# Multi-level parallelization of quantum-chemical calculations

Dmitri G. Fedorov[1*], Buu Q. Pham[2]

[1] *Research Center for Computational Design of Advanced Functional Materials (CD-FMat), National Institute of Advanced Industrial Science and Technology (AIST), 1-1-1 Umenzono, Tsukuba, Ibaraki 305-8568, Japan*

[2] *Department of Chemistry and Ames Laboratory, Iowa State University, 201 Spedding Hall, Ames, IA 50011, USA*

* Author to whom correspondence should be addressed: d.g.fedorov@aist.go.jp

**Abstract**

Strategies for multiple-level parallelizations of quantum-mechanical calculations are discussed, with an emphasis on using groups of workers for performing parallel tasks. These parallel programming models can be used for a variety *ab initio* quantum chemistry approaches, including the fragment molecular orbital method and replica-exchange molecular dynamics. Strategies for efficient load balancing on problems of increasing granularity are introduced and discussed. A 4-level parallelization is developed based on a multi-level hierarchical grouping, and a high parallel efficiency is achieved on the Theta supercomputer using 131,072 OpenMP threads.

## 1. Introduction

Quantum-mechanical (QM) methods [1] can be used for describing various physicochemical properties in molecular systems. Boosting scalability and efficiency of

QM methods is challenging due to the non-uniform granularity of work, for example, electron repulsion integrals (ERIs) computed in terms of atomic basis functions. There is a variety of massively parallel QM programs, such as NWChem,[2] NTChem,[3] OpenFMO,[4] Smash,[5] and Molcas[6] to name just a few.

A message passing interface (MPI) library optimized for a particular hardware can be used for parallelizing QM code. MPI can be combined with OpenMP in a hybrid parallel programming model, in which MPI handles multiple nodes efficiently while OpenMP uses CPU cores in each node. OpenMP can create and destroy teams of lightweight threads with low overhead, which can share memory reducing its footprint compared with the use of private replicated memory in MPI.

In a parallel QM computation, the entire set of compute nodes is conventionally used to process one task after another. There are, however, scenarios in QM calculations, where multiple loosely connected tasks can be done nearly independently. Typical examples of such tasks are numerical gradient and fragment-based QM methods.[7] In a fragment-based method, computing single point energy of fragments one by one using all nodes is not the most efficient approach. A large speedup can be gained by distributing these relatively independent tasks over groups of compute processes. For instance, by using groups, an fragment molecular

orbital (FMO) energy calculation of $(H_2O)_{256}$ was accelerated by a factor of 93. [8]

The independent nature of computational tasks facilitates localizing communications within groups of workers, and tasks are distributed among these groups. Such a group-based model can be implemented using a communicator splitting inherently available in MPI, which has been used in various quantum chemistry programs, for example, in ABINIT-MP[9,10] efficiently ported to the K[11] and Fugaku[12] supercomputers.

The use of groups can lead to a significant benefit in performance. However, there are also particular problems that arise due to the load balancing of tasks of different sizes. In this paper, a 4-level parallelization technique is developed that can significantly improve load balance and enhance computational efficiency on massively parallel computers.

## 2. Methodology

Many QM methods implemented in general atomic and molecular electronic structure systems (GAMESS)[13,14,15] are parallelized on both CPUs and GPUs[16,17,18,19]. In this work, mainly CPU parallelizations are described, but GPU usage is also briefly covered

separately. For clarity, the definition of terms used in this paper is provided in Table 1.

**Table 1. Definition of terms.**

| Term | Definition |
|---|---|
| Physical node | A computing unit containing one or more processors, such as a single standalone Unix computer. |
| Logical node | A fraction of resources (CPU cores) in a physical node. |
| DDI | The parallel library used in GAMESS. Loosely speaking, DDI=MPI and DDI=(socket library) are the two most commonly seen implementations, but there are others (DDI=ARMCI).[20] |
| DDI process | An instance of a program running in parallel with other likewise instances. Each DDI process has its own resources (files, network connections, and memory). |
| DDI rank | A serial number identifying a DDI process. |
| Data server process | A DDI process running GAMESS executable that serves data (shared arrays) to compute processes and handles their parallel operations. |
| Compute process | A DDI process running GAMESS executable that does QM calculations, so it is also called a worker. |
| Thread | A lightweight process executing a particular parallel task, spawned by a DDI process. |
| Group | A set of DDI processes doing a parallel task together. Although customarily the group size refers to compute processes only, in fact, a group includes both compute processes and data servers. |
| Scope | A span of DDI processes over which a parallel operation is executed; group, world, and universe and the main scopes. |

## 2.1 Load balancing

Load balancing refers to dividing work among DDI processes and/or OpenMP threads.

With a multi-level parallelization, work can be divided among groups at various levels.

In GAMESS, there are two main strategies, static (SLB) and dynamic (DLB) load balancings, which are chosen by setting the value of the BALTYP keyword to SLB or DLB. In the SLB, work is divided according to a predefined index, irrespective of the actual progress. As a result, many workers finishing ahead of others have to wait at a synchronization point. In the DLB, each worker asks for the next task whenever it is idle. Despite the extra bookkeeping cost,, DLB is usually the more efficient.

Besides the SLB and the DLB, there is a more sophisticated preoptimized work load balancing scheme, known as heuristic static load balancing (HSLB).[21] In HSLB, work indices are assigned based on a procedure optimizing the predicted cost of tasks. modeled by a polynomial of the number of basis functions *M*. However, the cost may not be a simple function of *M*, and also depend on the polarizing environment that can affect the number of self-consistent field (SCF) iterations. HSLB does not use dynamic counters, which may be costly on massively parallel computers. Instead, HSLB uses static load balancing, with a group division designed with the aid of external programs.

The load balancing can be more complex when heterogeneous computers are used, as in grid computing. Here, an efficient load balancing scheme should take into

account not only the task size but also the computational capability of workers.[22] Machine learning techniques can be used to improve load balancing.[23]

It is advantageous to mix SLB and DLB schemes in the semi-dynamic load balancing (SDLB),[24] used for a set of tasks, when few tasks are expensive, and the rest are cheap. Examples are fragment-based computations of a large solute with small solvent molecules, or a large ligand bound to a protein divided into small amino acid residues. Another scenario is when some fragments use a QM method with a very large computational cost. This occurs in fragment-based multiconfiguration SCF (MCSCF)[25] and time-dependent density functional theory (DFT), TDDFT,[26] where only one fragment is computed with these high scaling methods, and the rest are treated with much cheaper Hartree-Fock (HF) or DFT. SDLB is chosen in GAMESS by setting the BALTYP value to DLB. In addition, LOADGR defines the number of static groups and LOADBF sets the size of tasks considered large.[27]

For the case when tasks have very different cost, it is efficient to use large groups for expensive tasks, and small groups for small tasks. The DLB scheme is not efficient in this case because there is no guarantee that expensive tasks are assigned to large groups. In SDLB, SLB is used to assign large tasks to large groups statically, while

DLB is used for a dynamic distribution of small tasks over small groups. After finishing their tasks, SLB groups switch to DLB and processing the remaining small tasks.

## 2.2 One-level parallelizations

One-level parallelization corresponds to the most simple, flat model of parallelization in which there is just one set of workers assigned to do tasks one by one.

### 2.2.1 *Distributed Data Interface*

The distributed data interface (DDI)[28] is used in GAMESS as a front end hiding specific parallel library details from the use of parallel functionality in QM calculations. DDI can be implemented based on a native socket library or MPI. Almost all QM methods in GAMESS are parallelized using DDI. DDI offers a flat one-level parallelization when all nodes work together as one team. In DDI, two processes are typically executed per core, one doing calculations and another doing parallel communications (data server). The number of compute processes used in DDI is set in the running script *rungms*, passed to the kickoff program (*ddikick* for sockets and *mpirun* for MPI).

