Reproducibility in Benchmarking Parallel Fast Fourier Transform based Applications

Samar Aseeri
samar.aseeri@kaust.edu.sa
Extreme Computing Research Center, King Abdullah University of Science and Technology
Thuwal, Kingdom of Saudi Arabia

Benson K. Muite
benson.muite@ut.ee
Arvutiteaduse Instituut, Tartu Ülikool
Tartu, Estonia

Daisuke Takahashi
daisuke@cs.tsukuba.ac.jp
Center for Computational Sciences, University of Tsukuba
Tsukuba, Japan

ABSTRACT

An overview of concerns observed in allowing for reproducibility in parallel applications that heavily depend on the three dimensional distributed memory fast Fourier transform are summarized. Suggestions for reproducibility categories for benchmark results are given.

CCS CONCEPTS

• **Mathematics of computing → Computation of transforms**; • **Theory of computation → Massively parallel algorithms**; • **Software and its engineering → Software performance**; • **Hardware →** *Testing with distributed and parallel systems.*

KEYWORDS

Fast Fourier Transform, Benchmarks, Reproducibility

ACM Reference Format:
Samar Aseeri, Benson K. Muite, and Daisuke Takahashi. 2019. Reproducibility in Benchmarking Parallel Fast Fourier Transform based Applications. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3302541.3313105

1 INTRODUCTION

The fast Fourier transform (FFT) is used in a large number of applications. It is difficult to make it work well on distributed memory parallel computers because of communication that requires a high bandwidth low latency network. There are many different implementations of the FFT, and several suggested benchmarking strategies. This note describes the procedure used in benchmarking the Klein Gordon equation on thirteen different computers[2] and benchmarking FFT based programs on a further four additional supercomputers. The main contributions are suggestions for concrete reproducibility classes.

2 BACKGROUND

Reproducibility here is taken to mean either reproducible computational results and/or reproducible execution time measurements. A number of reproducibility initiatives have been proposed[15, 18, 30, 31]. These range from, a general algorithm description, making the code available, making an input deck and execution environment available, to documenting the entire workflow. In many parallel programs, bitwise reproducibility of the computational results (in particular with floating point arithmetic) is challenging because even the same program run on the same computer twice can have different orders of execution of arithmetic operations. Furthermore, simulations done on supercomputers may use supercomputing resources others cannot access to check results. Finally, on many computers, the network is a shared resource that can affect other running programs. These all make full bitwise reproducibility challenging, and in many cases do not detract from the scientific results which may still be obtained to within reasonable accuracy. Nevertheless, there are still applications such as cryptography, dynamical systems and number theory where bitwise reproducibility is required, though it may be possible to use integer arithmetic which alleviates some of the reproducibility challenges created by floating point arithmetic.

3 METHOD

In the original study[2], an open source Fourier pseudospectral code was used in a benchmark for the Klein Gordon equation[25]. The code uses MPI and is primarily written in Fortran. It makes heavy use of the library 2DECOMP&FFT[20, 21] to perform the three dimensional distributed memory fast Fourier transform. The user has a choice of underlying one dimensional fast Fourier transform engines, some of which are closed source vendor provided libraries. The library 2DECOMP&FFT uses auto tuning to choose the best domain decomposition for the distributed memory FFT at runtime. In performing the study, on each supercomputer used in the study, program output and the one dimensional FFT engine used were recorded. In addition, makefiles which were used for compilation of the programs were kept. Unfortunately, the exact system configuration (name and version of operating system, version of one dimensional FFT engine used in 2DECOMP&FFT, and if open source and compiled by the user or the supercomputing center, their configuration and compilation options) were not recorded. Upon publication, the exact source code version of the program used in the study was made available on arXiv[2], with a very similar program available on Github[26].

Further work has examined the performance of this code and a Navier Stokes solver[8, 9] on the K computer, Hazelhen (a Cray XC 40 supercomputer), Kabuki (an NEC SX-ACE supercomputer) and Shaheen II (a Cray XC 40 supercomputer). Changing compilation options and the one dimensional FFT engine used can lead to a factor of 10 difference in performance, without hand tuned program optimization. Kabuki is an NEC SX-ACE computer, where porting of 2DECOMP&FFT is required to use the appropriate optimized MathKeisan one dimensional FFT library. Both the K computer and NEC SX-ACE do not use x86 processors, which makes using packaging and autobuilding systems challenging. They do have support for C, C++ and Fortran compilers, which on a computer without many community code developers are the most important tools to allow for porting of existing high performance programs.

