Impact of Varying BLAS Precision on DCMESH

Nariman Piroozan
Intel Corporation
Santa Clara, CA, USA
nariman.piroozan@intel.com

S. John Pennycook
Intel Corporation
Santa Clara, CA, USA
john.pennycook@intel.com

Taufeq Mohammed Razakh
University of Southern California
Los Angeles, CA, USA
razakh@usc.edu

Peter Caday
Intel Corporation
Hillsboro, OR, USA
peter.caday@intel.com

Nalini Kumar
Intel Corporation
Santa Clara, CA, USA
nalini.kumar@intel.com

Aiichiro Nakano
University of Southern California
Los Angeles, CA, USA
anakano@usc.edu

***Abstract*—The limiting factor in the application of high-accuracy quantum molecular simulations to large systems has been the associated high computational costs in terms of both compute power and memory. In this paper we explore the use of various BLAS precision modes (BF16, TF32, and Complex_3M) in DCMESH (divide-and-conquer Maxwell-Ehrenfest-surface hopping), a framework utilized for the study of light-matter interaction. On a single stack of the Intel® Data Center GPU Max Series 1550, we are able to achieve a speedup of 1.35x while retaining accuracy in key output parameters such as the number of excited electrons, current density, and kinetic energy. For large problem sizes, we observe speed-ups of up to 3.91x for individual BLAS calls. Switching between BLAS precision modes requires no source code changes (only environment variables), and so the approach we demonstrate here could be readily applied to other High Performance Computing (HPC) workloads that spend a significant amount of time in BLAS calls.**

***Index Terms*—precision, cgemm, blas, performance modeling**

I. Introduction

The United States Department of Energy procured 1.8 billion dollars in 2018 for the delivery of three exascale high performance computing (HPC) clusters representing an exponential increase in memory, storage, and compute power. Concurrent with this was the accompanying effort for software modernization needed in order to be able to harness thousands of GPUs. Furthermore, exascale computing has also encouraged increased effort to be placed on converting traditional software frameworks to alternative precisions of FP32 or lower.

Alternative precision types such as BF16, TF32, or FP8 have become important in recent years with the explosion of Artificial Intelligence (AI) codebases [1]. For example, large language models (LLMs) such as Llama3 [2] or GPT-4 [3] have emerged as widely used generative AI models that are trained on a very large number of GPUs and then utilized for inference purposes. The training element of LLMs which can contain up to hundreds of billions of parameters produce enormous demands on computing systems in terms of both compute power and memory. If either FP64 or FP32 were to be used for LLMs of this size, training and inference would become impractical [4]. As a result, BF16 and FP8 precisions are used to partially alleviate these demands. The resultant precision studies can see less than 5% accuracy loss for some models [5]. In this paper, we explore the application of alternative precision methods to some aspects of traditional HPC applications.

One of the pressing issues within the domain of materials modeling, and one that is fundamental to the nature of computing in general, is the trade-off between accuracy and performance. For the molecular dynamics (MD) community, this is particularly important when we consider extremely large simulations. In addition, the selection of which precision to utilize in an MD study is contingent upon the nature of the problem being studied. For example, in simple cases where atomic structure is determined based on a straightforward set of equations, single precision is likely to be sufficient [6]. However, for systems containing higher order calculations there is a higher probability of round-off errors, and so it may be advisable to utilize mixed or double precision [7]. In the case of mixed precision, all important state vectors such as particle coordinates, velocities and forces are computed and stored in single precision, and other critical variables such as the virial are stored in double precision, striking a balance between improved time to solution, reduced memory footprint, and simulation accuracy.

The key contributions of this paper are:

- We introduce the various BLAS precision models (aka "alternative compute modes") available in the Intel® Math Kernel Library (Intel® oneMKL) and the expected performance improvement offered by each.
- We measure the impact of different BLAS precision modes upon the accuracy of DCMESH as represented by the number of excited electrons, current density, and kinetic energy.
- We demonstrate the impact of each BLAS precision mode upon the performance of DCMESH when run on Intel® Data Center GPU Max Series hardware.

II. Related Work

A. Quantum Dynamical Frameworks

While DCMESH introduces certain novel features (see Section II-C), there are a few other frameworks which are well-

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00187

established and widely used: Quantum Espresso (QE) [8], the Vienna Ab-Initio Simulation Package (VASP) [9] and QMCPACK [10].

Quantum Espresso implements plane wave density functional theory in conjunction with a wide variety of pseudopotentials in order to allow for the nanoscale modeling of materials. While similar to VASP in its general function, Quantum Espresso is open source and used within the MD community for a wide variety of functions. While VASP is not open source, it offers a somewhat broader assortment of pseudopotentials which in turn enable a broader set of material studies.

QMCPACK is a highly performant electronic structure framework which implements a wide variety of Quantum Monte Carlo (QMC) algorithms. By virtue of directly solving the Schrödinger equation, QMC methods offer much greater accuracy than traditional algorithms such as density functional theory (as found in both QE and VASP). While this comes at much greater computational cost, certain applications such as electronic structure calculations require it [10].

All these frameworks support GPUs, but at the time of writing only QMCPACK supports the Intel® Data Center GPU Max Series – ports of VASP and QE are underway. QMCPACK supports both mixed precision and FP64, while VASP and QE only support FP64.

B. Reduced and Mixed Precision

Many historical explorations of FP32 were driven by large differences in theoretical throughput on early Nvidia GPUs [11]. This transition to lower precision from traditional FP64 draws parallels to the efforts underway today with the transition to FP16, BF16, FP8 and below. With LLMs, high memory usage is traditionally the bottleneck. This has been a major driving force in transitioning to FP8 and lower for inference as a means to greatly reduce memory demands. Furthermore, FP8 compute is often twice that of BF16 so even compute bound workloads, such as inference with LLMs, on very large batch sizes can benefit [1].

The utilization of lower precision for molecular dynamics simulations has been of interest for some time. LAMMPS [7], for example, contains a series of accelerator packages that support specific pair styles which allow for the use of alternative algorithms, precision modes, and data layouts in order to achieve optimum performance on specific hardware. The INTEL and GPU packages both include build options to support single, mixed, and double floating point precision modes. GROMACS [6] can also be compiled in either mixed or double precision.

A new field within molecular dynamics is that of using AI accelerated potential models trained using data derived from Density Functional Theory (DFT) to be used in conjunction with classical MD frameworks such as LAMMPS or GROMACS to perform the scientific simulations. Two such models are DeePMD [12], which can use FP32 or BF16 [13], and Allegro [14], which uses FP32.

TABLE I
THEORETICAL PEAK THROUGHPUT FOR A SINGLE STACK.

