploits the gap between the level of accuracy by the applications and that provided by the computing system, for performance gains [7]. The approximate networks provide not only high-bandwidth approximate data transfer but also low-bandwidth perfect data transfer to support various kinds of parallel applications. Each parallel application is required to decide whether each communication between processes should be approximate or not. It is reported that the approximate network leads to significant performance improvements to applications, with speedup up to 2.94 when compared to the use of conventional networks [6].

The main drawback of the approximate network is the low productivity of parallel programs. It is difficult to find a decision to obtain the shortest execution time with enough QoRs to users for a given parallel application that does not consider the probability of bit flips.

In this study, we propose to apply an auto-tuning framework for approximate interconnection networks for the high productivity of parallel programs. We use OpenTuner framework that builds domain-specific multi-objective program autotuners [8] to automatically identifies whether every communication can be approximate data transfer or not, for a given parallel application. It executes a given parallel application thousands of times by varying the possible communication parameters for finding the best trade-off between execution time and QoR for a parallel application. Users are then free from the above massive tuning operation.

The multiple executions would generate different positions of bit flips on communication data that may provide different qualities of results even if the same parameters are taken. This uncertainty difficulties exist in the optimization of the auto-tuning method. In the auto-tuning framework, a large number of offline trials lead to a high probability of the success execution of the program.

Prior work manually finds out the best configuration of the communication data of CG and FT on an approximate network. However, it is too costly. The straight-forward application of OpenTuner to the approximate network cannot find the best configuration. This lower quality of the tuning would degrade the performance of the parallel applications on the approximate network. We resolve this problem of the auto-tuning method by concreting the reliable region to each communication parameter.

More specifically, our main contributions to this work are:

- Our auto tuning method with confidence-interval technique successfully finds out the best configuration of approximate communication for CG and FT parallel applications in case studies.
- In our auto-tuning method, the configuration of the 95% of confidence interval with 2000 times trials usually leads to a better execution time of a target parallel application.
- Using discrete-event simulation, we tuned high-performance MPI applications taken from NAS Parallel Benchmarks on an approximate network. We then result in 1.30 average performance improvement for the production run.

The background information and related work are discussed in Section II. The approximate network is explained in Section III, and the qualitative discussion of the auto-tuning method is shown in Section VI. Section VII concludes with a summary of our findings.

II. Background and Related Work

A. Network Error Detection and Correction

We consider a conventional network of electric packet switches interconnect via active optical cables (AOCs) as our baseline. The bandwidth of cost-effective commodity AOCs is currently 100 or 200 Gbps. As bandwidth increases in the near future, the reliability concern raises.

Since 400 GbE standards use multi-level modulation, noise immunity becomes terrible that is recovered by RS-FEC. The RS(544,514) and RS(528,514) are considered in 400 Gb/s Ethernet Task Force Group [9] [10]. The RS(528,514) requires approximately 100 ns overhead due to its 5280-bit buffer [9]. Various FEC schemes have already been proposed for pulse-amplitude modulation (PAM) signals in data center applications [11]. We expect that the FEC operation overhead becomes about 100 ns in a next-generation conventional network, and it is surprisingly the same as the switch delay. Since parallel computers typically are latency-sensitive, the FEC operation overhead cannot be accepted in interconnection networks.

B. Parallel Applications on Unreliable Interconnection Networks

Bit flips can be tolerated via a standard checkpoint approach. A drawback of a checkpoint approach is the performance loss due to checkpointing overhead. Another approach is approximate computing. For given applications, it is possible to relax correctness for hardware components so as to improve performance and power consumption. Such approximate components have been considered mainly for processors and storages [7]. However, to the best of our knowledge, approximate networks have been considered only for wireless local area networks for H.264 data transfers [12] for consumer applications on the Internet [13]. The prior work considered the case study of a Hamming code that can not provide rigid error-free communication [14]. Another prior work is the design of approximate networks that provide both high-bandwidth approximate and low-bandwidth perfect data transfers for highend datacenters or HPC systems [6]. In this study, we succeed the prior work [6] for the high productivity of the parallel application for approximate networks.