4 LESSONS LEARNED

The SPEC benchmarking process[14] makes build reproducibility significantly better than what was done in this study[2]. Unfortunately, it does not allow for tuning and code modifications that are typically required on new computer architectures.

Repeating experiments multiple times has been suggested as a means of verifying reproducibility in benchmarking[15]. Such a methodology has been implemented in gearshifft, a heterogeneous fast Fourier transform benchmark suite[30, 36]. In most cases experiments were repeated several times, usually successively, with minor differences between results. In cases where experiments have been repeated on the same machine, between several hours or several months apart or with a fresh installation, more significant differences have been found. These can be due to upgrades of system software, changes in job placement policy or due to a different amount of interference by other jobs on the system. Explaining and documenting all of these may be rather difficult - in particular in cases where a benchmark is run by a user interested in the application rather than a computer scientist or system administrator[5, 37]. Figure 1 demonstrates runtime variability when running fast Fourier Transform programs[8, 9] for solving the Navier-Stokes equations on Hazelhen, Kabuki and Shaheen II. Both Hazelhen and Shaheen II use adaptive routing and job placement to speed up job throughput[17], but this can result in some run time variability, a subject of current research[6, 29, 35, 38, 39]. Kabuki has a cross bar network, where job placement and job interference effects can also be important.

Performance tuning can also result in code that is not portable between systems, but is essential in obtaining results that depend on the highest available computational performance[10]. In such cases, a performance model, a clear description of the implementation and a clear and correct reference code with which to compare results may be more helpful in allowing for reproducibility.

Most scientific computing software does not automatically include error bounds for floating point calculations, for example using interval analysis[23], and error bounds for the approximate numerical methods utilized. Usually such bounds are very pessimistic, hence are not reported. More work is needed in this area.

Automatic build systems can make it easier to reproduce results[1, 4, 12–14]. Such systems are also helpful in benchmarking when the end user will not do software performance optimizations for the computer system they will run on. However, for most parallel benchmarks it seems likely that hand optimization by an end benchmarker will be required to obtain the highest performance on the computer system of interest.

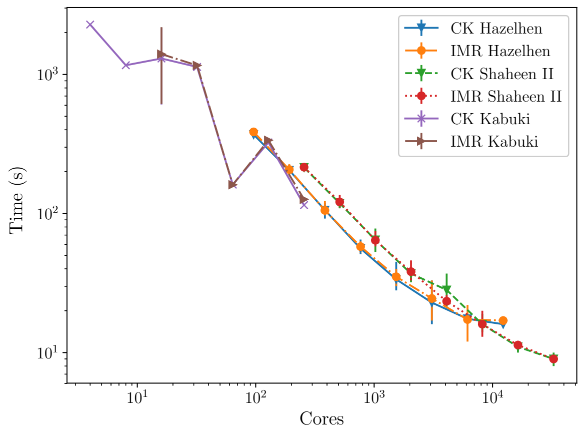

Figure 1: Performance variability for 20 timesteps of 0.005 of a 512^3 discretization on Kabuki, Hazelhen and Shaheen II when using Carpenter-Kennedy (abbreviated to CK in the legend)[9] and implicit midpoint rule timestepping (abbreviated to IMR in the legend)[8]. The error bars represent fastest and slowest execution times. On Hazelhen and Shaheen II, FFTW3[11] was used as the one dimensional FFT engine in 2DECOMP&FFT[20, 21], while on Kabuki MathKeisan FFT was used as the one dimensional FFT engine in 2DECOMP&FFT. Compilation and execution scripts are available on request. Data is available at [3].

On new hardware or on systems without a large number of users, only a limited amount of software will have been or will be ported to the system. On a system such as Kabuki, the build system Cmake[16, 22] requires significant effort to port. Cmake is an essential component of gearshifft, hence rather than using Cmake and gearshifft, it was easier to re-write the one dimensional FFT speed test in FFTE 6.0[32–34] to produce the results in Fig. 2. Benchmarks which do not allow for flexibility in re-writing code will stifle innovation by making it difficult to compare new approaches with established ones.