### 2.2.2 *OpenMP*

There were two independent OpenMP implementations[29,30] of GAMESS in private versions prior to the first official release of OpenMP. Although OpenMP can be used as

a one-level parallelization (one compute process spawns multiple threads), such runs are a particular case of a more general two-level parallelization, described below. The number of OpenMP threads is defined by setting the value to the `OMP_NUM_THREADS` variable in the running script *rungms*.

### 2.3 Two-level parallelizations

A two-level parallelization can be achieved either by using groups of DDI processes or in a hybrid DDI/OpenMP model.

#### 2.3.1 *Generalized distributed data interface*

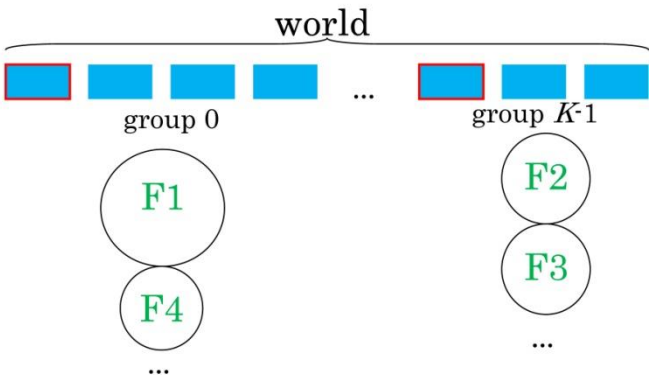


Figure 1. 2-level parallelization in GDDI/2 for dividing all nodes (world) into *K* groups. Workers are shown as blue boxes. The master process in each group is in a red box. Tasks assigned to groups are labeled as F*I*.

Generalized distributed data interface (GDDI)8 was developed to enable a two-level parallelization, so it is denoted by GDDI/2. GDDI is a modification of DDI, where compute nodes can be divided into an arbitrary number of groups *N*, by setting

NGROUP=*N*. Groups are assigned suitable computational tasks such as individual water molecules in a water droplet. Within each group, work is distributed among compute processes (Figure 1).

It is possible to create groups of equal size (e.g., the same number of compute processes), or groups of customized sizes (with the MANNOD keyword in $GDDI listing the size of each group). The customized group setting can take into account various physical node sizes in heterogeneous computing.

To overcome a limitation of GDDI that only whole nodes are divided into groups, it is possible to split physical nodes into logical nodes. These logical nodes may then be divided into groups. For example, by listing a node twice with a halved number of cores, two logical nodes can be created. For MPI, alternatively, logical node splitting can be done inside DDI by an environmental variable `DDI_LOGICAL_NODE_SIZE`.

One of the main features of DDI is to provide handling of distributed arrays allocated across multiple compute nodes. GDDI also supports distributed arrays in the scope of DDI ranks in a group (referred to as a group communication), or DDI ranks across all groups (a world communication). Parallel operations such as a global sum or a broadcast, can be performed in the scope of a group or among all groups (world).

The load balancing in GDDI takes place in the world or group scope. In the

world scope, fragments (or molecules) are distributed over groups. In the group scope (the load balancing controlled by the keyword BALTYP in the group $SYSTEM), work is divided among compute processes in each group. The load balancing at the world level (controlled by BALTYP in $GDDI) is handled by the data server of the lowest rank. In massively parallel environments with a huge number of tasks, this bookkeeping creates a heavy burden on the data server process, which can slow down the whole calculation.

To deal with this problem, an improved scheme of DLB can be used, in which an index of a global counter refers to blocks of tasks rather than single tasks. The size of the block of tasks is chosen with the keyword NUMDLB. The number of global counter requests is thus reduced by a factor of NUMDLB.

Applications of GDDI/2 are summarized in Table 2. They encompass a variety of methods featuring a granularity of tasks suitable for a two-level parallelization.

**Table 2. Two-level parallelization of QM methods using GDDI/2.**[a]

| Method | Upper level | References |
|---|---|---|
| Fragment molecular orbital (FMO) | fragments | 31,32 |
| Effective fragment molecular orbital (EFMO) | fragments | 33,34 |
| Replica-exchange molecular dynamics (REMD) | replicas | 35 |
| Umbrella sampling molecular dynamics (USMD) | windows | 35 |
| Replica-exchange umbrella sampling (REUS) | replicas | 36 |
| Numerical gradients | shifted molecules | 8 |
| Vibrational self-consistent field (VSCF) | molecules on a grid | 37,38 |
| VeraChem method 2 (VM2) | conformers | 39 |
| Divide-and-conquer (DC) | subsystems | 40 |
| Dynamic nucleation theory (DNT) | Monte-Carlo chains | 41,42 |

[a] A lower levels task is usually an integral batch.

### 2.3.2 *One-level parallelization using GDDI*

A one-level parallelization can be carried out using DDI. There are two other ways to perform one-level parallelization in GAMESS using a particular running mode of GDDI/2, denoted by GDDI/1. The first one (one-core groups) is to use *N* groups consisting of a single compute process created with NGROUP=*N*. The second way (split group) is to use a single group in GDDI, splitting it internally (NGROUP=1 NSUBGR=-1), so that each process behaves as if it is an independent group. These GDDI/1 running modes are often used for FMO based density-functional tight-binding (DFTB) calculations,[43,44] where it is advantageous to use a single compute process per fragment.

The two GDDI/1 models have usually a very similar performance, but a different handling of OS limitations (of the maximum number of processes etc). Of the two, the split group model is usually the better one.

### 2.3.3 *DDI/OpenMP*

An alternative way of doing a two-level parallelization is offered by a combination of DDI with OpenMP. The socket-based DDI cannot be properly used with OpenMP, so DDI/OpenMP usually means MPI/OpenMP.

The MPI version of DDI with the *multiple* type of thread safety (controlled by the `MPI_THREAD_MULTIPLE` environmental variable in a running script) can be used with OpenMP. DDI/OpenMP enables a two-level parallelization, where DDI compute processes are used for coarse-grained work distributions, while threads in OpenMP are used for actual computations.

A typical use of DDI/OpenMP is for nested loops. Figure 2a shows an outline of a parallel ERI evaluation. The outermost loop is distributed over DDI processes (lines 1 and 2) using either SLB or DLB, while inner loops are processed by OpenMP threads (lines 3-5). The inner loops are usually merged into a single flat loop for a more efficient work distribution. In OpenMP, rectangular loops can automatically be merged using the collapse clause (e.g., collapse(2) in Line 3).

**(a)**

```
1  ! Distribute over DDI compute processes
2  DO ISH = 1,NSHELL
3  !$omp parallel do collapse(2) schedule(dynamic)
4      DO JSH = 1,ISH
5      DO KSH = 1,ISH
6          LMOD = KSH
7          IF(KSH == ISH) LMOD = JSH
8          DO LSH = 1,LMOD
9              CALL ERI(ISH,JSH,KSH,LSH)
10         ENDDO
11     ENDDO
12     ENDDO
13 ENDDO
```

**(b)**

```
1  ! Single layer loop
2  DO II=1,NTASKS
3      CALL TASK(II)
4  ENDDO
5
6  ! Distribute over DDI compute processes
7  DO ICHUNK=1,NCHUNKS
8  !$omp parallel do schedule(dynamic)
9      DO II=NTASK_START, NTASK_END
10         CALL TASK(II)
11     ENDDO
12 ENDDO
```

Figure 2. Two-level DDI/OpenMP parallelization implementation for (a) nested loop and (b) single loop in lines 2-4 is replaced by lines 6-12. The DDI parallelization is hidden (shown schematically as comments in green color). The OpenMP parallelization is shown explicitly in red color.

One-layer loops can also be treated in a similar approach by dividing the loop into chunks and distributing chunks over DDI processes. Each chunk is then computed by OpenMP threads. Figure 2b shows a transformation of a single loop in DDI/OpenMP. The original single-layer loop is in lines 1-4, and the transformed loop is in lines 6-12.

