An efficient MPI/OpenMP parallelization of the Hartree-Fock method for the second generation of Intel® Xeon Phi™ processor

Yuri Alexeev[1], Vladimir Mironov[2], Kristopher Keipert[1,3], Michael D'Mello[4], Alexander Moskovsky[2], Mark Gordon[3]

[1]Argonne National Laboratory, [2]Moscow State University,, [3]Iowa State University, [4]Intel Corporation

Introduction

- Quantum Chemistry (QC):
  - Studies chemical systems by solving Schrodinger differential equation
  - The solution for Schrodinger equation is approximate
  - The input is the coordinates of atoms of the chemical system
  - The output is a wavefunction, which describes all chemical and physical properties of the chemical system

Introduction

- QC has a large variety of methods
- Computational aspects:
  - Many computational bottlenecks
  - Some methods do not use BLAS
  - Variety of data structures
  - Not all of methods can be easily and efficiently parallelized
  - Dependency between QC methods
  - Lack of optimized QC libraries
- High CPU and memory demands
  - Computational complexity of ab initio methods is $O(N^4)$ and higher
- Double precision is required for chemical accuracy

GAMESS

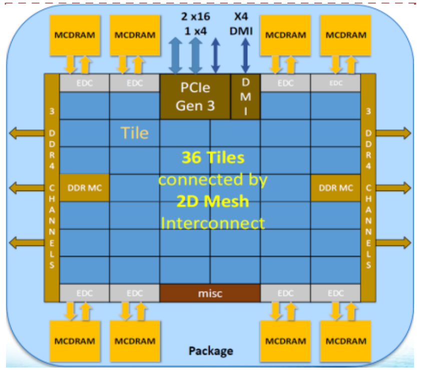
- Advantages:
  - One of the most popular QC programs (>10K user base)
  - Many QC methods are implemented in GAMESS
  - Free open-source code (custom license)
- Challenges:
  - Legacy code developed decades ago written in Fortran 77
  - Poor cache utilization
  - Hard to thread because of shared data structures (i.e. COMMON blocks)
- Our approach:
  - Rewrite and optimize code using modern languages
  - OpenMP parallelization and vectorization
  - Develop new algorithms
- Goal:
  - Optimization of GAMESS for massively parallel architectures

Benefits of using Intel Xeon Phi architecture

- Relatively straight forward porting and optimization of old Fortran code
- Xeon Phis are especially good for high level parallelism using large kernels
- Simultaneous performance improvement on both Xeon and Xeon Phi platforms

Theta supercomputer

- **System**:
  - Cray XC40 system installed in ALCF
  - 3,624 compute nodes/ 231,936 cores
  - 9.65 PetaFlops peak performance
- **Processor**:
  - Intel Xeon Phi, 2nd Generation 7230
  - 64 Cores
  - 1.3 GHz base
- **Memory**:
  - 16 GB MCDRAM per node
  - 192 GB DDR4-2400 per node
  - 754 TB of total system memory
- **Network**:
  - Cray Aries interconnect
  - Dragonfly network topology
- **Filesystems**:
  - Project directories: 10 PB Lustre file system
  - Home directories: GPFS



Chip
- 683 mm²
- 14 nm process
- 8 Billion transistors

Up to 72 Cores
- 36 tiles
- 2 cores per tile
- 3 TF per node

2D Mesh Interconnect
- Tiles connected by 2D mesh

On Package Memory
- 16 GB MCDRAM
- 8 Stacks
- 485 GB/s bandwidth

6 DDR4 memory channels
- 2 controllers
- up to 384 GB external DDR4
- 90 GB/s bandwidth

On Socket Networking
- Omni-Path NIC on package
- Connected by PCIe

Hartree-Fock (HF) method

- One of the first ab initio methods
- Basis for many other QC methods
- Self-consistent solution of the Hartree-Fock equations in matrix form:

$$\boldsymbol{FC} = \epsilon \boldsymbol{SC}$$

$\boldsymbol{F}$ – Fock matrix,
$\boldsymbol{C}$ – wavefunction expansion in the selected basis set,
$\boldsymbol{S}$ – matrix of the basis function overlap,
ϵ – vector of orbital energies