Figure 2 demonstrates that for small FFTs, Kabuki is less performant than either Hazelhen or Shaheen II, but for FFTs above a size of 512 points, Kabuki is significantly more performant than either Hazelhen or Shaheen II. Vector computers and graphics processing units typically have higher memory bandwidth but also higher latency than typical CPUs, thus similar results for one dimensional FFT performance are also reported in [30], where for small FFTs, CPU performance is best, but for large FFTs, GPUs are better.

There are many parallel scientific computing programs that have modeling assumptions built into them (for example in computational fluid mechanics, materials science and chemistry). The typical user may not be able to check all these modeling assumptions, and even if the code is open source, checking the code in general (let alone the executable produced to run on the users system) can be quite time consuming. Comparisons of computational results obtained for the same input data (ideally using different methods on different computational resources), can be very helpful in giving confidence in the reported results[19, 28].

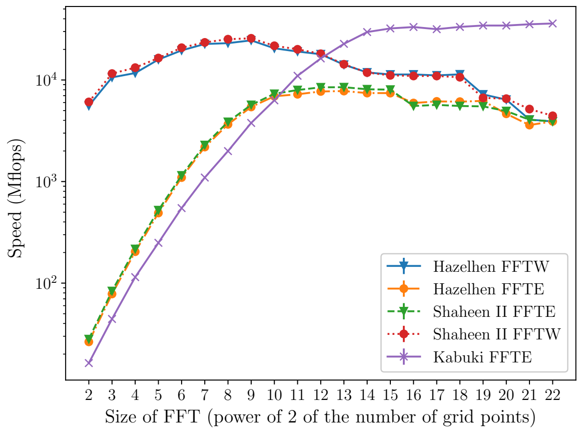

Figure 2: Performance of 1D FFT on a single core of Kabuki, Hazelhen and Shaheen II using the 1D speed test in FFTE, with either FFTE 6.0[32–34] or FFTW3[11]. Error bars representing the standard deviation in execution times are smaller than the points. Each test was repeated at least twice. Compilation and execution scripts are available on request. Data is available at [3].

Finally, the choice of programming language is also a key concern. Programming languages with a long lifetime, mature compilers and stable feature set such as C and Fortran make portable reproducibility on a wide variety of computers easier. Many current computers however do not have architectures that closely map to either C or Fortran[7].

Open source code is usually compiled and run on closed source hardware with closed source compilers and closed source runtimes. Thus, in some cases, users can only verify that the output is correct, but might not be able to determine possible causes of errors.

5 SUGGESTIONS

Rather than a single reproducibility standard, reproducibility classes should be used. The minimum amount of information for reproducibility of a computation is a verifiable input and a verifiable output. In the case of computer performance benchmarks, timing measurements for program execution are also relevant. Further characterization includes:

a) Closed source code
b) Open source code (optimized or reference) on closed source hardware
c) Open source code (optimized or reference) on open source hardware

Error characterizations include:

a) Bitwise reproducible
b) Numerical error bounds
c) Statistically reproducible

Workflow characterizations are also very helpful in enabling reproducibility. Due to the wide variations in supercomputer environments, a workflow specification on one computational platform will not be portable to all other relevant choices of computational platforms, but may be useful for similar computational platforms. Thus a workflow characterization should include

a) particular platforms for which it is suitable
b) whether the workflow execution contains elements outside the users control that may give different results on each execution due to other factors (for example runtime autotuning, shared network and input/output resources, etc)

Specifying such information would allow for appropriate documentation by researchers when doing their work and better allow reviewers to evaluate the work. This would allow the use of specialized hardware (such as the Anton or MDM molecular dynamics supercomputers[24, 27]), or using portable software that has been performance optimized. Commercial requirements imply that not all technology (software or hardware), will be available as open source, and many scientists will likely use some closed source component in their workflow. Compile and run, install and run or just run (for scripts or pre-installed software) may form a large part of the computing that is being done today, in particular in the tail end of high performance computing, but it is unlikely to be the only model in high performance computing. A means of describing reproducibility requirements for the range of situations encountered in high performance computing would be very helpful. Further elaboration on these reproducibility classes will be done while exploring performance of the FFT and applications which utilize the FFT.