In the transformed loop, NTASKS iterations are split into NCHUNKS, so that each chunk contains a number of iterations from NTASK_START to NTASK_END. The transformed loop becomes a regular nested loop. In DDI/OpenMP, the outer NCHUNKS is distributed over DDI processes (Lines 6 and 7), and the inner loop is parallelized by OpenMP (Lines 8 and 9).

In DDI/OpenMP, there are two layers of load balancing. The load balancing at the DDI level is chosen by BALTYP in $SYSTEM. The load balancing in OpenMP can be static, dynamic, and guided (the latter means dynamic with an automatically adjusted chunk size). The dynamic schedule (lines 3 and 8 in Figure 2a and 2b, respectively) is usually the most efficient scheme. When an OpenMP schedule is not hardwired in the code, the default is usually the static scheme. The OpenMP schedule can also be controlled by an environmental variable OMP_SCHEDULE set in the running script. The OpenMP parallelization in GAMESS is summarized in Table 3.

**Table 3. Methods in GAMESS parallelized with DDI/OpenMP**

| Method | References |
|---|---|
| Hartree-Fock | 45 |
| Density functional theory (DFT) | 46 |
| Resolution of the identity MP2 (RI-MP2) | 47,48 |
| Resolution of the identity coupled-cluster (RI-CC) | 49 |
| Polarizable continuum model (PCM) | 50 |
| Effective fragment potential (EFP) | 15 |

## 2.4 Three-level parallelizations

### 2.4.1 *GDDI/3*

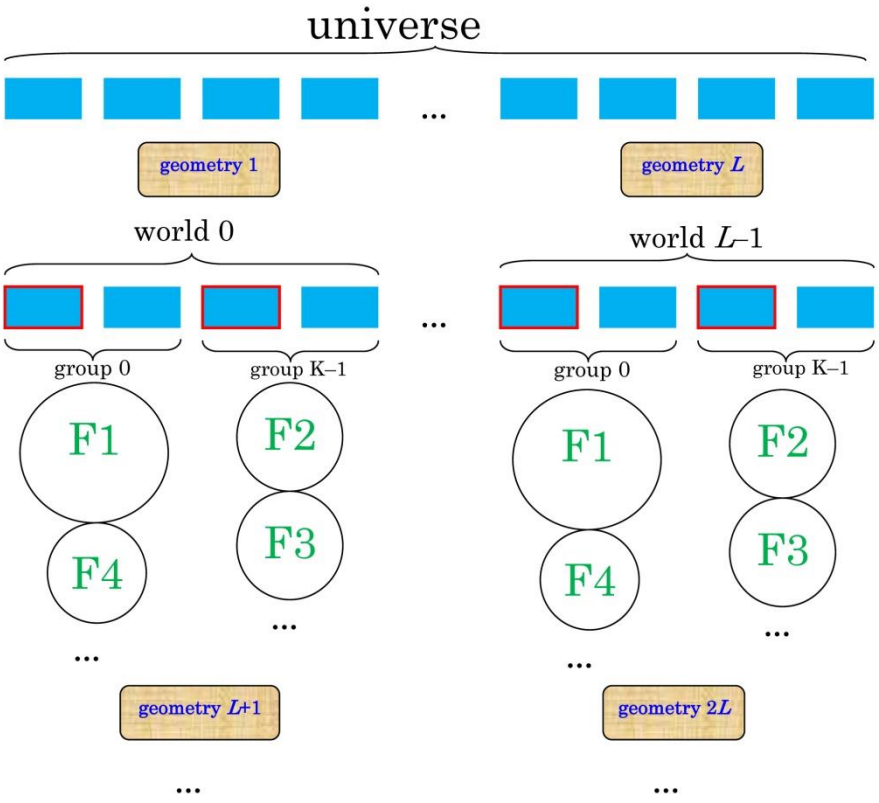


Figure 3. 3-level parallelization in GDDI/3 for a semi-numerical FMO Hessian. GDDI/3 divides all nodes (universe) into $L$ worlds, each of which is further divided into $K$ groups. Workers are shown as blue boxes. Workers whose lowest rank in each group are circled in red. Fragments tasks assigned to groups are labeled as F$I$.

A general GDDI/$n$ model for an $n$-level parallelization was developed[30] for QM problems of appropriate granularity. In practice, $n$=2 are $n$=3 are commonly used. In GDDI/3, the whole set of nodes is referred to as the universe, divided into $N$

GDDI/2-like worlds, which are further split into *M* groups using NGROUP=*N* NSUBGR=*M*.

A three-level parallelization (Figure 3) is useful for calculations with an appropriate structure of tasks. Their applications are listed in Table 4. It should be noted that a semi-numerical FMO Hessian requires recalculating the whole system for each shifted atom, because of the many-body polarization effects. In particular, the use of GDDI/3 is essential for computations of minimum energy crossing (MEX) of two spin surfaces with FMO. The GDDI/3 automatically resolves the problem of storing electronic states for two spin multiplicities of each fragment, which results in an unstable SCF convergence in the GDDI/2 model because the densities are overwritten when multiplicity changes.

**Table 4. Three-level parallelization of QM methods using GDDI/3.**

| FMO Method | Upper level | Middle level | Lower level | Refs |
|---|---|---|---|---|
| semi-numerical Hessians | shifted molecules | fragments | integral batches | 30 |
| minimum energy crossing | spin states | fragments | integral batches | 51 |

Similarly to the use of GDDI/2 for a one-level parallelization, the GDDI/3 model can be reduced to a two-level parallelization, with group sizes of 1. Such reduced model is sometimes employed in semi-numerical FMO-DFTB Hessians.

### 2.4.2 *GDDI/2+OpenMP*

OpenMP can be combined with GDDI/2 providing a way to do a three-level parallelization, as summarized in Table 5. At the highest level, fragments are distributed over groups. QM computations of each fragment are carried out by threads spawned by compute processes in these groups. Work distribution within each group of compute processes and threads is similar to the description in section 2.3.3.

In GDDI/2+OpenMP, the logical node size (Table 1) is usually set to one, i.e., each group contains a single compute process). The number of threads spawned by each compute process can be set by users using an environmental variable OMP_NUM_THREADS.

**Table 5. Three-level parallelization of QM methods using GDDI/2+OpenMP.**

| Method | Upper level | Middle level | Low level | References |
|---|---|---|---|---|
| FMO-RHF | fragments | Outermost loop[a] | Inner loops | 50 |
| FMO-RI-MP2 | fragments | Outermost loop (or chunks of loop) | Inner loops (or tasks in a chunk | 30,47,48 |
| FMO-DFT | fragments | Chunks of grid | Points in a chunk | 46 |

[a] Loops are over atomic orbitals in integrals of a fragment.

### 2.5 Four-level parallelization GDDI/3+OpenMP

GDDI/3 can be combined with OpenMP, resulting in a 4-level parallelization, developed in this work (Figure 4).

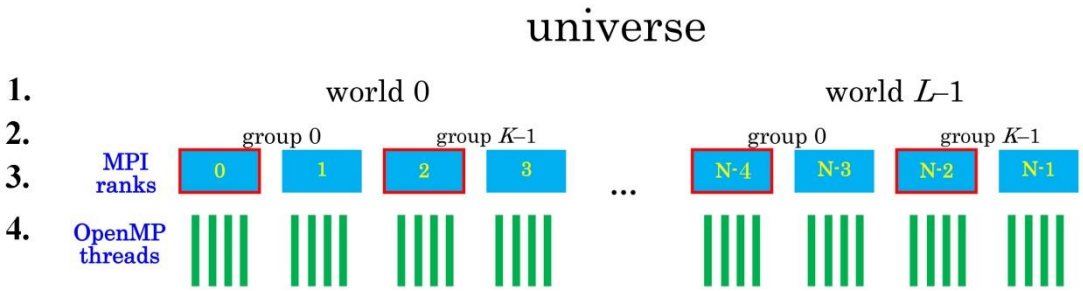


Figure 4. Schematic 4-level parallelization in GDDI/3+OpenMP.