C. Auto Tuning for Parallel-Computer Systems

A modern parallel-computer platform has already become too complicated to predict its performance for a given code accurately. It is difficult for a compiler to judge whether each code optimization improves the entire performance or not. Therefore, the empirical performance tuning by evaluating the actual performance of each kernel variant on the target platform is inevitable, especially in HPC systems.

As manual tuning is tedious and time-consuming to generate and evaluate multiple kernel variants, automatic empirical performance tuning, so-called *auto-tuning*, has been intensively studied for various libraries and applications, including BLAS libraries [15], FFT libraries [16][17], and stencil applications [18][19].

One computational kernel can be implemented in various ways due to the varieties of performance tuning parameters. Then, the user replaces the computational kernel with one of its kernel variants and evaluates the performance.

The target code is modified in various ways, and each of the modified versions of the target code is a kernel variant. The performance of every code variant is empirically evaluated so as to select a high-performance code variant. This performance tuning process is called *empirical performance tuning* [20].

The performances of kernel variants are evaluated one by one in a try-and-error fashion until a high-performance kernel variant is found.

III. Approximate Networks

Prior work proposed approximate networks that consist of electric switches using optical cables [6]. Approximate networks require updating (1) bit mappings on the multi-level modulation for optical communications, and (2) parallel applications for supporting error-prone communications. The auto-tuning framework identifies whether each communication is approximate or perfect while updating a program. This makes programmers overhead small. Each key element of the approx network approach is explained hereafter.

A. Error-Prone Multi-Level Modulation

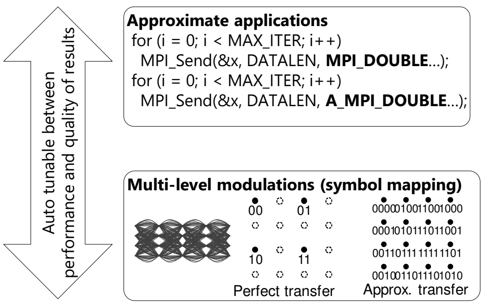
Conventional short-reach optical communication used on-off-keying (OOK) as digital modulation. To achieve up to 4× link bandwidth improvements, approx networks use M-QAM (or M-PAM) ($4 \leq M \leq 16$) as digital modulation for each optical link. When using 16-QAM (or 16-PAM), 4-bit per symbol can be transferred, while OOK transfer 1-bit per symbol.

M-QAM makes the margin to a neighboring symbol small, which degrades bit error rate (BERs). To reduce the effect of symbol errors at modulation, the conversion between signal and bit sequence uses the Gray code method to have a Hamming distance of 1 bit between neighbors.

The previous study illustrated that a device that can manipulate M-QAM obtains better BER by limiting the symbols and retaining more distance between symbols in the additive white Gaussian noise model (AWGN) [6]. It was reported that assuming nominal BER 10^{-5}, our calculations showed that M/4 symbol mapping leads to a BER of $1.52{\times}10^{-16}$, while the bandwidth was reduced by a factor 2 for M=4, respectively. Combining these mapping policies, we can dynamically adjust BER in accordance with bit priorities determined by bit positions in packets. For example, approximate data should be sent with M=4, while routing and flow control information should be M=2 in which the BER becomes the same as that for existing conservative interconnection networks.

B. Error-Prone Parallel Applications

Approximate networks are designed assuming that bit flips can occur with non-negligible probability. Target applications for these networks include approximate computing applications that can tolerate some bit flips and still produce acceptable results. In typical scientific computing applications, for instance, floating-point numbers encoded according to IEEE standards are being communicated. The IEEE 754 standard for floating-point numbers consists of a sign, exponent, and fraction bits. A bit flip in the sign and the exponent likely has a high impact on the numerical value of the floating-point number. Instead, a bit flip in the fraction bit, particularly in its least significant bits, has a smaller impact on the numerical value. We proposed a "symbol-mapping" method to map bit sequences to numerical values in a way that reduces the numerical impact of bit flips in [6]. This method essentially enforces different BERs for different parts of the communicated data so as to manage the knob between expected error on numerical values and network performance.



```
for (i = 0; i < MAX_ITER; i++)
  MPI_Send(&x, DATALEN, MPI_DOUBLE...);
for (i = 0; i < MAX_ITER; i++)
  MPI_Send(&x, DATALEN, A_MPI_DOUBLE...);
```

Fig. 1. Auto tuning framework on Approximate Networks.