ACKNOWLEDGMENTS

We thank all the authors of [2] and those who have given a presentation on their use of the FFT in the ongoing discussion at www.fft.report. We also thank Robert Henschel for an overview of the SPEC benchmarking process at the benchmarking in the data center workshop at HPC Asia 2019. We thank RIKEN for the use of the K computer, HLRS for the use of Kabuki and Hazelhen, and the KAUST Supercomputing Laboratory for the use of Shaheen II. B.K.M. was partially supported by HPC Europa 3 (INFRAIA-2016-1-730897). B.K.M. thanks H. Berger, A. Chepstov, J. Gracia and A. Jocksch for helpful hints and discussions.

REFERENCES

[1] D. Alvarez, A. O'Cais, M. Geimer, and K. Hoste. 2016. Scientific Software Management in Real Life: Deployment of Easybuild on a Large Scale System. In *Proceedings of the Third International Workshop on HPC User Support Tools (HUST '16)*. IEEE Press, Piscataway, NJ, USA, 31–40. https://doi.org/10.1109/HUST.2016.8
[2] S. Aseeri, O. Batrasev, M. Icardi, B. Leu, A. Liu, N. Li, B.K. Muite, E. Müller, B. Palen, M. Quell, H. Servat, P. Sheth, R. Speck, M. Van Moer, and J. Vienne. 2015. Solving the Klein-Gordon Equation Using Fourier Spectral Methods: A Benchmark Test for Computer Performance. In *Proceedings of the Symposium on High Performance Computing (HPC '15)*. Society for Computer Simulation International, San Diego, CA, USA, 182–191. http://dl.acm.org/citation.cfm?id=2872599.2872622;https://arxiv.org/abs/1501.04552
[3] S. Aseeri, B.K. Muite, and D. Takahashi. 2019. Data for figures in "Reproducibility in Benchmarking Parallel Fast Fourier Transform based Applications". https://doi.org/10.5281/zenodo.2571988
[4] G. Becker, P. Scheibel, M. LeGendre, and T. Gamblin. 2016. Managing Combinatorial Software Installations with Spack. In *Proceedings of the Third International Workshop on HPC User Support Tools (HUST '16)*. IEEE Press, Piscataway, NJ, USA, 14–23. https://doi.org/10.1109/HUST.2016.6
[5] A. Bhatele, K. Mohror, S.H. Langer, and K.E. Isaacs. 2013. There Goes the Neighborhood: Performance Degradation Due to Nearby Jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. ACM, New York, NY, USA, Article 41, 12 pages. https://doi.org/10.1145/2503210.2503247

[6] I. Chinavinijkul, J. Newcomb, L. Xi, and D.P. Bunde. 2018. Brief Announcement: Coloring-based Task Mapping for Dragonfly Systems. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA ’18)*. ACM, New York, NY, USA, 91–93. https://doi.org/10.1145/3210377.3210665

[7] D. Chisnall. 2018. C Is Not a Low-level Language. *Queue* 16, 2, Article 10 (April 2018), 13 pages. https://doi.org/10.1145/3212477.3212479

[8] B. Cloutier. 2014. MPI Fortran Carpenter-Kennedy Incompressible Navier Stokes Solver. (2014). https://github.com/bcloutier/PSNM/blob/master/NavierStokes/Programs/NavierStokes3dFortranMPI/navierstokes.f90.

[9] B. Cloutier. 2014. ‘MPI Fortran Implicit Midpoint Rule Incompressible Navier Stokes Solver. (2014). https://github.com/bcloutier/PSNM/blob/master/NavierStokes/Programs/NavierStokes3dFortranMPI/navierstokes_IMR.f90.