- Computational complexity is $\mathrm{O}(N^4)$
- HF requires calculation of N^4 two-electron repulsion integrals (ERIs)
- Screening techniques decrease time complexity down to $\mathrm{O}(N^{2.5})$ – $\mathrm{O}(N^3)$
- Memory requirements are from $\mathrm{O}(N^2)$ for “direct” HF to $\mathrm{O}(N^4)$ for disk-based methods

MPI parallelization algorithm

- Four nested loops over shells
- “Triangular” loop structure
- Schwartz screening to skip small ERI values (default: less than 10^{-10})
- Original MPI-only implementation:
  - MPI dynamic job load balance over top two shell loops
  - Fock matrix reduction across all MPI ranks at the end of HF iteration
  - All data structures (i.e. Fock and density matrices) are replicated
- Drawbacks of GAMESS implementation:
  - High memory footprint
  - Poor scalability of HF on a large number of MPI ranks because of 2-index load balancer

```
for $i = 1, NShells$ do
    for $j = 1, i$ do
        call ddi_dlbnext($ij$)    ▹ MPI DLB: check I and J indices
        for $k = 1, i$ do
            $k{=}{=}i \;?\; l_{max} \leftarrow k : l_{max} \leftarrow j$
            for $l = 1, l_{max}$ do
▹ Schwartz screening:
                $screened \leftarrow$ schwartz($i, j, k, l$)
                if not $screened$ then
                    call eri($i, j, k, l, X_{ijkl}$)    ▹ Calculate $(i, j|k, l)$
▹ Update process-local 2e-Fock matrix:
                    $Fock_{ij,kl,ik,jl,il,jk}$ +=
                        $X_{ijkl} \cdot D_{kl,ij,jl,ik,jk,il}$
                end if
            end for
        end for
    end for
end for
▹ 2e-Fock matrix reduction over MPI ranks:
call ddi_gsumf($Fock$)
```

Shared Fock matrix algorithm

- Each $(i,j|k,l)$ combination of indices requires up to six updates:

$$F_{kl} \leftarrow (i,j|k,l)\cdot D_{ij}$$

Thread safe update

$$F_{ij} \leftarrow (i,j|k,l)\cdot D_{kl}$$
$$F_{ik} \leftarrow (i,j|k,l)\cdot D_{jl}$$
$$F_{il} \leftarrow (i,j|k,l)\cdot D_{jk}$$

Unsafe update of F_{ix} elements

$$F_{jl} \leftarrow (i,j|k,l)\cdot D_{ik}$$
$$F_{jk} \leftarrow (i,j|k,l)\cdot D_{il}$$

Unsafe update of F_{jx} elements

- F_{ij}, F_{ik}, F_{il} updates are accumulated in F_i vector
- F_{jl}, F_{jk} updates are accumulated in F_j vector
- F_i and F_j vectors are summed and update F matrix when the corresponding index changes

```
loop
    !$omp master
    call ddi_dlbnext($ij$) ▹ MPI DLB: get new combined IJ index
    !$omp end master
    !$omp barrier
    $i, j \leftarrow ij$                                ▹ Deduce I and J indices
    $kl_{max} \leftarrow i, j$                          ▹ Deduce KL-loop limit
    $screened \leftarrow$ schwartz($i, j, i, j$)         ▹ I and J prescreening
    if not $screened$ then
        if $i \neq i_{old}$ then                        ▹ If $i$ was changed flush $F_I$
            $Fock(:, i) += \sum F_I(:, 1{:}nthreads)$
            !$omp barrier
        end if
        !$omp do schedule(dynamic,1)
        for $kl = 1, kl_{max}$ do
            $k, l \leftarrow kl$                        ▹ Deduce K and L indices
            $screened \leftarrow$ schwartz($i, j, k, l$) ▹ Schwartz screening
            if not $screened$ then
                call eri($i, j, k, l, X_{ijkl}$)        ▹ Calculate $(i, j|k, l)$
    ▹ Update private partial Fock matrices:
                $F_I(:, ithread)_{j,k,l} += X_{ijkl} \cdot D_{kl,jl,jk}$
                $F_J(:, ithread)_{k,l} += X_{ijkl} \cdot D_{il,ik}$
    ▹ Update shared Fock matrix:
                $Fock(k, l) += X_{ijkl} \cdot D(i, j)$
            end if
        end for
        !$omp end do
        $Fock(:, j) += \sum F_J(:, 1{:}nthreads)$       ▹ Flush $F_J$
        !$omp barrier
        $i_{old} \leftarrow i$
    end if
end loop
▹ Flush remainder $F_i$ contribution to $Fock$:
$Fock(:, i) += \sum F_I(:, 1{:}nthreads)$
```