IV. Auto Tuning between Performance and Accuracy in Approximate Networks

We target parallel programs that correctly worked in an existing perfect parallel compute system. In approximate networks, each communication can be either protected (perfect) or non-protected (approximate) by changing the symbol mapping at multi-level modulation. As explained in the previous section, non-protected communication is much faster than protected communication. Thereby, an application can be much faster. However, an execution with error-prone communications would produce wrong application results. Accordingly, we present an auto-tuning for the whole parameter space, including both bit mask of each numerical data expressed by IEEE 754 format transferred in approximate networks, as illustrated in Figure 1.

A. Condition

The auto-tuning needs to verify the final execution results of an application with a specific parameter set in addition to the execution performance. General correctness of the final execution results of an application cannot be often defined. However, it is able to verify when a verification code is provided or the user of the application can verify them. For example, it is difficult to completely resolve the NP-hard problem, such as Traveling Salesman Problem (TSP). However, the approximate solution obtained by the approximate algorithm can be quickly verified.

In this study, for the sake of simplicity, it is assumed that a target application already has an execution script with input data. Therefore, the application can be executed using the scripts with the proposed auto-tuning method for the performance evaluation and the result verification. When an execution with a specific parameter set turns out to be a wrong execution, the method can detect the wrong execution and keep away from selecting the parameter set for further tuning.

B. Search Space of Auto Tuning

Parallel applications need to communicate with each other for parallel execution. Existing applications assume accurate communications for exact execution. As a result, it is difficult to correctly know what communication is not necessarily accurate, and how many bit errors are tolerable for the application to keep its execution. To fully utilize approximate network's performance, an auto-tuning method is used to search what commutation is needed to be accurate and how many bits of numerical transferred data are tolerable for the application. In approximate networks, the number of protected bits of a numerical transferred data is managed as a parameter, and the proposed auto-tuning method searches the parameter.

There is a trade-off between the performance and BER in an approximate network. When a lower BER is needed, a more accurate error correction method is needed and therefore the performance of the network degrades. On the other hand, when a higher BER is able to be used for an application, a higher performance of the network can be used. As this trade off is not able to determine statically, the auto-tuning method is used to search the desired point.

Because these parameters are related to each other, it is not effective to search for the best point of each parameter independently. A correct final execution result can only be achieved with a set of error tolerable communication and BER, in which the execution performance of the application is determined with the corresponding parameters of accurate commutation and BER. As a result, all parameters are included in the searching space of the auto-tuning to find the best parameter set.

C. Dynamic Optimization Search with Confidence Interval

The auto tuning in this work needs to verify the final execution results of an application with a specific parameter set. While general correctness of the final execution results of an application is not able to define, it is able to verify when a verification code is provided or the user of the application can verify them.

In this work, it is assumed that a target application already has an execution script with input data. Therefore, using the scripts, the application can be executed with the proposed auto tuning method for performance evaluation and result verification.

When an execution with a specific parameter set turns out to be a wrong execution, the search method can detect the wrong execution and keep away from selecting the parameter set for further tuning. Parameters can be the length of protecting bits of an IEEE floating-point number in communication data.

It is necessary to accurately compare trial results of different parameter sets to accurately optimize each parameter set in auto tuning for approximate networks. Thus, an infinite number of trial results are better to accurately achieve an accurate statistical population because the performance fluctuation occurs by a number of different reasons of the complex computer systems. For example, the position of a bit flip on IEEE 754 floating point number strongly impacts on the computational results for a given parameter set.

However, it is difficult to complete an infinite number of trial results for a specific parameter set within a realistic time scale. Instead, a specimen average is acquired with a number of trials for the dynamic optimization search in this study. An interval estimation is used to predict the statistical population by the specimen average. The statistical population and the specimen average are presumed to follow a normal distribution.

A confidence interval is the possibility where a statistical average is included in the interval of the specimen average and the variance.

The width of the confidence interval affects the whole trial time of the optimization because an interval estimation is used to estimate the confidence interval. The width of the confidence interval should be carefully decided according to the trade-off between the length of overall trial time and the optimized performance. We investigate the impact of the confidence interval on the application performance in the next section.