Precision	Theoretical Peak	Engines
FP64	26 TFLOP/s	Vector
FP32	26 TFLOP/s	Vector
TF32	209 TFLOP/s	Matrix
BF16	419 TFLOP/s	Matrix
FP16	419 TFLOP/s	Matrix
INT8	839 TOP/s	Matrix

C. DCMESH Overview

DCMESH [15] is a novel MD framework used to perform quantum molecular studies of light-matter interaction. The most unique characteristic of DCMESH is its implementation of a globally-sparse and locally-dense electronic solver with multiple time-scale splitting, and shadow dynamics in order to achieve high scalability. DCMESH consists of a Local Field Dynamics (LFD) portion to describe light-electron interaction, and a Quantum Excitation Molecular Dynamics (QXMD) portion to describe electron-atom coupling. In the latest implementation, LFD runs on the GPU and QXMD runs on the CPU, and CPU-GPU data transfers are minimized through the use of shadow dynamics.

III. Background

A. Intel® Data Center GPU Max Series

The Intel® Data Center GPU Max Series 1550 is based on the micro-architecture previously code named "Ponte Vecchio" [16]. Each GPU consists of two stacks (or tiles), and each stack consists of multiple Xe cores. Each Xe core features eight 512-bit vector engines designed to accelerate traditional graphics, compute and local memory. In addition, they contain eight matrix engines that can be programmed using Intel® Xe Matrix Extensions (Intel® XMX).

Table I [16] shows the theoretical peak for a single stack using six different levels of precision. The FP32 and FP64 peaks are achievable using the vector engines, but achieving peak throughput at lower precision requires use of Intel® XMX. Making use of the dedicated matrix hardware thus provides an opportunity for significant speed-ups: in theory, switching from FP32 to FP16 could deliver a speed-up of 16x for matrix operations (alongside a 2x reduction in memory footprint and bandwidth requirements).

B. Intel® MKL Precision Configuration

Intel® MKL supports several *alternative compute modes* for BLAS level-3 routines that can offer improved performance in exchange for reduced accuracy or different numerical behavior. Alternative modes include several low precision accelerated modes (leveraging BF16 or TF32 computation) and 3M complex multiplication. By default, MKL does not enable any alternative modes; they are enabled either via dedicated APIs or the `MKL_COMPUTE_MODE` environment variable. The environment variable provides a quick means for application developers to evaluate whether any of the alternative compute

TABLE II
AVAILABLE BLAS COMPUTE MODES. PEAK THEORETICAL SPEEDUP IS QUOTED RELATIVE TO FP32.

Compute Mode	Environment Variable	Peak Theoretical
BF16	FLOAT_TO_BF16	16x
BF16x2	FLOAT_TO_BF16X2	(16/3)x
BF16x3	FLOAT_TO_BF16X3	(8/3)x
TF32	FLOAT_TO_TF32	8x
Complex_3m	COMPLEX_3M	(4/3)x

modes offer performance benefits and acceptable accuracy for the application at hand. Table II provides an overview of the currently available alternative compute modes in MKL.

In the `float_to_{BF16,BF16x2,BF16x3}` modes, MKL internally converts single precision input data to sums of 1, 2, or 3 BF16 values, then uses the fast systolic arrays available on recent discrete GPUs to multiply the resulting BF16 component matrices and accumulate in single precision. As the number of BF16 components increases, so does expected accuracy, to the point that BF16x3 accuracy is comparable to standard single-precision arithmetic. The `float_to_TF32` mode works similarly, with TF32 in place of BF16.

Finally, the `complex_3M` mode enables 3M complex multiplication, reducing the four real multiplications within a standard complex multiplication to three multiplications at the expense of extra additions. This improves theoretical peak performance for level-3 BLAS routines by $\frac{4}{3}$, subject to memory/cache bandwidth restrictions. In general 3M accuracy is comparable with standard complex arithmetic, but with different numeric cancellation behavior.

An element of importance relating to our study in this paper is the percentage of peak performance we observe as it relates to performance speedup. In Table II, the peak theoretical speedups for each of the compute modes is shown, with the maximum being 16x. Actual speedups are more modest, limited by power and bandwidth considerations, and vary by matrix size.

IV. EXPERIMENTAL SETUP AND PRECISION LEVELS

In this section, we describe the hardware and software setup used in our experiments. We also discuss how the varying BLAS precision modes were altered in the application.

A. Hardware Setup

All runs were performed using a dual-card node containing two Intel® Data Center GPU Max Series 1550 [17]. Each GPU has Intel® XMX [18] enabled and contains 128 GB of High Bandwidth Memory (HBM). We ran all experiments on a single stack to avoid Non-Uniform Memory Access (NUMA) effects. Each stack contains 448 EUs at up to 1.6 GHz frequency. At the core of the GPU is the Xe HPC Stack, which is comprised of various tiles stacked on each other as part of a single package. The stack includes the Xe-core Tile (compute tile), L2 cache tile, base tile, high memory bandwidth tile, Xe link tile for scaling, as well as the embedded multi-die interconnect bridge (EMIB) for communication between different Xe HPC Stacks.

Intel® Xe architecture consists of one of three possible micro-architectures: the Xe LP which provides a low power solution, the Xe HPG which is optimized for high fidelity graphical imaging, or the Xe HPC which is optimized for HPC and AI acceleration. The Intel® Data Center GPU Max Series used in this paper is built upon the Xe HPC micro-architecture [19].

Each compute tile in Xe HPC comprises multiple Xe cores. Each Xe core has eight vector engines supporting FP64, FP32, and FP16 precisions, as well as Intel® XMX to support systolic numerics such as TF32, FP16, BF16, and Int8. Integer and floating-point operations execute on separate ports to improve instruction throughput.

B. Software Setup

For all runs, we compiled the most recent release of DCMESH with the Intel® oneAPI Base Toolkit 2024.2.0 and the Intel® oneAPI Math Kernel Library (Intel® oneMKL) 2024.2. The Level Zero driver version used is 23.22.26516.34. DCMESH was built from source using the Intel® DPC++/C++ compiler and the Intel® Fortran Compiler with SYCL and OpenMP offload enabled. In all cases, the offload frameworks are configured to use Level Zero [20]. The build flags used for both FP64 and Mixed Precision versions are included in the Appendix.