As a demonstration, GDDI/3+OpenMP is applied to a semi-numerical Hessian computed with FMO-RI-MP2/cc-pVDZ for a cluster of 64 water molecules $(H_2O)_{64}$, divided into 64 fragments (one water molecule per fragment). Water clusters continue to fascinate us,[52,53,54] and they can be used to describe water-air interface effects.[55] The water cluster geometry was optimized using FMO2-DFTB3/D3(BJ)[43,56] method with 3ob parameters.[57] The semi-numerical Hessian was calculated by doubly differentiating analytic energy gradients,[48,58] which require $64\times3\times3\times2=1152$ single point gradients.

Calculations were carried out using 256, 512, and 1024 Knights Landing (KNL) nodes on the Theta supercomputer at Argonne National Laboratory. One KNL node has 64 physical cores.

The adopted grouping strategies and the corresponding wall-clock timings are summarized in Table 6. In order to define the numbers of worlds and groups, whose product is equal to the number of DDI compute processes, physical nodes were divided into logical nodes. For instance, each 64-core KNL node can be divided into 16 logical

nodes, with each with one DDI rank allotted 4 cores. Two threads are spawned for each physical core introducing a total of 8 threads for each DDI rank. With 256 KNL nodes, the logical node setup results in 256×16=4096 DDI ranks. These ranks can be divided into 64 worlds, each world has 64 groups (64×64=4096).

With this setup, 64 shifted geometries are computed simultaneously (one per world). For each geometry, the gradient for a fragment is calculated using one DDI rank assigned 8 OpenMP threads. The total number of threads used for this calculation is 32,768.

**Table 6. 4-level parallelization strategy.**

| Nodes | Worlds per universe | Groups per world [a] | Ranks per group [a] | Threads per rank | Threads, total |
|---|---|---|---|---|---|
| 256 | 64 | 64,64 | 1,1 | 8 | 32,768 |
| 512 | 128 | 64,64 | 1,1 | 8 | 65,536 |
| 1024 | 128 | 64,128 | 2,1 | 8 | 131,072 |

[a] Listed separately for fragments and their pairs.

The wall-clock timings and speed-ups relative to the 256-node calculations are shown in Figure 5. The parallel efficiency of about 88% is obtained on 131,072 OpenMP threads. Some loss of performance (~12%) is attributed to the small size of

fragments, where the amount of work does not scale sufficiently well when the number of threads is too large.

Comparing the original 3-level parallelization with the 4-level scheme developed in this work, a speed-up of 4.6 is obtained (Table 7). It is attributed to the efficient use of the extra level of parallelization. It may be expected that some improvement could be obtained for both 3 and 4-level parallelizations by varying the parameters (the number of worlds and groups) not fully explored because of the limited CPU time allocation.

**Table 7. Comparison of 3 and 4-level parallelization schemes on 256 nodes of Theta, for $(H_2O)_{64}$ computed with FMO2-RI-MP2/cc-pVDZ.**

| Parallelization level | worlds | Groups per world[a] | Ranks per group[a] | Threads per rank | Timing, sec |
|---|---|---|---|---|---|
| 3 (GDDI/2+OpenMP) | - | 64,256 | 4,1 | 64 | 28178.1[b] |
| 4 (GDDI/3+OpenMP) | 64 | 64,64 | 1,1 | 8 | 6015.9 |

[a] Listed separately for fragments and pairs.

[b] Estimated for 1152 offsets from 339 computed offsets that took 8292 sec.

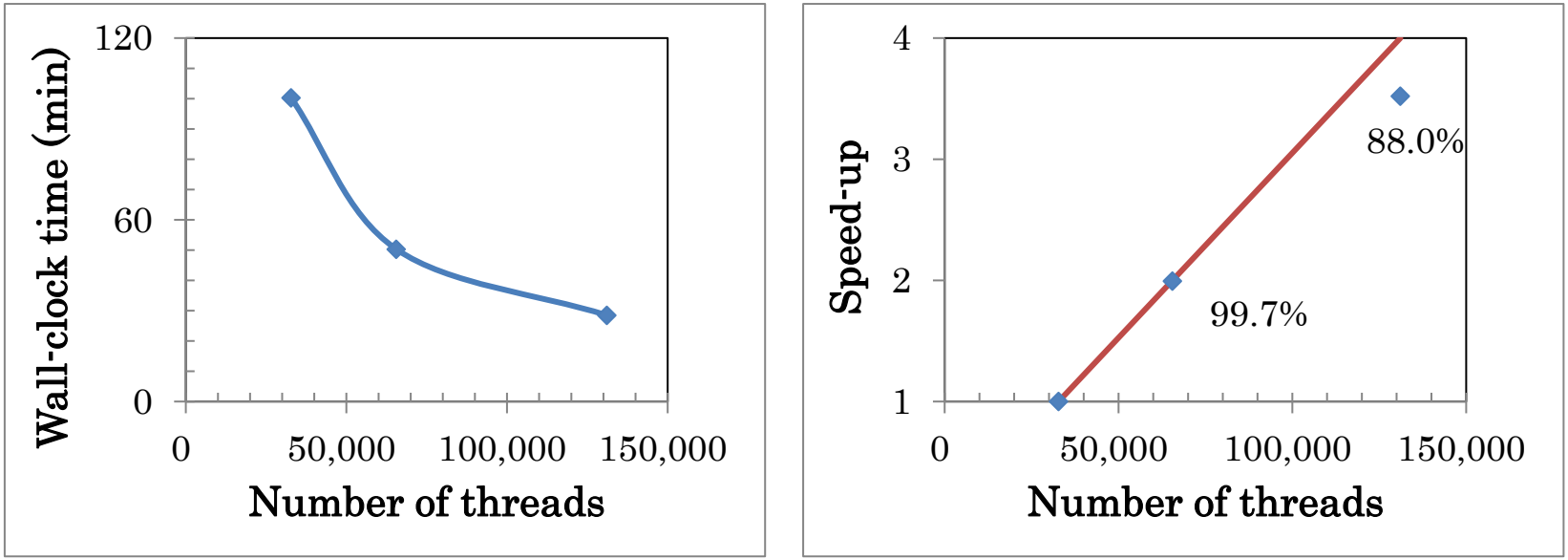


Figure 5. Wall-clock time and parallel speed-up for a semi-numerical Hessian of $(H_2O)_{64}$ computed with FMO-RI-MP2/cc-pVDZ on Theta (the ideal scaling is shown as the red line).

### 2.6 Multi-level parallelization on GPUs

GPU can be used for parallelizing QM tasks using GAMESS interfaced with libCChem based on CUDA.[15] In the sense of multi-level parallelization, such MPI+CUDA usage may be characterized as a 2-level model (1-level if MPI is not relevant). A number of QM calculations in GAMESS can be parallelized in this way.[15]

Another possibility is to use an OpenMP type of offloading of calculations to GPUs.[15] In the sense of a multi-level parallelization, such usage can be characterized as other examples of OpenMP usage described above.

The use of GDDI/2-like parallelization based on MPI combined with GPUs has been recently reported (GDDI/2+CUDA),[59] which may be characterized as a 3-level parallelization, similar conceptually to GDDI/2+OpenMP described above.

### 2.7 Comparative discussion of multi-level parallelizations

An important comparison is that of GDDI/(*n*-1)+OpenMP with GDDI/*n*. The former has several important advantages and some disadvantages. Namely, GDDI/(*n*-1)+OpenMP can be used for more types of calculations than GDDI/*n* because there are many general QM methods with sufficient granularity for *n*=2 (Table 5), but only two specialized tasks for *n*=3 (Table 4). DDI/OpenMP can be used for almost any QM method in GAMESS, whereas GDDI/2+OpenMP can be employed for a limited set of QM

methods. The use of OpenMP reduces the memory footprint because the number of executables loaded in memory is determined by the number of DDI ranks.