V. Evaluation

In this section, we use discrete-event simulation to evaluate the performance of parallel applications and benchmarks obtained by the auto-tuning method when executed on approx networks and on conventional networks. Specifically, we use the SimGrid simulation framework (v3.12) [21]. SimGrid implements validated simulation models, is scalable, and makes it possible to simulate the execution of unmodified parallel applications that use the Message Passing Interface (MPI) [22]. We implemented the bit-error model of optical links with AWGN and our proposed bit protection mechanisms within the SimGrid simulation models.

A. Methodology

1) Network Configuration: We compare a conventional interconnection network without FEC (Conv. NW), and an approximate network (Approx. NW). Link bandwidth is set to 400 Gbps in the conventional interconnection network, while the approximate network has up to 8 Tbps. A BER of 10^{-8} has been set for the approximate 8 Tbps links. A conventional 400 Gbps network without FEC could be optimistic. However, we assume that FEC-free network can be designed for up to 400 Gbps.

We use torus network topology. For the topology, we use a shortest path routing scheme using Dijkstra's algorithm. Each switch has a 51 nsec delay. Each host has eight cores for an overall computation speed of 1 TFlops. We configure SimGrid to utilize its built-in version of the MVAPICH2 of MPI collective communications [23].

2) Data Protection Mechanisms: We implemented an interface that allows the software to specify the relative importance of bits in data types used in MPI communication [6]. The network handles transactions with perfect data integrity for standard MPI data types, as specified by the data type argument passed on MPI send functions. We also define *Approximate*

Data Types that can be used by the MPI application developer. The relative importance of bits in these data types is defined as error masks by the developer ahead of time, making it possible to protect certain bits via the reliable communication methods described above. In this study, we assume two levels of bit importance, i.e., protected and unprotected.

Note that routing and flow control information is protected within the same framework we established for payload data. Therefore, the BER of such information becomes the same as that for non-approximate networks.

3) Implementation of Auto-Tuning Method: The auto-tuning method is implemented on top of OpenTuner [8] framework. It finds the best parameter set that results in the best execution performance of a target application. The auto-tuning process is an offline tuning process, which terminates before the actual production run of each target application. During the tuning process, the target application is executed with a specific parameter set, and the execution performance is saved. The tuning framework then starts other execution instances of the target application. After a series of trial runs, the framework terminates the tuning process and returns the best execution performance of the target application and the corresponding parameter set.

4) Parallel Applications used in Evaluation: We simulate the execution of the applications of the NAS Parallel Benchmarks (version 3.3.1, MPI versions) [24], Class A, and Graph 500 benchmark, Himeno benchmark, and a k-means clustering application. Table I gives salient characteristics of our used applications.

BT: Block Tri-diagonal solver is a program to test the performance of output capabilities of high-performance parallel computing system.

CG: Conjugate Gradient (CG) is one of the popular application that solves a linear system $A.x = b$, where A is a symmetric positive-definite matrix. We use the Fortran implementation in the NPB benchmark suite, and run it on 64 and 256 processors. The metric for result quality is the average mean error compared to the result produced by an error-free execution.

DT: Data Traffic is a benchmark to test unstructured computations and parallel data movement among parallel computer systems.

FT: Fast Fourier Transform (FT) is a popular numerical application that is well-suited to an approximate computing approach. The metric for result quality is the average mean error compared with the result produced by an error-free execution.

IS: Integer Sort is a benchmark to test the performance of random memory accesses, with a large integer sort which is important in particle method applications.

LU: Lower-Upper Gauss-Seidel parallel solver for nonlinear partial partial differential equations.

MG: Multi Grid is a simple 3-D multigrid benchmark. The iterations of a multigrid algorithm are used to obtain an approximate solution to the discrete Poisson problem.

SP: Scalar Pentadiagonal parallel solver for nonlinear partial partial differential equations.

MM: Matrix Multiplication example provided in the SimGrid distribution.

Graph: Graph 500 reference implementation application version 2.1.4.

Himeno: Himeno benchmark, which calculates fluid dynamic analysis.

k-means: k-means clustering vector quantization application.

B. The Confidence of the Auto Tuning

We use 10^{-8} as the error rate of the approximate network and 0.05 as the bandwidth factor for the tuning parameters to evaluate the tuning process of MPI applications.