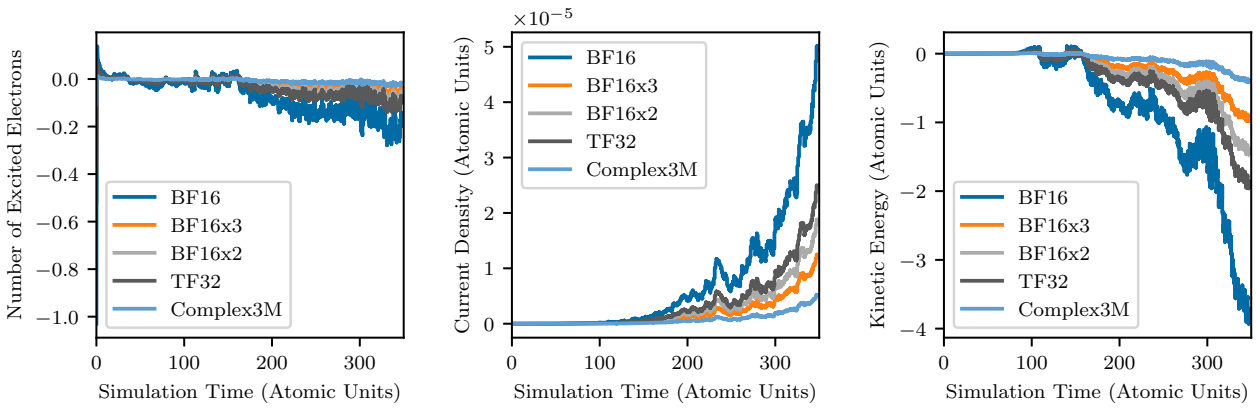
In order to determine performance over a set number of quantum dynamical steps, we used the Profiling Tools Interface for GPU (PTI-GPU) [21]. One particular tool within this interface, called unitrace, was utilized in order to be able to record kernel and other event timings using GPU-side timers.

C. Precision Levels Studied

The different portions of DCMESH either have fixed or varying precision modes. The QXMD portion of the code, which is run exclusively on CPU, is written entirely in Fortran. In addition, it can only be run using FP64 precision as this represents a critical portion of the simulation wherein the wavefunction is initialized by the Self-Consistent Field (SCF) method. The LFD portion of the code, which is offloaded to the GPU, can be run in a range of modes. These include LFD at FP64, FP32, and FP32 with varying BLAS precision levels. The BLAS precision modes available include BF16, BF16X2, BF16X3, TF32, and Complex_3M. Specifically, we are interested in understanding the effects that varying precisions used in matrix multiplication within BLAS routines have on key metrics determined by the LFD portion of DCMESH. Table II contains a more detailed view of the varying BLAS precision modes including the environmental variable which should be set in order to implement the different compute modes.

D. Usage of BLAS in DCMESH

Among the most time-intensive portions of the entire LFD portion of the DCMESH codebase is the nonlocal correction



(a) Number of excited electrons. (b) Current density. (c) Kinetic energy.

Fig. 1. Accuracy of three output metrics, measured as the deviation from FP32 for BLAS calls using different compute modes.

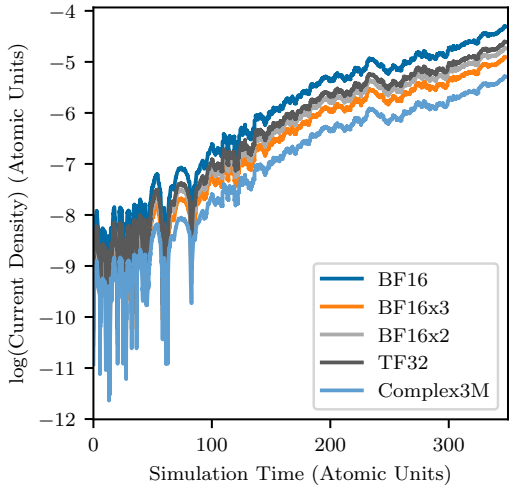


Fig. 2. Accuracy of current density, measured as the log of the deviation from FP32 for BLAS calls using different compute modes.

for time propagation of electronic wave functions. While the electronic wave functions are represented on a finite-difference mesh for simple data parallelism in LFD, this representation is not optimal for the nonlocal correction. We thus map the nonlocal computation to the vector space spanned by the Kohn-Sham electronic wave functions [22]. As a result, this correction is cast into matrix operations by defining an $N_{\text{grid}} \times N_{\text{orb}}$ wave-function matrix, where N_{grid} and N_{orb} are the number of grid points to represent each wave function and that of KS wave functions, respectively.

$$\Psi(t) = c\Psi(0)\Psi^{\tau}(0)\Psi(t) \tag{1}$$

This can be performed as shown in Equation 1, where c is a complex number and Ψ^{τ} denotes a Hermitian transpose matrix. This operation is implemented in the code through use of BLAS kernels. In addition to this, "BLASified" nonlocal correction appears in two other functions in LFD. These include the energy calculation (`calc_energy`) and the remapping of the final wave functions to occupation numbers (`remap_occ`).

Our study is concerned only with how varying the precision of these specific BLAS calls affects performance and numerical accuracy, and we do not consider varying the precision outside of BLAS calls. Additionally, because the Intel® MKL controls are environment variables affecting the library as a whole, our study here is limited to configurations where all BLAS calls are run at the same precision. The effects of running different BLAS calls at different levels of precision is left to future work.

E. DCMESH Configuration Details

Exposing a material such as lead titanate to laser-induced excitation dynamics can be one step towards the development of super capacitors. For the purposes of this study, we are interested in observing the varying effects of precision modes upon various calculated outputs such as the number of excited electrons and the kinetic energy. In terms of the number of atoms studied, we look at a 40 atom and 135 atom systems with mesh grid sizes of 64^3 and 96^3, respectively. For relevance towards the matrix operations within BLAS, the 135 atom system is defined by $N_{\text{grid}} \times N_{\text{orb}}$. While N_{grid} is detailed by the mesh grid size, N_{orb} for the 40 atom system is 256 and is 1024 for the 135 atom system.

In addition to these system details, several more of interest include the resolution of the MD simulation, measured in terms of timesteps, as well as the total number of quantum dynamical steps and the total simulation time in femtoseconds (fs). These are detailed for both system sizes in Table III. Furthermore, Table V contains the number of electronic orbitals and mesh grid sizes for the different system sizes studied.

TABLE III
KEY SIMULATION PARAMETERS.

Simulation Variable	Value
Timestep	0.02
Total Number of QD Steps	21,000
Total Simulation Time (fs)	10

Ultimately, in order to determine the effect of using alternative precision modes upon the framework as a whole, we have to look at the deviation from reference for the most relevant calculated values impacted directly by the BLAS computations. The reference precision, in order to understand the effect of the BLAS calls directly, is FP32 for the LFD portion of DCMESH.

V. ACCURACY RESULTS

In this section we discuss the accuracy of the results for some of the key output metrics that are dependent upon BLAS routines in the LFD portion of the code. The precision of the BLAS routines are the only ones that change and therefore influence the accuracy of the studied output metrics. Chief among these are the kinetic energy and the number of excited electrons, *nexc*. Nonlocal corrections are less pronounced for the use case that we are studying in this paper. This manifests itself computationally such that after every series of 500 quantum dynamical steps (LFD portion at FP32), we execute Self-Consistent Field (SCF) at FP64 to update the wave function and then proceed to the next series of 500 QD steps. A wavefunction is the mathematical description of the quantum state of a series of particles and defines a system's energy, momentum, position, and spin. Updating the wavefunction with FP64 precision prevents the buildup of truncation errors which may otherwise accumulate through the use of lower precision calculations. This is the fundamental reason why the code is able to run with alternative BLAS precision modes.