There is another very important advantage of using OpenMP. With it, the number of data servers is greatly reduced, freeing up CPU for more computations. For example, consider a DDI run on 2 nodes consisting of 8 cores, where 16 compute processes and 16 data servers are executed. For DDI/OpenMP, executed as 1 DDI process per node, there are only 2 data servers, a reduction by the factor of 8. Although some early attempts to exclude data servers were successful,[20] they continue to be employed.

The disadvantages of using OpenMP are mainly limited to the lack of OpenMP parallelization for many methods in GAMESS. When OpenMP code is not available, only the parallelization among DDI processes is effective, resulting in a poor usage of CPU for these steps. In addition, some methods in GAMESS coded with older programming models may simply fail to work when compiled with OpenMP.

**2.8. Multi-level parallelization of fragment-based many-body expansions**

The parallelization of FMO[60,32,61] or EFMO[33,34] is challenging due to the use of many-body expansions (MBE)[62,63] in these and many other fragment-based methods,7 resulting in the need to compute conglomerates of fragments (pairs, triples, quadruples,

etc). Each step in these calculations has a very different number of tasks. In MBE truncated at 3-body terms, the total energy $E$ is contributed from the energy of fragments ( $E_I'$ ), their pairs ( $\Delta E_{IJ}$ ), and, optionally, trimers ( $\Delta E_{IJK}$ ) :

$$E = \sum_{I=1}^{N} E_I' + \sum_{I>J}^{N} \Delta E_{IJ} + \sum_{I>J>K}^{N} \Delta E_{IJK} \qquad (1)$$

Even though the number of pairs and trimers can be linearized by the dimer and trimer cutoff approximations RESDIM [64] and RITRIM[31], respectively, the multiplicative prefactors determining the number of fragments, dimers and triples as a function of the number of fragments $N$ are different. Consequently, the number of tasks and their sizes are different for fragments, their pairs, and triples, which require a different parallelization strategy for each of these steps.

In practice, the number of groups in GDDI/2 can be set via the NGRFMO keyword in $FMOPRP, listing the number of groups for each step in the calculations, replacing a single group count NGROUP in $GDDI. Likewise, a manual group definition, as used in SDLB, has to be done separately for each step, controlled by MANNOD in $FMOPRP as a continuous list for all steps, replacing a single list MANNOD in $GDDI.

### 2.9. Massively parallel calculations on supercomputers

Multi-level parallelization in GAMESS has a high efficiency suitable for massively parallel calculations on supercomputers. Some of these calculations on CPUs are summarized in Table 8. With a GDDI/2+CUDA-like parallelization, FMO-RI-MP2 calculations on Summit were done on 27,600 GPUs with a spectacular efficiency.[59]

**Table 8. Use of multi-level parallelization on CPU-based supercomputers.**

| Supercomputer | CPU cores used | parallelization | Method | Reference |
|---|---|---|---|---|
| K computer | 196,608 | GDDI/2+OpenMP | FMO-RI-MP2 | 30 |
| Intrepid | 131,072 | GDDI/2 | FMO-MP2 | 65 |
| Mira | 262,144 | GDDI/2 | FMO-RHF | 66 |
| Stampede2 | 8,704 | GDDI/2+OpenMP | FMO-RHF | 50 |
| Theta | 131,072 | GDDI/2+OpenMP | FMO-RHF | 50 |
| Theta | 65,536 | GDDI/3+OpenMP | FMO-RI-MP2 | This work |

In addition to big computers, GAMESS can be used for big data applications. Several all-atom QM calculations of systems containing more than one million atom calculations were reported.[67,68] Such applications became possible as a result of combing many efforts in improving the efficiency of GAMESS: file-less execution and linearization of required memory.[27]

## 3. Application

The FMO-MP2 Hessian results obtained with a 4-level GDDI/3+OpenMP parallelization on Theta as described above were processed with the partition analysis of vibrational energy (PAVE).[69,70] The total vibrational zero-point energy (ZPE) was decomposed into the contributions of individual water molecules. The results are shown in Figure 6. It can be seen that there is some variation in ZPE depending on the local environment in the water droplet.

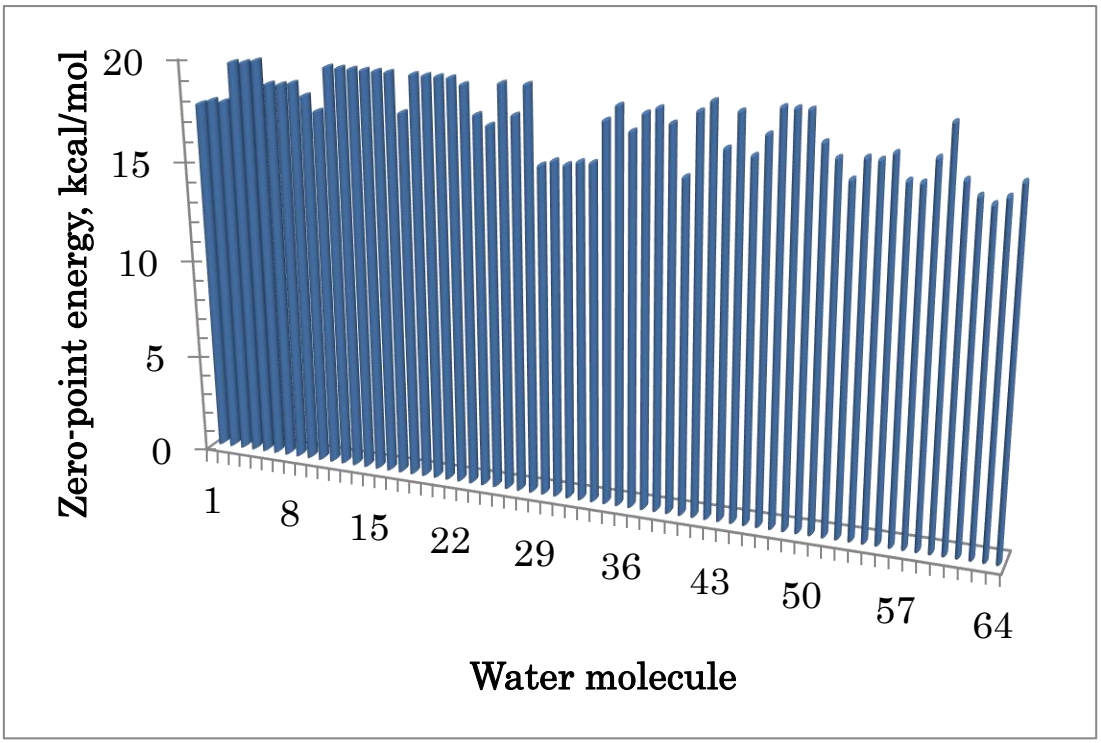


Figure 6. Zero point energy of individual water molecules in $(H_2O)_{64}$ at the level of FMO-RI-MP2/cc-pVDZ, obtained with PAVE.

## 4. Conclusions and outlook

Parallelization strategies up to 4 levels in complexity are developed for use in large-scale QM calculations. These techniques have been shown to improve parallel efficiency on petascale computers. Load balancing requires special care and must be handled separately at each level. Various approaches to static and dynamic load balancings have been proposed and implemented.

Owing to advances in computational science outlined above, QM calculations can now be routinely applied to large molecular systems, such as protein-ligand complexes.[71,72] Efficient parallel solutions are crucial for advancing drug discovery[73,74] and material science[75,76] applications. Thermochemical properties obtained from Hessian calculations can improve the estimation of binding energy and reaction barriers.[70] Semi-analytical Hessians are needed when analytic Hessians are not available, for example, for FMO-TDDFT[77] or FMO-DFTB with periodic boundary conditions.[78]

The multi-level parallelization techniques, illustrated on QM methods implemented in GAMESS, can be applied to other scientific applications, facing similar performance problems. In particular, the 4-level parallelization is a promising technique that can harness high efficiency on exascale computers such as Aurora.