Figure 2 shows the percentage of obtaining the manual best parameter sets of CG and FT, which are used in [6]. We can see the method with 95% confidence interval comparison represents higher percentages in finding the manual best parameter sets compared to the conventional OpenTuner searching and the method with 99% confidence interval. Therefore, the auto-tuning method with the 95% confidence interval is used hereafter for the performance evaluations.

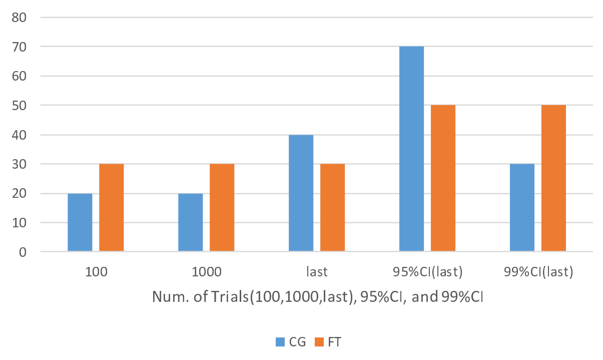


Fig. 2. The Percentage of Successfully Obtaining the Manual Best on CG and FT.

C. Simulation Results

1) Statics: Table I shows the applications that we used for our evaluation. We used nine NPB applications, Graph 500 application, Himeno benchmark application, and k-means clustering applications. In our evaluation, an approximate data type is specified for each application. A bit mask is defined corresponding to the approximate data type of all applications. Each MPI call is re-compiled for both a conventional call and an approximated call, corresponding to the parameter set of a specific trial. As a result, a total searching space can be defined for each application. (From 10^{16} to 10^{40} for our evaluation)

2) Application Execution Time: We measure application execution times and determine the impact of network characteristics (switch delay, link bandwidth, and network topology). Note that we define the execution time as the wall-clock

TABLE I
AUTO-TUNING SEARCHING SPACE OF APPLICATIONS USED IN THE EVALUATION. [1] SEARCHING SPACE OF BITS TO MASK FOR PROTECTION ON EXCHANGED MESSAGES [2] SEARCHING SPACE OF MPI SEND/RECV STATIC CALLS IN THE APPLICATION SOURCE CODES

Application	Description	Approximate Data Type	MASK[1]	MPIs[2]	Total
BT	NPB BT	`MPI_INTEGER`	2^{64}	2^{36}	10^{24}
CG	NPB CG(Conjugate Gradient)	`MPI_DOUBLE`	2^{64}	2^{10}	10^{18}
DT	NPB DT	`MPI_INTEGER`	2^{64}	2^{19}	10^{21}
FT	NPB FT(FFT)	`MPI_DOUBLE_COMPLEX`	2^{128}	2^{12}	10^{40}
IS	NPB IS	`MPI_INTEGER`	2^{64}	2^{11}	10^{18}
LU	NPB LU	`MPI_INTEGER`	2^{64}	2^{41}	10^{27}
MG	NPB MG	`MPI_DOUBLE`	2^{64}	2^{12}	10^{18}
SP	NPB SP	`MPI_INTEGER`	2^{64}	2^{24}	10^{23}
MM	Matrix Multiply	`MPI_DOUBLE`	2^{64}	2^{10}	10^{18}
Graph	Graph 500	`MPI_INTEGER`	2^{64}	2^{29}	10^{25}
Himeno	Himeno Benchmark	`MPI_DOUBLE`	2^{64}	2^{6}	10^{16}
k-means	k-means clustering	`MPI_DOUBLE`	2^{64}	2^{8}	10^{17}

time between the start of the execution until its completion (which thus encompasses all computation and communication operations).

Figures 3 and 4 show execution times, normalized to that on the conventional network (Conv.), for our approximate computing applications using a Torus topology with 64 nodes and 256 nodes, respectively. Values above 1 denote improvement over the conventional network. Note that a speedup of s for the overall execution time of a parallel application implies a communication speedup greater than s. While the approx network (Approx) uses bit protection mechanisms to limit data corruption, the fully-approximated network (Full-approx) exposes all communicated bits to a high BER.