A. Overall Accuracy

The first step in determining the effect of various precisions in BLAS operations is to understand what these operations influence. There are only a handful of BLAS calls within the LFD portion of the DCMESH code, but their influence is profound. The three primary functions which contain BLAS calls are `nlp_prop`, `calc_energy`, and `remap_occ`. While the time propagation of the electronic wave functions are calculated in `nlp_prop`, BLASified nonlocal correction appears in the energy calculation in `calc_energy`, and in remapping the final wave functions to occupation numbers in `remap_occ`.

Considering where BLAS is implemented, we can examine some of the key metrics that are computed by these functions. Two include the number of excited electrons, *nexc*, and the kinetic energy of the system as a whole. Figure 1 details the effect of varying BLAS precision modes on these two outputs. We also include current density in this set. The latter is not directly computed through BLAS, but is still influenced by computations within BLAS calls, and can be used as a reference. All 5 precision modes were used for simulations of roughly 10 fs for a system of 135 atoms and 1024 orbitals. The exact same computations were performed in each, to ensure a fair comparison. The difference in the value of the outputs between the alternate precision and that of FP32 were extracted and plotted over time.

We observe that the deviation increases over the course of the simulation and increases most significantly for the three variants where BF16 is used: BF16, BF16x2 and BF16x3. These three variants allow a trade-off between accuracy and performance, with BF16 being the fastest in terms of performance and BF16x3 being the most accurate.

TABLE IV
NUMBER OF EXPONENT AND MANTISSA BITS FOR EACH PRECISION FORMAT STUDIED.

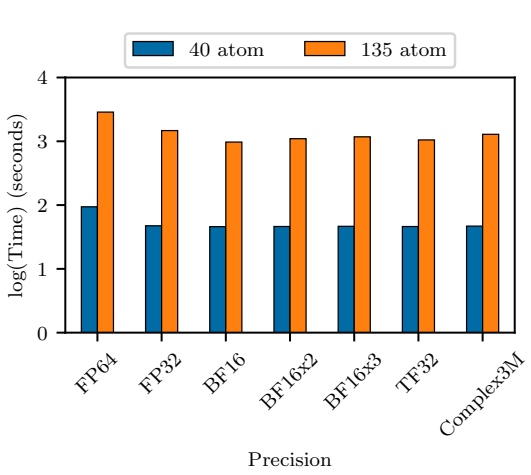
Precision	Exponent Bits	Mantissa Bits
FP64	11	52
FP32	8	23
TF32	8	10
BF16	8	7

In Table IV, we show the formats for the different precision types based on the number of exponent and mantissa bits. TF32 has the same number of mantissa bits as FP16 but the same exponent range of BF16. As a result, TF32 contains slightly higher precision than BF16 and this is also revealed in our results.

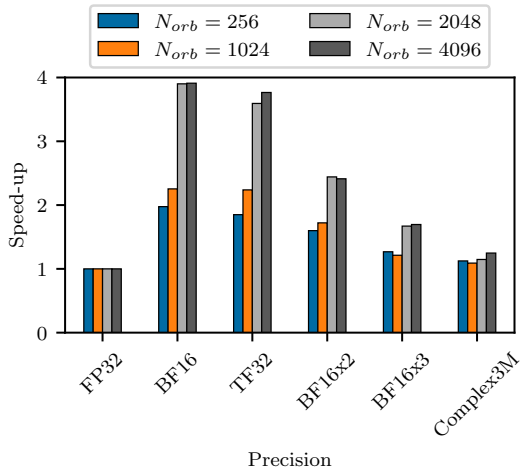
The maximum deviation from FP32 is revealed to be for the case where BF16 is used in BLAS computations of kinetic energy. The deviation is nearly 5 Hartree units which is significant. Typically for a system of this size the energy deviation should not exceed 2-3 Hartree units but given the high dynamical nature of our problem, represented by the fact that the kinetic energy of the system is increasing quickly, it is acceptable. Furthermore, the energy of the entire, 135 atom system is in the order of 10^3 Hartree. The deviation is seemingly much less striking for the case of the number of excited electrons, and negligible for the case of current density where deviation is measured in the order of 10^{-5} Atomic Units.

It should be noted, however, that these deviations must also be understood within context. The deviations relative to the absolute values of each metric are roughly equivalent to each other, in the order of 1%. This indicates that, assuming random data in a bounded range and the absence of numerical cancellation, the relative error of BLAS compute in BF16 to the other modes is independent of matrix size. This lack of significant relative deviation is something that will be discussed further later in the paper.

Kinetic energy is computed through the BLAS call in function `calc_energy`, and is based on a matrix-matrix multiplication with tensor size N_{grid} x N_{orb}. *Nexc* is computed through a BLAS call in function `remap_occ` and is based on a matrix-matrix multiplication with a tensor size of N_{orb} x



(a) Time to completion for 500 Quantum Dynamical steps of 40 and 135 atom systems using DCMESH with varying precision.



(b) Speed-up for the set of BLAS calls as measured with `MKL_Verbose` for a 40 atom system with different orbital sizes.

Fig. 3. Performance results.

TABLE V
SYSTEM SIZES STUDIED. LARGEST SYSTEM THAT CAN FIT WITHIN THE 64GB MEMORY OF A SINGLE GPU STACK IS A 135 ATOM LEAD TITANATE SUPERCELL OF MESH GRID 96X96X96 AND 1024 ELECTRONIC ORBITALS.

Number of Atoms	Mesh Grid Size	N_{orb}
40	64x64x64	256
40	64x64x64	1024
40	64x64x64	2048
40	64x64x64	4096
135	96x96x96	864
135	96x96x96	1024

N_{orb}.

Furthermore, these tensor sizes are also determined based on the size of the system itself. Table V contains the sizes of the mesh grids and number of orbitals studied in this work. As the system size increases, the tensor sizes used in the matrix-matrix multiplications within the BLAS calls also increase.

In Figure 2, we detail a logarithmic scale of the deviation from FP32 for the different precision modes for current density. Over the course of the simulation, BF16, TF32, and BF16X3 track closely with one another and do not show any signs of divergence.

B. BLAS Accuracy

Mentioned earlier, we do not observe that the relative error for `FLOAT_TO_BF16` compared with other modes is significantly different when varying input matrix sizes to the BLAS routines. This requires further explanation.

To understand the accuracy impact from MKL's alternative compute modes, we can consider a simple proxy model with real-valued inputs, in which all but the lowest n mantissa bits of the GEMM input matrices are rounded off. Except for denormal inputs, this induces a maximum 2^{-n-1} relative error in the inputs. Letting a, b represent a full (real) single-precision input value and Δa, Δb the associated perturbations due to rounding, the relative error in multiplication is:

$$\left|\frac{(a+\Delta a)(b+\Delta b)-ab}{ab}\right| \leq \left|\frac{a-\Delta a}{a}\right| + \left|\frac{b-\Delta b}{b}\right| + \left|\frac{\Delta a \cdot \Delta b}{ab}\right| \leq 2^{-n} + o(2^{-n}).$$

Note this is independent of the input data a, b. Each entry of the matrix product AB is a sum of such values, so the above error bound is retained if all products have the same sign.