## Acknowledgements

We thank Argonne Leadership Computing Facility for a time allocation on Theta and Dr. Colleen Bertoni for assistance in using the resources. BQP thanks the Department of Energy Exascale Computing Project (17-SC-20-SC) for support.

## Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## References

[1] A. V. Akimov, and O. V. Prezhdo, "Large-scale computations in chemistry: a bird's eye view of a vibrant field," Chem. Rev. **115**, 5797−5890 (2015).
[2] K. Kowalski, R. Bair, N. P. Bauman, J. S. Boschen, E. J. Bylaska, J. Daily, W. A. de Jong, T. Dunning Jr., N. Govind, R. J. Harrison, M. Keçeli, K. Keipert, S. Krishnamoorthy, S. Kumar, E. Mutlu, B. Palmer, A. Panyala, B. Peng, R. M. Richard, T. P. Straatsma, P. Sushko, E. F. Valeev, M. Valiev, H. J. J. van Dam, J. M. Waldrop, D. B. Williams-Young, C. Yang, M. Zalewski, and T. L. Windus, "From NWChem to NWChemEx: Evolving with the computational chemistry landscape," Chem. Rev. **121**, 4962–4998 (2021).
[3] T. Nakajima, M. Katouda, M. Kamiya, Y. Nakatsuka, "NTChem: A high-performance software package for quantum molecular simulation," Int. J. Quant. Chem. **115**, 349–359 (2015).
[4] H. Kitoh-Nishioka, H. Umeda, and Y. Shigeta, "Open-architecture program of fragment molecular orbital method for massive parallel computing (OpenFMO) with GPU acceleration," in *Recent advances of the fragment molecular orbital method*, Y. Mochizuki, S. Tanaka, K. Fukuzawa (Eds), Springer, Singapore, 2021, pp. 77–90.
[5] K. Ishimura, "Development of massively parallel quantum chemistry program SMASH," AIP Conf. Proc. **1702**, 090053 (2015).
[6] F. Aquilante, J. Autschbach, A. Baiardi, S. Battaglia, V. A. Borin, L. F. Chibotaru, I. Conti, L. De Vico, M. Delcey, I. F. Galván, N. Ferré, L. Freitag, M. Garavelli, X. Gong, S. Knecht, E. D. Larsson, R. Lindh, M. Lundberg, P. Å. Malmqvist, A. Nenov, J. Norell, M. Odelius, M. Olivucci, T. B. Pedersen, L. Pedraza-González, Q. M. Phung, K. Pierloot, M. Reiher, I. Schapiro, J. Segarra-Martí, F. Segatta, L. Seijo, S. Sen, D.-C.

Sergentu, C. J. Stein, L. Ungur, M. Vacher, A. Valentini, and V. Veryazov, "Modern quantum chemistry with [Open]Molcas," J. Chem. Phys. **152**, 214117 (2020).

[7] M. S. Gordon; S. R. Pruitt, D. G. Fedorov, and L. V. Slipchenko, "Fragmentation methods: a route to accurate calculations on large systems," Chem. Rev. **112**, 632–672 (2012).

[8] D. G. Fedorov, R. M. Olson, K. Kitaura, M. S. Gordon, and S. Koseki, "A new hierarchical parallelization scheme: generalized distributed data interface (GDDI), and an application to the fragment molecular orbital method (FMO)," J. Comput. Chem. **25**, 872–880 (2004).

[9] Y. Mochizuki, K. Yamashita, T. Murase, T. Nakano, K. Fukuzawa, K. Takematsu, H. Watanabe, and S. Tanaka, "Large scale FMO-MP2 calculations on a massively parallel-vector computer," Chem. Phys. Lett. **457**, 396–403 (2008).

[10] Y. Mochizuki, T. Nakano, K. Sakakura, Y. Okiyama, H. Watanabe, K. Kato, Y. Akinaga, S. Sato, J. Yamamoto, K. Yamashita, T. Murase, T. Ishikawa, Y. Komeiji, Y. Kato, N. Watanabe, T. Tsukamoto, H. Mori, K. Okuwaki, S. Tanaka, A. Kato, C. Watanabe, and K. Fukuzawa. "The ABINIT-MP program," In *Recent advances of the fragment molecular orbital method*, Y. Mochizuki, S. Tanaka, K. Fukuzawa (Eds), Springer, Singapore, 2021, pp. 53–67.

[11] K. Takaba, C. Watanabe, A. Tokuhisa, Y. Akinaga, B. Ma, R. Kanada, M. Araki, Y. Okuno, Y. Kawashima, H. Moriwaki, N. Kawashita, T. Honma, K. Fukuzawa, and S. Tanaka, "Protein-ligand binding affinity prediction of cyclin-dependent kinase-2 inhibitors by dynamically averaged fragment molecular orbital-based interaction energy," J. Comput. Chem. **43**, 1362–1371 (2022).

[12] S. Tanaka, S. Tokutomi, R. Hatada, K. Okuwaki, K. Akisawa, K. Fukuzawa, Y. Komeiji, Y. Okiyama, and Y. Mochizuki, "Dynamic cooperativity of ligand-residue interactions evaluated with the fragment molecular orbital method," J. Phys. Chem. B **125**, 6501–6512 (2021).

[13] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery, "General atomic and molecular electronic structure system," J. Comput. Chem. **14**, 1347 (1993).

[14] M. S. Gordon and M. W. Schmidt, "Advances in electronic structure theory: GAMESS a decade later," in *Theory and applications of computational chemistry the first forty years*, C. Dykstra, G. Frenking, K. Kim, G. Scuseria (Eds.), Elsevier Science: Amsterdam, 2005; pp 1167–1189.

[15] G. M. J. Barca, C. Bertoni, L. Carrington, D. Datta, N. D. Silva, J. E. Deustua, D. G. Fedorov, J. R. Gour, A. O. Gunina, E. Guidez, T. Harville, S. Irle, J. Ivanic, K. Kowalski, S. S. Leang, H. Li, W. Li, J. J. Lutz, I. Magoulas, J. Mato, V. Mironov, H. Nakata, B. Q. Pham, P. Piecuch, D. Poole, S. R. Pruitt, A. P. Rendell, L. B. Roskop, K. Ruedenberg, T. Sattasathuchana, M. W. Schmidt, J. Shen, L. Slipchenko, M. Sosonkina, V. Sundriyal, A. Tiwari, J. L. G. Vallejo, B. Westheimer, M. Wloch, P. Xu, F. Zahariev, and M. S. Gordon, "Recent developments in the general atomic and molecular electronic structure system," J. Chem. Phys. **152**, 154102 (2020).

[16] B. Q. Pham, M. Alkan, and M. S. Gordon, "Porting fragmentation methods to GPU using an OpenMP API: offloading the Fock build for low angular momentum functions," submitted.