Figure 3 shows results for our approximate computing applications, executed with 64 MPI ranks. The most notable result is that the approx network reduces execution time significantly compared to the conventional network. For FT, the approximate network with appropriate bit protection achieves a speedup of 1.53 compared to the conventional network, and it is 1.33 for MG. For CG, only 2% of the transferred messages allow soft errors to occurs, and yet the execution time on the approx network is close to that of the conventional network. The average of the application's speedup is 1.30, which implies that using our approach even for only a small portion of the communication payload can alleviate the communication bottleneck and lead to significant performance improvements. Moreover, an 1.18 speedup on average can be achieved with smaller computation system, "Tuned-by-4nodes" in the figure, shows that the proposed auto-tuning method is able to achieve higher tuned performance with much smaller computation system compared to the larger production system.

Figure 4 shows results for our approximate computing applications, executed with 256 MPI ranks. Notes that the performance improvement ratio of Graph is 1.05, which is smaller than the case of 64 MPI ranks. The performance improvement ratio of MM is 0.94, which is much smaller than that of 64 MPI ranks, while CG still do not show much performance improvement. The average performance improvement ratio is 1.19, which is smaller than that of 64 MPI ranks. This fact implies that with more network overhead, our approximate network with the auto-tuning method leads to more performance improvements.

VI. DISCUSSION

A. Auto Tuning on Approximate Networks vs. Custom Tuning on Perfect Networks

Recently, 4, 8, and 16-bit numerical-number expressions are frequently used to implement approximate computing, such as custom deep neural-network accelerators for obtaining high throughput at the expense of computational accuracy on FPGAs [25]. This approach with short numerical-number expressions compresses the data itself, and it assumes to use perfect interconnection networks.

By contrast, we assume to use a rigid data format, such as IEEE 754 standard for floating-point arithmetic, used in the standard parallel programming that may cause a gap between the accuracy required by the applications and that provided by the computing system. Thus, our approach increases link bandwidth by allowing "error-prone" data transfer. In this context, our approach is radically different from the custom tuning on perfect networks. Since we assume to use the existing programing model, operating system, hardware, as a baseline, the productivity of the parallel application on approximate networks is not terrible. Their comprehensive comparison is impressive, and it is our future work.

B. Burst Bit Error

The prior study illustrated the influence of burst bit errors on the performance of approximate networks [14]. In the Gilbert-Elliot burst-error model [26], a system is modeled with two states, i.e., the good and bad states, in which the possibility of a bit-flip depends on the state of the previous bit. It is reported the success execution rate of the NAS parallel benchmarks in AWGN is similar to that in Gilbert-Elliot burst-error cases in [14]. In this study, we show the case for AWGN model. However, the auto-tuning is expected to work well under the burst bit-flip model in approximate networks.

C. Uncertainty

In a lucky case, an attempt to execute a parallel application includes no bit flips even when BER is terrible may result in a successful execution. However, the execution with the best