C. Performance

We measured the performance of individual BLAS calls within DCMESH using unitrace and the environment variable `MKL_VERBOSE`. Unitrace provides the time spent in each kernel (as measured by Level Zero) while the latter gives a breakdown of time for each BLAS call, in addition to precision, matrix sizes, and other parameters. In this section, we will often refer to the GEMM matrix dimensions, which are traditionally labeled m, n, and k, where in the matrix multiplication $C \leftarrow AB$, C is m rows by n columns, A is $m \times k$, and B is $k \times n$.

In Figure 3, the log scale of the average time to completion of 500 quantum dynamical steps for both a 40 and 135 atom system are shown for each precision mode. The 40 atom system size in Figure 3a is studied with N_{orb}=256, while the 135 atom system contains N_{orb}=1024. In the 40 atom system, very little performance change is observed between FP32 and the runs with different BLAS compute modes. Indeed, only between the runs with FP64 and FP32 precisions do we observe any significant change in performance. If we now focus on the 135 atom system, since it is the more computationally demanding variant, the time to complete 500 QD steps is over

TABLE VI
MAXIMUM OBSERVED SPEEDUP OF BLAS ROUTINES USING DCMESH AND COMPARED WITH THE MAXIMUM THEORETICAL SPEEDUP OF BLAS WHEN USING THE DIFFERENT COMPUTE MODES IN MKL.

Compute Modes	Measured Speedup	Theoretical Speedup
BF16	3.91x	16x
TF32	3.76x	8x
BF16X2	2.41x	$(16/3)$x
BF16X3	1.69x	$(8/3)$x
Complex_3M	1.12x	$(4/3)$x

TABLE VII
M, N, AND K INDICES BASED ON INCREASING ORBITAL SIZES FOR THE GEMM ROUTINE IN `remap_occ`. MESH GRID SIZE REMAINS THE SAME AND ORBITAL SIZE INCREASE IS REPRESENTED BY AN INCREASE IN THE VALUE FOR INDEX N.

Number of Atoms	N_{orb}	m	n	k
40	256	128	128	262144
40	1024	128	896	262144
40	2048	128	1920	262144
40	4096	128	3978	262144

2800 seconds at FP64 precision, 1472 seconds at FP32, and 972 seconds when using the BF16 compute mode.

With the FP32 run as the reference, we see significant performance speedup for the LFD code, with the peak performance observed using BF16, resulting in a roughly 1.35x speedup. TF32 provides the next best performance benefit and Complex_3M providing the least performance speedup. As we previously noted for the 40 atom system, there is very little difference in time to completion between the varying BLAS compute modes. This indicates that performance speedup using the different compute modes is directly influenced by the size of the matrices used within BLAS routines.

Figure 3b details a case study for a series of N_{orb} sizes including 256, 1024, 2048, and 4096 and the subsequent effects on the BLAS performance at alternative precisions. This case study allows us to gain a deeper understanding into the impact of matrix size upon efficiency, and hence the relative speed-up achievable at larger matrix sizes. The case with the smallest number of orbitals provides the least degree of improvement while the largest case translates into the greatest speedup between FP32 and alternative precisions.

Table VI compares the maximum observed speedups measured with the theoretical speedups. The maximum speedup we achieved was 3.91x when using the BF16 compute mode, despite the peak theoretical speedup for a BF16 BLAS routine being 16x. There are a number of factors that limit performance relative to theoretical maximum speedups, particularly memory and cache bandwidth limitations and power limitations. The bandwidth limitations stem primarily from the relatively small $m = 128$ dimension, while the power limitations are tied to hardware design.

Previously in Figure 3b we described the BLAS speedup factors for 40 atom systems based on different orbital sizes. In order to more properly understand how the orbital size affects BLAS performance, Table VII contains information regarding the values of m, n, and k for the GEMM call in `remap_occ`. Here, we observe that the value of m remains constant at 128. Similarly, value of k is 64^3, which is the size of the mesh grid for a 40 atom system. The index n is directly based on n_{orb}.

VI. CONCLUSIONS

In this paper, we examined the potential of applying alternative precision modes (BF16, TF32, and Complex_3M) to the execution of BLAS calls in DCMESH and weighed the performance benefits against loss in accuracy for key output parameters.

On a single stack of an Intel® Data Center GPU Max Series 1550, we observed speedups of up to 1.35x while retaining accuracy of key output parameters such as the number of excited electrons, current density, and kinetic energy. For large problem sizes, we observe speed-ups of up to 3.91x for individual BLAS calls.

In future work, we intend to explore the use of alternative BLAS precision modes in other workloads such as QMCPACK. Furthermore, we would like to continue our work with DCMESH in the analysis of how alternative BLAS precision modes impact accuracy and performance in multi-stack and multi-node runs.

ACKNOWLEDGMENT

We would like to thank the Hatch Cluster, used to run the framework on the Intel® Data Center GPU Max Series 1550. TMR is supported by an NSF grant OAC-2118061, and AN is supported by a PSU/ONR grant, S005542-ONR.

DISCLAIMERS

REFERENCES

[1] H. Peng, K. Wu, Y. Wei, G. Zhao, Y. Yang, Z. Liu, Y. Xiong, Z. Yang, B. Ni, J. Hu, *et al.*, "Fp8-lm: Training fp8 large language models," *arXiv preprint arXiv:2310.18313*, 2023.

[2] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[3] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[4] D. Li, A. S. Rawat, M. Zaheer, X. Wang, M. Lukasik, A. Veit, F. Yu, and S. Kumar, "Large language models with controllable working memory," *arXiv preprint arXiv:2211.05110*, 2022.