[17] M. Alkani, B. Q. Pham, J. R. Hammond, and M. S. Gordon, "Enabling Fortran standard parallelism in GAMESS for accelerated quantum chemistry calculations," in preparation.
[18] S. Bak, C. Bertoni, S. Boehm, R. Budiardja, B. M. Chapman, J. Doerfert, M. Eisenbach, H. Finkel, O. Hernandez, J. Huber, S. Iwasaki, V. Kale, P. R. C. Kent, J. Kwack, M. Lin, P. Luszczek, Y. Luo, B. Pham, S. Pophale, K. Ravikumar, V. Sarkar, T. Scogland, S. Tian, and P. K. Yeung, "OpenMP application experiences: Porting to accelerated nodes," Parall. Comput. **109**, 102856 (2022).
[19] B. Chapman, B. Pham, C. Yang, C. Daley, C. Bertoni, D. Kulkarni, D. Oryspayev, E. D'Azevedo, J. Doerfert, K. Zhou, K. Ravikumar, M. Gordon, M. D. Ben, M. Lin, M. Alkan, M. Kruse, O. Hernandez, P. K. Yeung, P. Lin, P, Xu, S. Pophale, T. Sattasathuchana, V. Kale, W. Huhn, and Y. He, "Outcomes of OpenMP Hackathon: OpenMP Application Experiences with the Offloading Mode (Part I)," Proc. of IWOMP 2021, Bristol, UK, pp. 67–80 (2021).
[20] A. Asadchev, B. M. Bode, and M. S. Gordon, "Performance of electronic structure calculations on BG/L and XT4 computers," J. Comput. Theor. Nanosc. **6**, 1290–1296 (2009).
[21] Y. Alexeev, A. Mahajan, S. Leyffer, G. Fletcher, and D. G. Fedorov, "Heuristic static load-balancing algorithm applied to the fragment molecular orbital method," Proc. Supercomputing 2012, IEEE Computer Society, Salt Lake City, 2012.
[22] S. K. Talamudupula, M. Sosonkina, A. Gaenko, and M. W. Schmidt, "Fragment molecular orbital method adaptations for heterogeneous computing platforms," Proc. Comput. Sc. **9**, 489–497 (2012).
[23] Y. Ma, Z. Li, X. Chen, B. Ding, N. Li, T. Lu, B. Zhang, B. Suo, and Z. Jin, "Machine-learning assisted scheduling optimization and its application in quantum chemical calculations," J. Comput. Chem., in press. (2023), https://doi.org/10.1002/jcc.27075
[24] T. Ikegami, T. Ishida, D. G. Fedorov, K. Kitaura, Y. Inadomi, H. Umeda, M. Yokokawa, and S. Sekiguchi, "Full electron calculation beyond 20,000 atoms: ground electronic state of photosynthetic proteins," Proc. of Supercomputing 2005, IEEE Computer Society, 2005.
[25] D. G. Fedorov and K. Kitaura, "Multiconfiguration self-consistent-field theory based upon the fragment molecular orbital method," J. Chem. Phys. 122, 054108 (2005).
[26] M. Chiba, D. G. Fedorov, and K. Kitaura, "Time-dependent density functional theory based upon the fragment molecular orbital method," J. Chem. Phys. **127**, 104108 (2007).
[27] Fedorov, D. G. *Complete guide to the fragment molecular orbital method in GAMESS*, World Scientific, Singapore, 2023.
[28] G. D. Fletcher, M. W. Schmidt, B. M. Bode, and M. S. Gordon, "The distributed data interface in GAMESS," Comput. Phys. Commun. **128**, 190–200 (2000).
[29] K. Ishimura, K. Kuramoto, Y. Ikuta, and S. Hyodo, "MPI/OpenMP hybrid parallel algorithm for Hartree−Fock calculations," J. Chem. Theory Comput. **6**, 1075–1080 (2010).
[30] V. A. Mironov, Y. Alexeev, D. G. Fedorov, H. Umeda, S. Pruitt, A. Gaenko, and M. S. Gordon, "Multi-level parallelization of the fragment molecular orbital method in GAMESS," in *Recent advances of the fragment molecular orbital method*, Y.

Mochizuki, S. Tanaka, K. Fukuzawa (Eds), Springer, Singapore, 2021, pp. 601–616.
[31] D. G. Fedorov and K. Kitaura, "The importance of three-body terms in the fragment molecular orbital method," J. Chem. Phys. **120**, 6832–6840 (2004).
[32] D. G. Fedorov, "The fragment molecular orbital method: theoretical development, implementation in GAMESS and applications," WIREs: Comput. Mol. Sc. **7**, e1322 (2017).
[33] C. Steinmann, D. G. Fedorov, and J. H. Jensen, "Effective fragment molecular orbital method: a merger of the effective fragment potential and fragment molecular orbital methods," J. Phys. Chem. A **114**, 8705–8712 (2010).
[34] C. Bertoni and M. S. Gordon, "Analytic gradients for the effective fragment molecular orbital method," J. Chem. Theory Comput. **12**, 4743–4767 (2016).
[35] D. G. Fedorov, Y. Sugita, and C. H. Choi, "Efficient parallel implementations of QM/MM-REMD (quantum mechanical/molecular mechanics-replica-exchange MD) and umbrella sampling: isomerization of $H_2O_2$ in aqueous solution," J. Phys. Chem. B **117**, 7996–8002 (2013).
[36] S. Ito, D. G. Fedorov, Y. Okamoto, and S. Irle, "Implementation of replica-exchange umbrella sampling in GAMESS," Comp. Phys. Comm. **228**, 152–162 (2018).
[37] T. Taketsugu, K. Yagi, and M. S. Gordon, "A vibrational analysis of the 7-azaindole-water complex: Anharmonicities using the quartic force field," Int. J. Quant. Chem. **104**, 758–772 (2005).
[38] B. Njegic and M. S. Gordon, "Predicting accurate vibrational frequencies for highly anharmonic systems," J. Chem. Phys. **129**, 164107 (2008).
[39] P. Xu, T. Sattasathuchana, E. Guidez, S. P. Webb, K. Montgomery, H. Yasini, I. F. M. Pedreira, and M. S. Gordon, "Computation of host–guest binding free energies with a new quantum mechanics based mining minima algorithm," J. Chem. Phys. **154**, 104122 (2021).
[40] M. Katouda, M. Kobayashi, H. Nakai, and S. Nagase, "Two-level hierarchical parallelization of second-order Møller–Plesset perturbation calculations in divide-and-conquer method," J. Comput. Chem. **32**, 2756–2764 (2011).
[41] A. Devarajan, T. L. Windus, and M. S. Gordon, "Implementation of dynamical nucleation theory effective fragment potentials method for modeling aerosol chemistry," J. Phys. Chem. A **115**, 13987–13996 (2011).
[42] A. Devarajan, A. Gaenko, M. S. Gordon, and T. L. Windus, "Nucleation using the effective fragment potential and two-level parallelism," in *Fragmentation: toward Accurate calculations on complex molecular systems.* M. S. Gordon (Ed.), Wiley, Hoboken, 2017, pp. 209–226.
[43] Y. Nishimoto, D. G. Fedorov, and S. Irle, "Density-functional tight-binding combined with the fragment molecular orbital method," J. Chem. Theory Comput. **10**, 4801–4812 (2014).
[44] D. G. Fedorov, "Parametrized quantum-mechanical approaches combined with the fragment molecular orbital method," J. Chem. Phys. **157**, 231001 (2022).
[45] V. Mironov, A. Moskovsky, M. D'Mello, Y. Alexeev, "An efficient MPI/OpenMP parallelization of the Hartree–Fock–Roothaan method for the first generation of Intel® Xeon Phi™ processor architecture," Int. J. High Perf. Comput. Appl. **33**, 212–224 (2019).