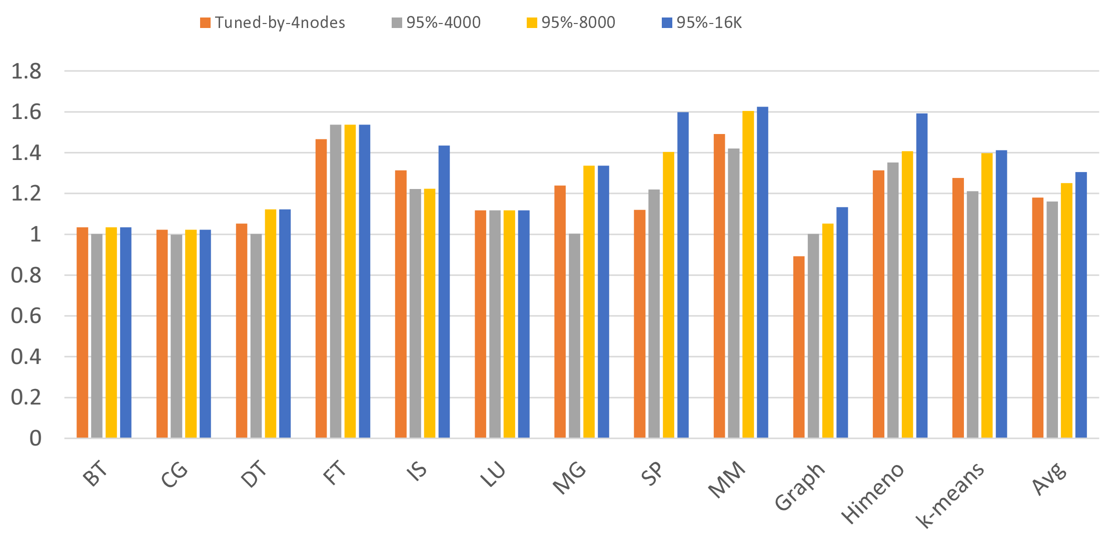


Fig. 3. Relative performance improvement with the tuning results for parallel apps with 64 MPI ranks

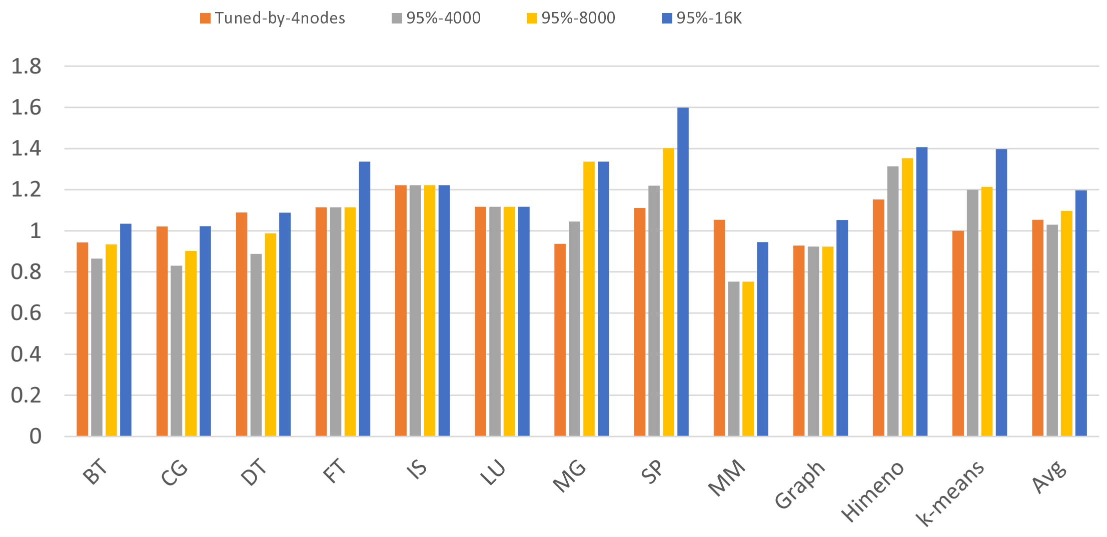


Fig. 4. Relative performance improvement with the tuning results for parallel apps with 256 MPI ranks

parameter set found with the auto-tuning process is expected to succeed in the execution in a higher probability because of the large number of the offline trial runs during the tuning process with confidence intervals.

D. System Size vs Tuning Result

From our evaluation results, a tuning result from a smaller computation system is still able to achieve higher performance results on a larger, usually a production, system. This means that we can use the proposed method in a simple smaller testing computation system to achieve a desired improved execution performance, while executing the tuned applications in larger production systems, which are usually limitted by temporal and/or financial budgets.

VII. Conclusions

Computational applications are subject to various kinds of numerical errors due to bit flips. In this context, the prior work proposed high-bandwidth, low-latency approximate networks. The approximate networks enact different trade-offs between performance and bit error rates (BERs); they support both approximate high-bandwidth and perfect low-bandwidth data transfers.

In this study, we apply an auto-tuning framework to an approximate interconnection network. It automatically identifies whether each communication should be approximate data transfer or not, for a given parallel application.

Using discrete-event simulation, we tuned high-performance MPI applications on the high-performance approximate networks, resulting in 1.30 average performance improvements for the production run.

As the process of our tuning process is entirely autonomous on an existing programming environment, the tedious manual programming and optimization process is unnecessary. Moreover, even if an application user hardly understands its program code, our proposed auto-tuning method can correctly work. This is because our proposed auto-tuning only needs to verify the final execution results of an application with a specific parameter set in addition to the execution performance.