[10] M. Eleftheriou, B.G. Fitch, A. Rayshubskiy, T.J.C. Ward, P. Heidelberger, and R.S. Germain. 2008. A Study of the Effects of Machine Geometry and Mapping on Distributed Transpose Performance. In *Proceedings of the 5th Conference on Computing Frontiers (CF ’08)*. ACM, New York, NY, USA, 79–86. https://doi.org/10.1145/1366230.1366243

[11] M. Frigo and S. G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (Feb 2005), 216–231. https://doi.org/10.1109/JPROC.2004.840301

[12] T. Gamblin, M. LeGendre, M.R. Collette, G.L. Lee, A. Moody, B.R. de Supinski, and S. Futral. 2015. The Spack Package Manager: Bringing Order to HPC Software Chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’15)*. ACM, New York, NY, USA, Article 40, 12 pages. https://doi.org/10.1145/2807591.2807623

[13] M. Geimer, K. Hoste, and R. McLay. 2014. Modern Scientific Software Management Using EasyBuild and Lmod. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST ’14)*. IEEE Press, Piscataway, NJ, USA, 41–51. https://doi.org/10.1109/HUST.2014.8

[14] Standard Performance Evaluation Corporation High Performance Group. 2011. SPEC MPI2007 Run and Reporting Rules. (August 2011).

[15] T. Hoefler and R. Belli. 2015. Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses when Reporting Performance Results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’15)*. ACM, New York, NY, USA, Article 73, 12 pages. https://doi.org/10.1145/2807591.2807644

[16] B. Hoffman and K. Martin. 2003. The CMake Build Manager. *Dr. Dobbs Journal* (Jan. 2003). http://www.drdobbs.com/cpp/the-cmake-build-manager/184405251#

[17] N. Jain, A. Bhatele, X. Ni, N.J. Wright, and L.V. Kale. 2014. Maximizing Throughput on a Dragonfly Network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’14)*. IEEE Press, Piscataway, NJ, USA, 336–347. https://doi.org/10.1109/SC.2014.33

[18] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. 2017. The popper convention: Making reproducible systems evaluation practical. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 1561–1570.

[19] A.M. Jokisaari, P.W. Voorhees, J.E. Guyer, J. Warren, and O.G. Heinonen. 2017. Benchmark problems for numerical implementations of phase field models. *Computational Materials Science* 126 (2017), 139 – 151. https://doi.org/10.1016/j.commatsci.2016.09.022

[20] N. Li. 2019. 2DECOMP&FFT. http://www.2decomp.org/

[21] N. Li and S. Laizet. 2010. 2DECOMP&FFT – A highly scalable 2D decomposition library and FFT interface. In *Proc. Cray User Group 2010 conference*.

[22] K. Martin and B. Hoffman. 2007. An Open Source Approach to Developing Software in a Small Organization. *IEEE Software* 24, 1 (Jan 2007), 46–53. https://doi.org/10.1109/MS.2007.5

[23] R. Moore, R. Kearfott, and M. Cloud. 2009. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9780898717716

[24] T. Narumi, R. Susukita, T. Koishi, K. Yasuoka, H. Furusawa, A. Kawai, and T. Ebisuzaki. 2000. 1.34 Tflops Molecular Dynamics Simulation for NaCl with a Special-purpose Computer: MDM. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (SC ’00)*. IEEE Computer Society, Washington, DC, USA, Article 54. http://dl.acm.org/citation.cfm?id=370049.370465

[25] D. San Roman Alerigi, S. Balakrishanan, A.H. Bargash, G. Chen, B. Cloutier, N. Li, D. Malicke, B.K. Muite, M. Quell, P. Rigge, M. Solimani, A. Souza, A.S. Thiban, J. West, and M. Van Moer. 2018. Parallel Spectral Numerical Methods. (November 2018). http://en.wikibooks.org/w/index.php?title=Parallel_Spectral_Numerical_Methods.

[26] D. San Roman Alerigi, S. Balakrishanan, A.H. Bargash, G. Chen, B. Cloutier, N. Li, D. Malicke, B.K. Muite, M. Quell, P. Rigge, M. Solimani, A. Souza, A.S. Thiban, J. West, and M. Van Moer. 2019. Parallel Spectral Numerical Methods. (January 2019). https://github.com/openmichigan/PSNM