[5] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[6] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1, pp. 19–25, 2015.
[7] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. In't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, *et al.*, "Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications*, vol. 271, p. 108171, 2022.
[8] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.
[9] J. Hafner, "Ab-initio simulations of materials using vasp: Density-functional theory and beyond," *Journal of computational chemistry*, vol. 29, no. 13, pp. 2044–2078, 2008.
[10] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley, *et al.*, "Qmcpack: an open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids," *Journal of Physics: Condensed Matter*, vol. 30, no. 19, p. 195901, 2018.
[11] "Performance and precision: Floating point for nvidia gpus." https://docs.nvidia.com/cuda/pdf/Floating_Point_on_NVIDIA_GPU.pdf. Accessed: 2024-08-07.
[12] H. Wang, L. Zhang, J. Han, and E. Weinan, "Deepmd-kit: A deep learning package for many-body potential energy representation and molecular dynamics," *Computer Physics Communications*, vol. 228, pp. 178–184, 2018.
[13] N. Piroozan and N. Kumar, "Enabling performant thermal conductivity modeling with deepmd and lammps on cpus," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 88–94, 2023.
[14] A. Musaelian, S. Batzner, A. Johansson, L. Sun, C. J. Owen, M. Kornbluth, and B. Kozinsky, "Learning local equivariant representations for large-scale atomistic dynamics, 2022," *arXiv preprint arXiv:2204.05249*.
[15] T. M. Razakh, T. Linker, Y. Luo, R. K. Kalia, K.-I. Nomura, P. Vashishta, and A. Nakano, "Accelerating quantum light-matter dynamics on graphics processing units," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1057–1066, IEEE, 2024.
[16] H. Jiang, "Intel's Ponte Vecchio GPU : Architecture, Systems & Software," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, pp. 1–29, 2022.
[17] "Intel® data center gpu max series 1550." https://www.intel.com/content/www/us/en/products/details/discrete-gpus/data-center-gpu/max-series.html. Accessed: 2024-08-06.
[18] "Intel® xe matrix extensions." https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2023-1/intel-xe-gpu-architecture.html. Accessed: 2024-08-06.
[19] "Intel® data center gpu max series 1550 architecture." https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html. Accessed: 2024-08-08.
[20] "Intel®oneapi level zero offload." https://www.intel.com/content/www/us/en/developer/articles/technical/zero-in-on-level-zero-oneapi-open-backend-approach.html. Accessed: 2024-08-07.
[21] M. Umar and M. Jong, "Pti-gpu: Kernel profiling and assessment on intel gpus," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 681–684, 2023.
[22] C. Y. Wang, P. Elliott, S. Sharma, and J. K. Dewhurst, "Real time scissor correction in td-dft," *Journal of Physics: Condensed Matter*, vol. 31, no. 21, p. 214002, 2019.

APPENDIX

A. Hardware Specifications

Intel® Data Center GPU Max Series : 1-node, Total Memory 128 GB, kernel 5.14.21-150500.55.52-default, compiler gcc 7.5.0 20210514, Intel® oneAPI 2024.2.0, DC-MESH (Git Commit Version: a47575 2232fdedfcc0ec5aefed79ce555ecb3960), Intel® oneMKL 2024.2.0, Intel® pti-gpu.

B. Build Flags

Mixed Precision Build Flags
-DCMAKE_CXX_COMPILER=icpx
-DCMAKE_Fortran_COMPILER=ifx
-DLFD_ENABLE_OFFLOAD=ON
-DDCMESH_ENABLE_MPI=OFF
-DDCMESH_ENABLE_SYCL=ON
-DOFFLOAD_ARCH=pvc
-DCMAKE _CXX_FLAGS="-march=native -g"
-DCMAKE_Fortran_FLAGS="-march=native -g"
-DLFD_ENABLE_MIXED_PRECISION=ON
Double Precision Build Flags
-DCMAKE_CXX_COMPILER=icpx
-DCMAKE_Fortran_COMPILER=ifx
-DLFD_ENABLE_OFFLOAD=ON
-DDCMESH_ENABLE_MPI=OFF
-DDCMESH_ENABLE_SYCL=ON
-DOFFLOAD_ARCH=pvc
-DCMAKE _CXX_FLAGS="-march=native -g"
-DCMAKE_Fortran_FLAGS="-march=native -g"
-DLFD_ENABLE_MIXED_PRECISION=OFF

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 Improve performance of DCMESH through the introduction of alternative precisions for the various BLAS routines in the code.

C_2 Show that the resultant 1.35x speedup for the time to solution of the simulation does not result in a precipitous decline in accuracy for 3 key metrics.

C_3 Examine the improved BLAS routine performance for large problem sizes and compare to peak theoretical improvement on the GPU.

B. Computational Artifacts

Computational Artifact and respective DOI.

A_1 DOI:10.1109/IPDPSW63119.2024.00176

A_2 DOI:10.1109/IPDPSW63119.2024.00176

A_3 DOI:10.1109/IPDPSW63119.2024.00176

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1	Tables 1-2 Figure 3
A_2	C_2	Tables 3-5 Figures 1-2
A_3	C_3	Tables 6-7 Figures 3

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

In order to reproduce all elements described in C_1, it is important to recognize that the artifact A_1 contains the basis for how to set up and run the code, DCMESH. Upon this, we then are able to introduce alternative BLAS precisions in order to ascertain the improved performance observed in the paper.

Expected Results

After importing the DCMESH codebase and building it, the simulation will progress over the course of many series of quantum dynamical steps. For this specific contribution being studied, each simulation will take roughly 30 minutes to complete. The principal outputs relevant for C_1 are given by the the use of Intel®Unitrace. In order to gauge the speedup factor when using DCMESH with alternative BLAS precisions, we can use this tool for a test simulation over several hundred quantum dynamical tests.

The larger the matrix size, the more impact alternative BLAS precisions will have. For a 135 atom system, the fastest simulation is for the case when BLAS precision is BF16, followed by TF32, BF16X2, BF16X3, Complex_3M, FP32, and then FP64.

Expected Reproduction Time (in Minutes)

The expected time to reproduce the results for C_1 is about 30 minutes for each simulation on the GPU.

Artifact Setup (incl. Inputs)

Hardware: The hardware to be used is the Intel®Data Center GPU Max Series 1550. To reproduce results, only a single tile is needed. Each card has 2 tiles.

Software: The principal software package that is to be used is the Intel®oneAPI Base Toolkit 2024.2.0 and the Intel®oneAPI Math Kernel Library (oneMKL) 2024.2. The L0 driver version used is 23.22.26516.34. DCMESH was built from source using the Intel®DPC++/C++ compiler icpx CXX and the Intel®Fortran Compiler ifx with SYCL and OpenMP offload enabled. In all cases the offload frameworks are configured to use Level Zero. The build flags used for both FP64 and Mixed Precision versions are included in the Disclaimers section of this paper. In order to determine performance over a set number of quantum dynamical steps, we use the Profiling Tools Interface for GPU (PTI-GPU). The particular tool that is relevant is called Unitrace, which is used to record kernel and other event timings using GPU-side timers.

Please note that DCMESH is a private repository and requires access to be granted prior to being able to build the code.

Datasets / Inputs: The datasets to be used in the codebase in order to reproduce the results can be provided by the authors of the paper. These include PTOquick.dc, CONFIG, and lfd.in. There are different input files for the various 135 and 40 atom system sizes.

Installation and Deployment: In order to install and then build the workflow in your workspace of choice, please follow these steps.

git clone https://github.com/USCCACS/DCMESH.git
cd DCMESH/configs
sh build_ortce.sh
cd build_intel_oneapi_offload_sycl_debug/Sources/tests/dcehd-quick-PTO-p1

Use appropriate PTOquick.dc and CONFIG files under folder /control Use appropriate lfd.in file under folder /control_lfd.

At this point, the code can be executed using the steps outlined in the next section.