We conclude that our proposed auto-tuning method is inevitable for utilizing high-performance approximate networks.

Acknowledgment

This paper is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] IEEE 802.3 Ethernet working group, http://www.ieee802.org/3/.

[2] G. Tzimpragos, C. Kachris, I. B. Djordjevic, M. Cvijetic, D. Soudris, and I. Tomkos, "A Survey on FEC Codes for 100G and Beyond Optical Networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 209–221, Firstquarter 2016.

[3] M. Andrewartha, B. Booth, and C. Roth, "Feasibility and Rationale for 3m no-FEC server and switch DAC," http://www.ieee802.org/3/by/public/Sept15/andrewartha_3by_01a_0915.pdf, Sep. 2015.

[4] IEEE P802.3bs 400Gbps Ethernet Study Group, "400GbE PCS Architectural Options," http://www.ieee802.org/3/400GSG/public/13_07/gustlin_400_02_0713.pdf, 2013.

[5] B. Towles, J. P. Grossman, B. Greskamp, and D. E. Shaw, "Unifying on-chip and inter-node switching within the Anton 2 network ," in *Proc. of the ACM/IEEE International Symposium on Computer Architecture, (ISCA)*, June. 2014, pp. 1–12.

[6] D. Fujiki, K. Ishii, I. Fujiwara, H. Matsutani, H. Amano, H. Casanova, and M. Koibuchi, "High-bandwidth low-latency approximate interconnection networks," in *Proc. of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb 2017, pp. 469–480.

[7] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[8] OpenTuner, An extensible framework for program autotuning, http://opentuner.org/.

[9] IEEE P802.3bj 100 Gb/s Backplane and Copper Cable Task Force, "FEC Options," http://www.ieee802.org/3/bj/public/jan12/gustlin_01_0112.pdf.

[10] IEEE P802.3bs 200 Gb/s and 400 Gb/s Ethernet Task Force, http://www.ieee802.org/3/bs/, Dec 2017.

[11] M. N. Sakib and O. Liboiron-Ladouceur, "A Study of Error Correction Codes for PAM Signals in Data Center Applications," *IEEE Photonics Technology Letters*, vol. 25, no. 23, pp. 2274–2277, Dec 2013.

[12] S. Sen, S. Gilani, S. Srinath, S. S. hmitt, and S. Banerjee, "Design and implementation of an "approximate" communication syste m for wireless media applications," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2010, pp. 15–26.

[13] B. Ransford and L. Ceze, "SAP: an architecture for selectively approximate wireless commu nication," *CoRR*, vol. abs/1510.03955, 2015.

[14] T. N. Truong, H. Matsutani, and M. Koibuchi, "Low-reliable low-latency networks optimized for HPC parallel applications," in *IEEE International Symposium on Network Computing and Applications, NCA*, 2018, pp. 1–10.

[15] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," *Parallel Computing*, vol. 27, pp. 3 – 35, 2001.

[16] M. Frigo, "A fast fourier transform compiler," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI. ACM, 1999, pp. 169–180. [Online]. Available: http://doi.acm.org/10.1145/301618.301661

[17] A. Nukada and S. Matsuoka, "Auto-tuning 3-D FFT library for CUDA GPUs," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, Nov 2009, pp. 1–10.

[18] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," in *High Performance Computing, Networking, Storage and Analysis, International Conference for*, Nov 2008, pp. 1–12.

[19] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams, "An auto-tuning framework for parallel multicore stencil computations," in *Parallel Distributed Processing (IPDPS), IEEE International Symposium on*, April 2010, pp. 1–12.

[20] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *Proceedings of the International Conference on Parallel Architectures and Compilation (PACT)*. New York, NY, USA: ACM, 2014, pp. 303–316.

[21] SimGrid: Versatile Simulation of Distributed Systems, http://simgrid.gforge.inria.fr/.

[22] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Appl ications and Platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[23] "MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE." [Online]. Available: http://mvapich.cse.ohio-state.edu/

[24] The NAS Parallel Benchmarks. [Online]. Available: http://www.nas.nasa.gov/Software/NPB/

[25] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of low numeric precision deep learning inference using intel® fpgas," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 73–80.

[26] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell Labs Technical Journal*, vol. 39, no. 5, pp. 1253–1265, 1960.