[27] D.E. Shaw, J.P. Grossman, J.A. Bank, B. Batson, J.A. Butts, J.C. Chao, M.M. Deneroff, R.O. Dror, A. Even, C.H. Fenton, A. Forte, J. Gagliardo, G. Gill, B. Greskamp, C.R. Ho, D.J. Ierardi, L. Iserovich, J.S. Kuskin, R.H. Larson, T. Layman, L.-S. Lee, A.K. Lerer, C. Li, D. Killebrew, K.M. Mackenzie, S.Y.-H. Mok, M.A. Moraes, R. Mueller, L.J. Nociolo, J.L. Peticolas, T. Quan, D. Ramot, J.K. Salmon, D.P. Scarpazza, U. Ben Schafer, N. Siddique, C.W. Snyder, J. Spengler, P.T.P. Tang, M. Theobald, H. Toma, B. Towles, B. Vitale, S.C. Wang, and C. Young. 2014. Anton 2: Raising the Bar for Performance and Programmability in a Special-purpose Molecular Dynamics Supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’14)*. IEEE Press, Piscataway, NJ, USA, 41–53. https://doi.org/10.1109/SC.2014.9

[28] M.R. Shirts, C. Klein, J.M. Swails, J. Yin, M.K. Gilson, D.L. Mobley, D.A. Case, and E.D. Zhong. 2017. Lessons learned from comparing molecular dynamics engines on the SAMPL5 dataset. *Journal of Computer-Aided Molecular Design* 31, 1 (01 Jan 2017), 147–161. https://doi.org/10.1007/s10822-016-9977-1

[29] S. Sreepathi, E. D’Azevedo, B. Philip, and P. Worley. 2016. Communication Characterization and Optimization of Applications Using Topology-Aware Task Mapping on Large Supercomputers. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE ’16)*. ACM, New York, NY, USA, 225–236. https://doi.org/10.1145/2851553.2851575

[30] P. Steinbach and M. Werner. 2017. gearshifft – The FFT Benchmark Suite for Heterogeneous Platforms. In *High Performance Computing*, J.M. Kunkel, R. Yokota, P. Balaji, and D. Keyes (Eds.). Springer International Publishing, Cham, 199–216.

[31] V. Stodden, C. Hurlin, and C. PÃľrignon. 2012. RunMyCode.org: A novel dissemination and collaboration platform for executing published computational results. In *2012 IEEE 8th International Conference on E-Science*. 1–8. https://doi.org/10.1109/eScience.2012.6404455

[32] D. Takahashi. 2001. A Blocking Algorithm for FFT on Cache-Based Processors. In *High-Performance Computing and Networking*, B. Hertzberger, A. Hoekstra, and R. Williams (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 551–554.

[33] D. Takahashi. 2003. A parallel 1-D FFT algorithm for the Hitachi SR8000. *Parallel Comput.* 29, 6 (2003), 679 – 690. https://doi.org/10.1016/S0167-8191(03)00039-5

[34] D. Takahashi. 2014. FFTE: A Fast Fourier Transform Package. http://ffte.jp/

[35] X. Wang, M. Mubarak, X. Yang, R. B. Ross, and Z. Lan. 2018. Trade-Off Study of Localizing Communication and Balancing Network Traffic on a Dragonfly System. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1113–1122. https://doi.org/10.1109/IPDPS.2018.00120

[36] M. Werner and P. Steinbach. 2019. mpicbg-scicomp/gearshifft: v0.4.0. https://doi.org/10.5281/zenodo.2555649

[37] L. Wu, X. Xu, Y. Wei, and X. Liu. 2017. A Survey About Quantitative Measurement of Performance Variability in High Performance Computers. In *Advanced Parallel Processing Technologies*, Y. Dou, H. Lin, G. Sun, J. Wu, D. Heras, and L. Bougé (Eds.). Springer International Publishing, Cham, 76–86.

[38] P. Yébenes, J. Escudero-Sahuquillo, P.J. García, and F.J. Quiles. 2016. Straightforward solutions to reduce HoL blocking in different Dragonfly fully-connected interconnection patterns. *The Journal of Supercomputing* 72, 12 (01 Dec 2016), 4497–4519. https://doi.org/10.1007/s11227-016-1756-1

[39] Y. Zhang, O. Tuncer, F. Kaplan, K. Olcoz, V. J. Leung, and A. K. Coskun. 2018. Level-Spread: A New Job Allocation Policy for Dragonfly Networks. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1123–1132. https://doi.org/10.1109/IPDPS.2018.00121