[46] V. Mironov, Y. Alexeev, and D. G. Fedorov, "MPI+OpenMP parallelization of DFT method in GAMESS," poster presentation at Supercomputing 2019, Denver. URL: https://sc19.supercomputing.org/proceedings/tech_poster/poster_files/rpost211s2-file2.pdf (Accessed on January 31, 2023).
[47] B. Q. Pham and M. S. Gordon, "Hybrid distributed/shared memory model for the RI-MP2 method in the fragment molecular orbital framework," J. Chem. Theory Comput. **15**, 5252–5258 (2019).
[48] B. Q. Pham and M. S. Gordon, "Development of the FMO/RI-MP2 fully analytic gradient using a hybrid-distributed/shared memory programming model," J. Chem. Theory Comp. **16**, 1039–1054 (2020).
[49] D. Datta and M. S. Gordon, "A massively parallel implementation of the CCSD(T) method using the resolution-of-the-identity approximation and a hybrid distributed/shared memory parallelization model," J. Chem. Theory Comput. **17**, 4799–4822 (2021).
[50] V. Mironov, Y. Alexeev, and D. G. Fedorov, "Multithreaded parallelization of the energy and analytic gradient in the fragment molecular orbital method," Int. J. Quant. Chem. **119**, e25937 (2019).
[51] D. S. Kaliakin, D. G. Fedorov, Y. Alexeev, and S. A. Varganov, "Locating minimum energy crossings of different spin states using the fragment molecular orbital method," J. Chem. Theory Comput. **15**, 6074–6084 (2019).
[52] S. R. Pruitt, S. S. Leang, P. Xu, D. G. Fedorov, and M. S. Gordon, "Hexamers and witchamers: Which hex do you choose?" Comp. Theor. Chem. **1021**, 70–83 (2013).
[53] A. Rakshit, P. Bandyopadhyay, J. P. Heindel, and S. S. Xantheas, "Atlas of putative minima and low-lying energy networks of water clusters $n$ = 3–25," J. Chem. Phys. **151**, 214307 (2019).
[54] A. Gijón and E. R. Hernández, "Quantum simulations of neutral water clusters and singly-charged water cluster anions," Phys. Chem. Chem. Phys. **24**, 14440–14451 (2022).
[55] R. Kusaka, S. Nihonyanagi, T. Tahara, "The photochemical reaction of phenol becomes ultrafast at the air–water interface," Nat. Chem. **13**, 306–311 (2021).
[56] Y. Nishimoto, H. Nakata, D. G. Fedorov, and S. Irle, "Large-scale quantum-mechanical molecular dynamics simulations using density-functional tight-binding combined with the fragment molecular orbital method," J. Phys. Chem. Lett. **6**, 5034–5039 (2015).
[57] M. Gaus, A. Goez, M. Elstner, "Parametrization and benchmark of DFTB3 for organic molecules," J. Chem. Theory Comput. **9**, 338–354 (2013).
[58] T. Nagata, D. G. Fedorov, K. Ishimura, K. Kitaura, "Analytic energy gradient for second-order Møller-Plesset perturbation theory based on the fragment molecular orbital method," J. Chem. Phys. **135**, 044110 (2011).
[59] G. Barca, C. Snowdon, J. Vallejo, F. Kazemian, A. P. Rendell, and M. S. Gordon, "Scaling correlated fragment molecular orbital calculations on Summit," Proc. Supercomputing 2022, IEEE Computer Society, Dallas, 2022, pp. 72–85.
[60] K. Kitaura, E. Ikeo, T. Asada, T. Nakano, and M. Uebayasi, "Fragment molecular orbital method: an approximate computational method for large molecules," Chem. Phys. Lett. **313**, 701–706 (1999).

[61] K. Fukuzawa and S. Tanaka, "Fragment molecular orbital calculations for biomolecules," Curr. Opin. Struct. Biol. **72**, 127–134 (2022).
[62] D. G. Fedorov, N. Asada, I. Nakanishi, and K. Kitaura "The use of many-body expansions and geometry optimizations in fragment-based methods," Acc. Chem. Res. **47**, 2846–2856 (2014).
[63] S. P. Veccham, J. Lee, and M. Head-Gordon, "Making many-body interactions nearly pairwise additive: the polarized many-body expansion approach," J. Chem. Phys. **151**, 194101 (2019).
[64] T. Nakano, T. Kaminuma, T. Sato, K. Fukuzawa, Y. Akiyama, M. Uebayasi, and K. Kitaura, "Fragment molecular orbital method: use of approximate electrostatic potential," Chem. Phys. Lett. **351**, 475–480 (2002).
[65] G. D. Fletcher, D. G. Fedorov, S. R. Pruitt, T. L. Windus, and M. S. Gordon, "Large-scale MP2 calculations on the Blue Gene architecture using the fragment molecular orbital method," J. Chem. Theory Comp. **8**, 75–79 (2012).
[66] S. R. Pruitt, H. Nakata, T. Nagata, M. Mayes, Y. Alexeev, G. Fletcher, D. G. Fedorov, K. Kitaura, and M. S. Gordon, "Importance of three-body interactions in molecular dynamics simulations of water demonstrated with the fragment molecular orbital method," J. Chem. Theory Comput. **12**, 1423–1435 (2016).
[67] Y. Nishimoto, D. G. Fedorov, and S. Irle, "Density-functional tight-binding combined with the fragment molecular orbital method," J. Chem. Theory Comput. **10**, 4801–4812 (2014).
[68] Y. Nishimoto and D. G. Fedorov, "Adaptive frozen orbital treatment for the fragment molecular orbital method combined with density-functional tight-binding," J. Chem. Phys. **148**, 064115 (2018).
[69] D. G. Fedorov, "Partitioning of the vibrational free energy," J. Phys. Chem. Lett. **12**, 6628–6633 (2021).
[70] T. Nakamura, T. Yokaichiya, and D. G. Fedorov, "Analysis of guest adsorption on crystal surfaces based on the fragment molecular orbital method," J. Phys. Chem. A **126**, 957–969 (2022).
[71] K. Okuwaki, K. Akisawa, R. Hatada, Y. Mochizuki, K. Fukuzawa, Y. Komeiji, and S. Tanaka, "Collective residue interactions in trimer complexes of SARS-CoV-2 spike proteins analyzed by fragment molecular orbital method," Appl. Phys. Expr. **15**, 017001 (2022).
[72] S. Monteleone, D. G. Fedorov, A. Townsend-Nicholson, M. Southey, M. Bodkin, and A. Heifetz, "Hotspot identification and drug design of protein-protein interaction modulators using the fragment molecular orbital method," J. Chem. Inf. Model. **62**, 3784-3799 (2022).
[73] A. Acharya, R. Agarwal, M. B. Baker, J. Baudry, D. Bhowmik, S. Boehm, K. G. Byler, S. Y. Chen, L. Coates, C. J. Cooper, O. Demerdash, I. Daidone, J. D. Eblen, S. Ellingson, S. Forli, J. Glaser, J. C. Gumbart, J. Gunnels, O. Hernandez, S. Irle, D. W. Kneller, A. Kovalevsky, J. Larkin, T. J. Lawrence, S. LeGrand, S.-H. Liu, J. C. Mitchell, G. Park, J. M. Parks, A. Pavlova, L. Petridis, D. Poole, L. Pouchard, A. Ramanathan, D. M. Rogers, D. Santos-Martins, A. Scheinberg, A. Sedova, Y. Shen, J. C. Smith, M. D. Smith, C. Soto, A. Tsaris, M. Thavappiragasam, A. F. Tillack, J. V. Vermaas, V. Q. Vuong, J. Yin, S. Yoo, M. Zahran, and L. Zanetti-Polzi, "Supercomputer-based ensemble docking drug discovery pipeline with application to covid-19," J. Chem. Inf.

Model. **60**, 5832–5852 (2020).
[74] H. Lim, H.-N. Jeon, S. Lim, Y. Jang, T. Kim, H. Cho, J.-G. Pan, and K. T. No, "Evaluation of protein descriptors in computer-aided rational protein engineering tasks and its application in property prediction in SARS-CoV-2 spike glycoprotein," Comput. Str. Biotechn. J. **20**, 788–798 (2022).
[75] C. Higuchi and K. Yoshizawa, "Energy decomposition analysis of the adhesive interaction between an epoxy resin layer and a silica surface," Langmuir **37**, 8417–8425 (2021).
[76] T. Nakamura and D. G. Fedorov, "The catalytic activity and adsorption in faujasite and ZSM-5 zeolites: the role of differential stabilization and charge delocalization," Phys. Chem. Chem. Phys. **24**, 7739–7747 (2022).
[77] H. Nakata and D. G. Fedorov, "Analytic gradient for time-dependent density functional theory combined with the fragment molecular orbital method," J. Chem. Theory Comput., in press, DOI: 10.1021/acs.jctc.2c01177.
[78] Y. Nishimoto and D. G. Fedorov, "The fragment molecular orbital method combined with density-functional tight-binding and periodic boundary conditions," J. Chem. Phys. **154**, 111102 (2021).