Artifact Execution

Table 1 and 2 do not require execution of the code to determine. These can be calculated based on the hardware specifications. These include the number of EUs, peak frequency, and the precision in question. Figure 3, however, is

directly determined based on the execution of the simulation. This section will detail how to run the code. Figure 3a details performance improvements for both 40 and 135 atom systems. Figure 3b represents the speedup factor for a 40 atom system at varying number of orbitals which is controlled in PTOquick.dc. Using the appropriate input data, which can be provided by the authors, you execute the code from the following folder:

cd /some/workspace/DCMWESH/build_intel_oneapi_offload_sycl_debug/Sources/tests/dcehd-quick-PTO-p1

Run the following command at this point for 500 QD steps to test performance for FP32 precision:

export KMP_BLOCKTIME=0
unitrace -k ../../../bin/dcehd

Run the following command at this point for 500 QD steps to test performance for BF16 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16
unitrace -k ../../../bin/dcehd

Run the following command at this point for 500 QD steps to test performance for BF16X2 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X2
unitrace -k ../../../bin/dcehd

Run the following command at this point for 500 QD steps to test performance for BF16X3 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X3
unitrace -k ../../../bin/dcehd

Run the following command at this point for 500 QD steps to test performance for TF32 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_TF32
unitrace -k ../../../bin/dcehd

Run the following command at this point for 500 QD steps to test performance for COMPLEX_3M BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=COMPLEX_3M
unitrace -k ../../../bin/dcehd

Artifact Analysis (incl. Outputs)

With each of the above executions, you will obtain an L0 time that will represent the total wall time on the GPU. Comparing these times between the various precision types forms the basis for Figure 3a. Figure 3b is determined by a separate computational method and will be described in another section.

B. Computational Artifact A_2

Relation To Contributions

In order to reproduce all elements described in C_2, it is important to recognize that the artifact A_2 contains the basis for how to set up and run the code, DCMESH. Upon this, we then are able to introduce alternative BLAS precisions in order to ascertain the accuracy determined in the paper by comparing alternative precisions with FP32.

Expected Results

After importing the DCMESH codebase and building it, the simulation will progress over the course of many series of quantum dynamical steps. For this specific contribution being studied, each simulation will take roughly 2 days to complete. In order to gauge the accuracy of specific outputs when using DCMESH with alternative BLAS precisions, we can simply run the code with the appropriate input files.

Expected Reproduction Time (in Minutes)

The expected time to reproduce the results for C_2 is about 2 days for each simulation on the GPU.

Artifact Setup (incl. Inputs)

Hardware: The hardware to be used is the Intel®Data Center GPU Max Series 1550. To reproduce results, only a single tile is needed. Each card has 2 tiles.

Software: The principal software package that is to be used is the Intel®oneAPI Base Toolkit 2024.2.0 and the Intel®oneAPI Math Kernel Library (oneMKL) 2024.2. The L0 driver version used is 23.22.26516.34. DCMESH was built from source using the Intel®DPC++/C++ compiler icpx CXX and the Intel®Fortran Compiler ifx with SYCL and OpenMP offload enabled. The build flags used for both FP64 and Mixed Precision versions are included in the Disclaimers section of this paper.

Datasets / Inputs: The datasets to be used in the codebase in order to reproduce the results can be provided by the authors of the paper. These include PTOquick.dc, CONFIG, and lfd.in. There are different input files for the various 135 and 40 atom system sizes.

Installation and Deployment: In order to install and then build the workflow in your workspace of choice, please follow these steps.

git clone https://github.com/USCCACS/DCMESH.git
cd DCMESH/configs
sh build_ortce.sh
cd build_intel_oneapi_offload_sycl_debug/Sources/tests/dcehd-quick-PTO-p1

Use appropriate PTOquick.dc and CONFIG files under folder /control Use appropriate lfd.in file under folder /control_lfd.

At this point, the code can be executed using the steps outlined in the next section.

Artifact Execution

Tables 3, 4, and 5 do not require execution of the code to determine. Table 3 is obtained directly from the input file, Table 4 is commonly known exponent and mantissa bit information regarding different precision types, and Table 5 contains information directly available from the input data scripts. Figures 1 and 2, however, are directly determined based on the execution of the simulation. This section will detail how to run the code. Figure 1 regards the accuracy measurements determined as the deviation from FP32 for the various BLAS precision modes. Figure 1a concerns the number of excited electrons, Figure 1b concerns the current density, and Figure 1c concerns the kinetic energy of the system. Figure 2 shows the accuracy measurement for current density as the deviation from FP32 for the various BLAS precision modes in log scale. Use the following steps in order to reproduce these results:

cd /some/workspace/DCMWESH/build_intel_oneapi_offload_sycl_debug/Sources/tests/dcehd-quick-PTO-p1

Run the following command at this point for the full simulation to test performance for FP32 precision:

```
export KMP_BLOCKTIME=0
../../../bin/dcehd
```

Run the following command at this point for the full simulation to test performance for BF16 BLAS Precision:

```
export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16
../../../bin/dcehd
```

Run the following command at this point for the full simulation to test performance for BF16X2 BLAS Precision:

```
export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X2
../../../bin/dcehd
```

Run the following command at this point for the full simulation to test performance for BF16X3 BLAS Precision:

```
export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X3
../../../bin/dcehd
```

Run the following command at this point for the full simulation to test performance for TF32 BLAS Precision:

```
export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_TF32
../../../bin/dcehd
```

Run the following command at this point for the full simulation to test performance for COMPLEX_3M BLAS Precision:

```
export KMP_BLOCKTIME=0
export MKL_BLAS_COMPUTE_MODE=COMPLEX_3M
../../../bin/dcehd
```

Artifact Analysis (incl. Outputs)

In each of the above simulations, it is advisable to pipe the output into a respective text file. The main portions of the simulations which are of concern regard the main LFD loops. Each successive MD step will contain a series of QD steps. These QD steps are what is of interest and contain all of the important output variables. In order from left to right, these are ekin, epot, etot, eexc, nexc, Aext, and javg. For the purposes of this section, we are concerned with ekin, nexc, and javg. These are the kinetic energy, number of excited electrons, and the current density.

C. Computational Artifact A_3

Relation To Contributions

In order to reproduce all elements described in C_3, it is important to recognize that the artifact A_3 contains the basis for how to set up and run the code, DCMESH. Upon this, we then are able to introduce alternative BLAS precisions in order to ascertain the improved performance observed in the paper for the specific BLAS functions.

Expected Results

After importing the DCMESH codebase and building it, the simulation will progress over the course of 50 QD steps. For this specific contribution being studied, each simulation will take roughly 5 minutes to complete. In order to determine the runtime for each BLAS call, we can use a command within MKL to print these times.

Expected Reproduction Time (in Minutes)

The expected time to reproduce the results for C_2 is about 10 minutes for each simulation on the GPU.