Shared Fock matrix algorithm

- Each $(i,j|k,l)$ combination of indices requires up to six updates:

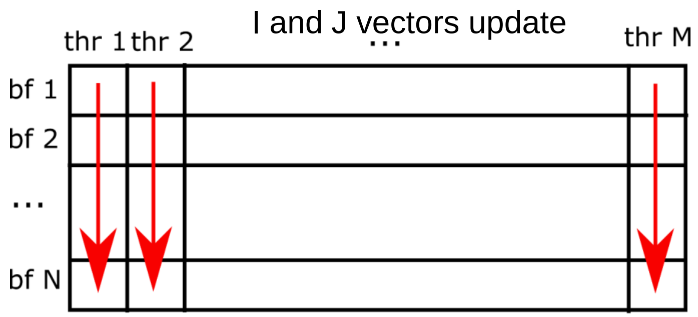
$$F_{kl} \leftarrow (i,j|k,l) \cdot D_{ij}$$
Thread safe update

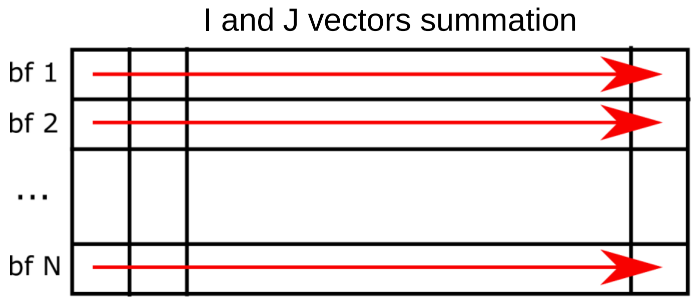
$$F_{ij} \leftarrow (i,j|k,l) \cdot D_{kl}$$
$$F_{ik} \leftarrow (i,j|k,l) \cdot D_{jl}$$
$$F_{il} \leftarrow (i,j|k,l) \cdot D_{jk}$$
Unsafe update of F_{ix} elements

$$F_{jl} \leftarrow (i,j|k,l) \cdot D_{ik}$$
$$F_{jk} \leftarrow (i,j|k,l) \cdot D_{il}$$
Unsafe update of F_{jx} elements