Artifact Setup (incl. Inputs)

Hardware: The hardware to be used is the Intel®Data Center GPU Max Series 1550. To reproduce results, only a single tile is needed. Each card has 2 tiles.

Software: The principal software package that is to be used is the Intel®oneAPI Base Toolkit 2024.2.0 and the Intel®oneAPI Math Kernel Library (oneMKL) 2024.2. The L0 driver version used is 23.22.26516.34. DCMESH was built from source using the Intel®DPC++/C++ compiler icpx CXX and the Intel®Fortran Compiler ifx with SYCL and OpenMP offload enabled. The build flags used for both FP64 and Mixed Precision versions are included in the Disclaimers section of this paper.

Datasets / Inputs: The datasets to be used in the codebase in order to reproduce the results can be provided by the authors of the paper. These include PTOquick.dc, CONFIG, and lfd.in. There are different input files for the various 40 atom system sizes.

Installation and Deployment: In order to install and then build the workflow in your workspace of choice, please follow these steps.

git clone https://github.com/USCCACS/DCMESH.git
cd DCMESH/configs
sh build_ortce.sh
cd build_intel_oneapi_offload_sycl_debug/Sources/tests/dcehd-quick-PTO-p1

Use appropriate PTOquick.dc and CONFIG files under folder /control Use appropriate lfd.in file under folder /control_lfd.

At this point, the code can be executed using the steps outlined in the next section.

Artifact Execution

MKL_VERBOSE=2 can be used as the environmental variable to give detailed BLAS information. This feature will provide the matrix size as well as synchronous timing. By comparing the average BLAS time for the specific BLAS call in question, we will be able to repeat this for all of the precision modes. With this timing obtained, then Table 6 can be reproduced. Furthermore, Table 7 can also be determined by using this environmental variable for 40 atom systems at varying number of orbitals. The BLAS matrix dimensions m, n, and k are directly sourced from the outputs of the code.

Figures 3b is the speedup factor for 40 atom systems at varying number of orbitals and can be plotted with information directly obtained using the aforementioned environmental variable. The timings for each BLAS call are printed on the wall of the simulation when the main LFD loop initiates for each MD step. In order to actually run and reproduce results, using the correct input data, please follow these steps:

cd /some/workspace/DCMWESH/build_intel_oneapi_offload_sycl _debug/Sources/tests/dcehd-quick-PTO-p1

Run the following command at this point for the full simulation to test performance for FP32 precision:

export KMP_BLOCKTIME=0
export MKL_VERBOSE=2
../../../bin/dcehd

Run the following command at this point for the full simulation to test performance for BF16 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_VERBOSE=2
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16
../../../bin/dcehd

Run the following command at this point for the full simulation to test performance for BF16X2 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_VERBOSE=2
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X2
../../../bin/dcehd

Run the following command at this point for the full simulation to test performance for BF16X3 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_VERBOSE=2
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X3
../../../bin/dcehd

Run the following command at this point for the full simulation to test performance for TF32 BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_VERBOSE=2
export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_TF32
../../../bin/dcehd

Run the following command at this point for the full simulation to test performance for COMPLEX_3M BLAS Precision:

export KMP_BLOCKTIME=0
export MKL_VERBOSE=2
export MKL_BLAS_COMPUTE_MODE=COMPLEX_3M
../../../bin/dcehd

Artifact Analysis (incl. Outputs)

In each of the above commands, the relevant information is directly printed to the wall of the simulation. Please wait for the main LFD loop to initiate. Each QD steps contains 9 BLAS calls and these are represented by 9 outputs when using MKL_VERBOSE=2.

Artifact Evaluation (AE)

A. Computational Artifact A_1

Artifact Setup (incl. Inputs)

All relevant information has been described in the AD section regarding how to build and compile DCMESH on the Intel®Data Center GPU Max Series 1550.

Artifact Execution

All relevant information regarding how to execute DCMESH have been included in the AD portion.

Artifact Analysis (incl. Outputs)

Contributions included within C_1 of this paper are obtained using Intel®Unitrace. This tool, and how to execute it, have been described in the AD portion, and the expectation is that when utilized you will observe an output directly printed to the wall of your application used to SSH into the respective node containing the GPU. The most pertinent information is obtained at the very top of this output block and contains the Total L0 Time in nanoseconds. Please use this number to determine the performance of your simulation. This data is used to reproduce Figure 3a.

B. Computational Artifact A_2

Artifact Setup (incl. Inputs)

All relevant information has been described in the AD section regarding how to build and compile DCMESH on the Intel®Data Center GPU Max Series 1550.

Artifact Execution

All relevant information regarding how to execute DCMESH have been included in the AD portion.

Artifact Analysis (incl. Outputs)

Contributions included within C_2 of this paper are obtained by simply running the code in the manner outlined in the AD portion. The expectation is that when utilized you will observe an output directly printed to the wall of your application used to SSH into the respective node containing the GPU. It will take roughly 2 days to run the entire simulation using a single tile of this GPU. The pertinent information utilized to measure accuracy is printed in the LFD loop of each MD step. Each MD step contains 500 QD steps. When running your simulations, it is expected that you will obtain at least 40 MD steps in total. When the LFD loop initiates, please plot nexc, ekin, and Javg over the course of all QD steps. The results from the FP32 simulation are the reference. The values for each variable, when using the alternative precisions are also determined in the exact same way. Please note that in Figure 1, the plots shown reflect the difference between the outputs of a compute mode and FP32. Figure 2 represents the log of the deviation from FP32 for the various BLAS precision modes.

C. Computational Artifact A_3

Artifact Setup (incl. Inputs)

All relevant information has been described in the AD section regarding how to build and compile DCMESH on the Intel®Data Center GPU Max Series 1550.

Artifact Execution

All relevant information regarding how to execute DCMESH have been included in the AD portion.

Artifact Analysis (incl. Outputs)

Contributions included within C_3 of this paper are obtained by simply running the code in the manner outlined in the AD portion. This includes the use of the environmental variable MKL_VERBOSE=2. The information that is desired forms the basis for Tables 6, 7 and Figure 3b.

When using this environmental variable, the relevant information is outputted directly to the wall of your software you are using to SSH into the node containing the GPU. When the main LFD loop initiates, please look for 9 outputs containing all of the relevant information for each of the BLAS calls. This information includes the dimensions of the matrix as well as the timing to complete the matrix multiplication. This information serves as the basis for Tables 6, 7 and Figure 3b.

Figure 3b contains the speedup factor for the BLAS times at various precisions. FP32 is the reference. Please compare the time to complete the BLAS operation at each alternative precision mode to FP32 and you will be able to reproduce Figure 3b. This data is also used in Table 6 which also serves to compare your speedup against the peak theoretical speedup possible. Table 7 contains the matrix sizes for each BLAS operation and is directly printed by MKL_VERBOSE.

With this, all of the results described in this paper can be reproduced. Thank you very much for your interest in our work!