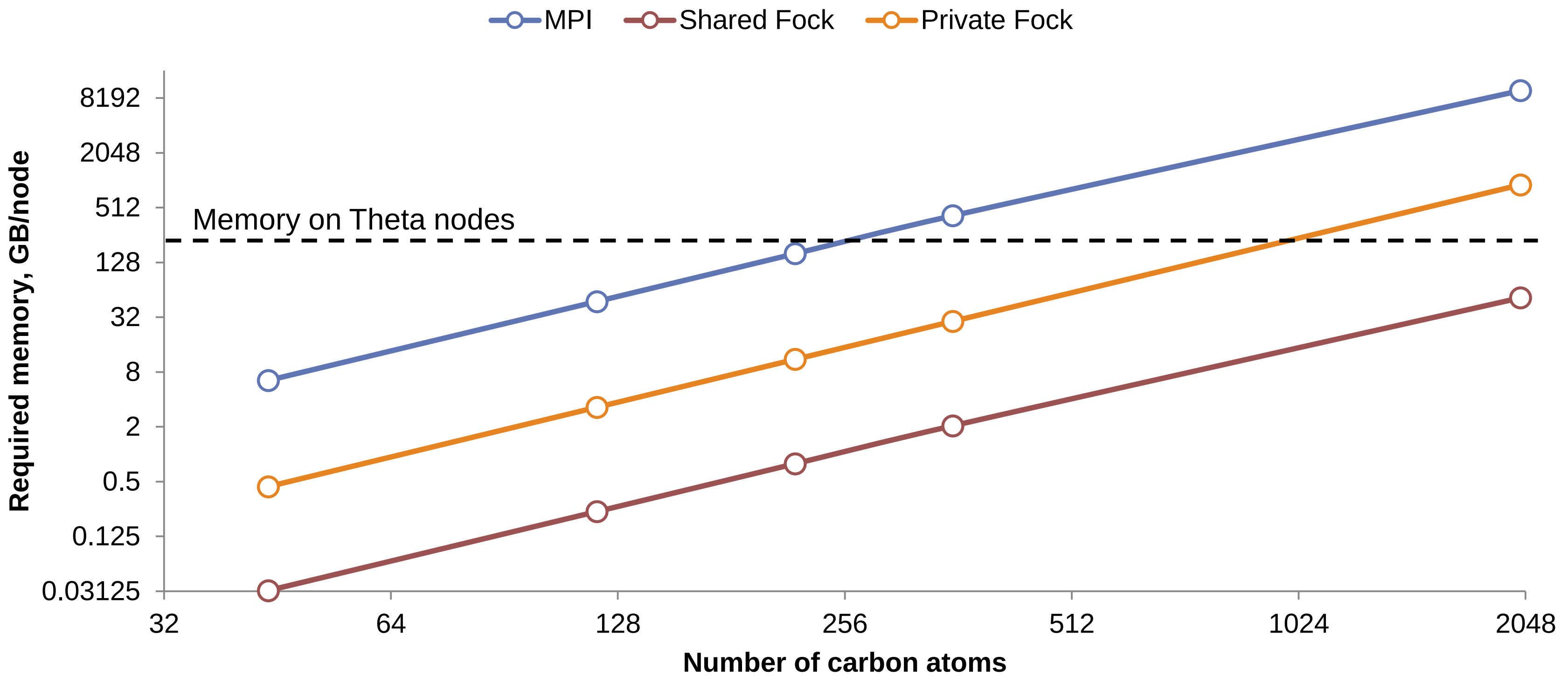
- F_{ij}, F_{ik}, F_{il} updates are accumulated in F_i vector
- F_{jl}, F_{jk} updates are accumulated in F_j vector
- F_i and F_j vectors are summed and update F matrix when the corresponding index changes

Use of I and J vectors:





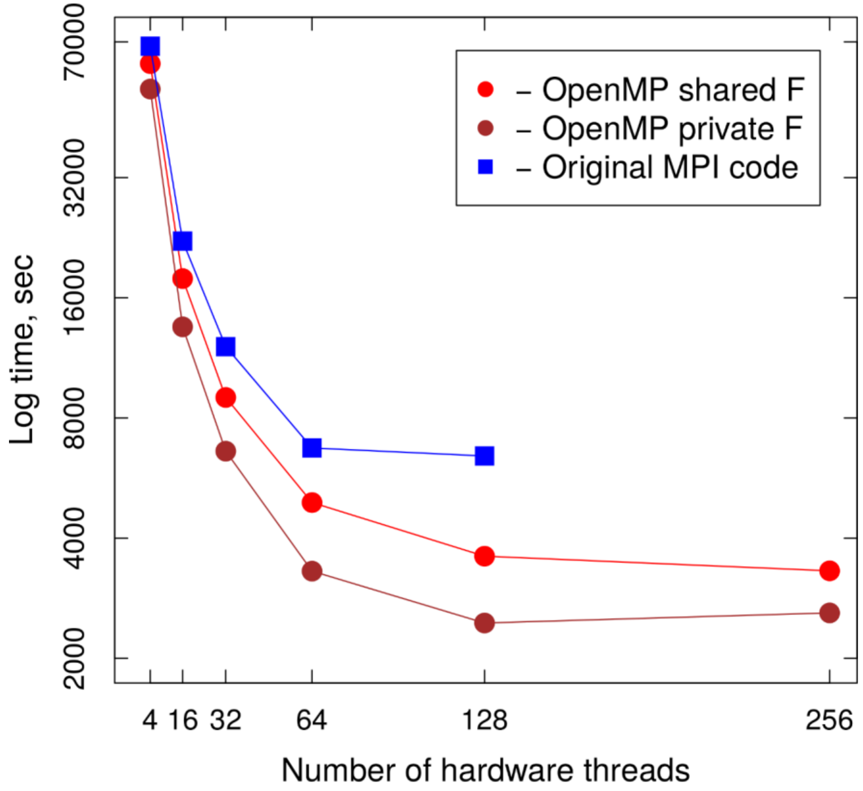
Memory requirements



Setup: MPI-only HF algorithm – 256 MPI ranks/processor; Private and Shared Fock algorithms – 1 MPI ranks/processor, 256 threads/MPI rank

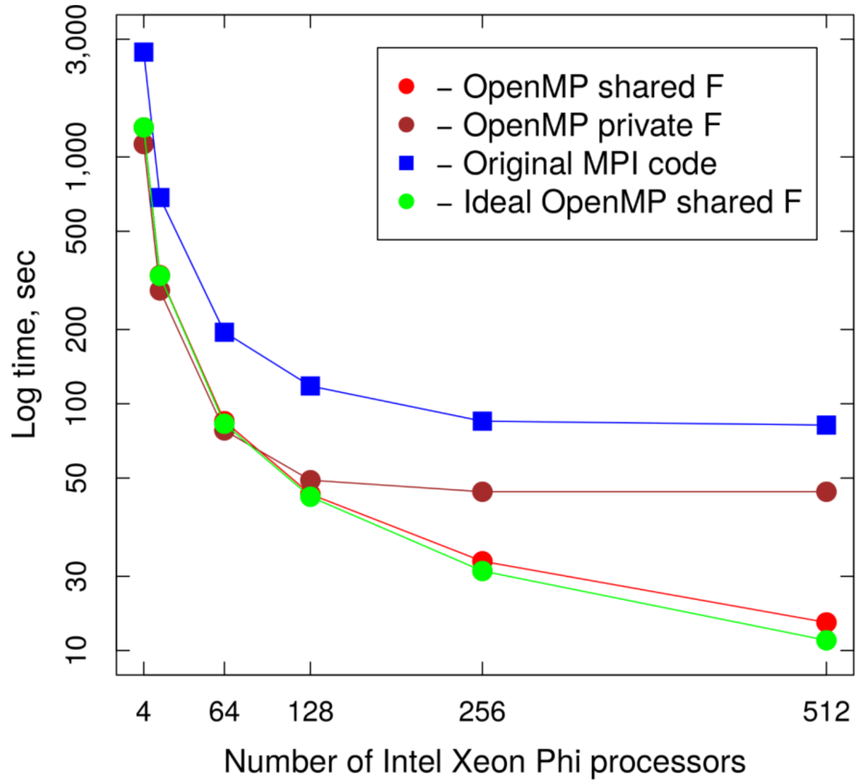
MPI/OpenMP scaling on single Xeon Phi node

- Chemical system: 1.0 nm graphene bilayer (C_{120},1800 basis functions)
- Single Intel Xeon Phi processor
- MPI/OpenMP versions of HF method run faster than the original MPI-only code by ~2.5 times
- MPI-only code can run maximum on 128 hardware threads because of large memory requirements per MPI rank
- MPI/OpenMP hybrid versions of HF method can utilize all 256 hardware threads



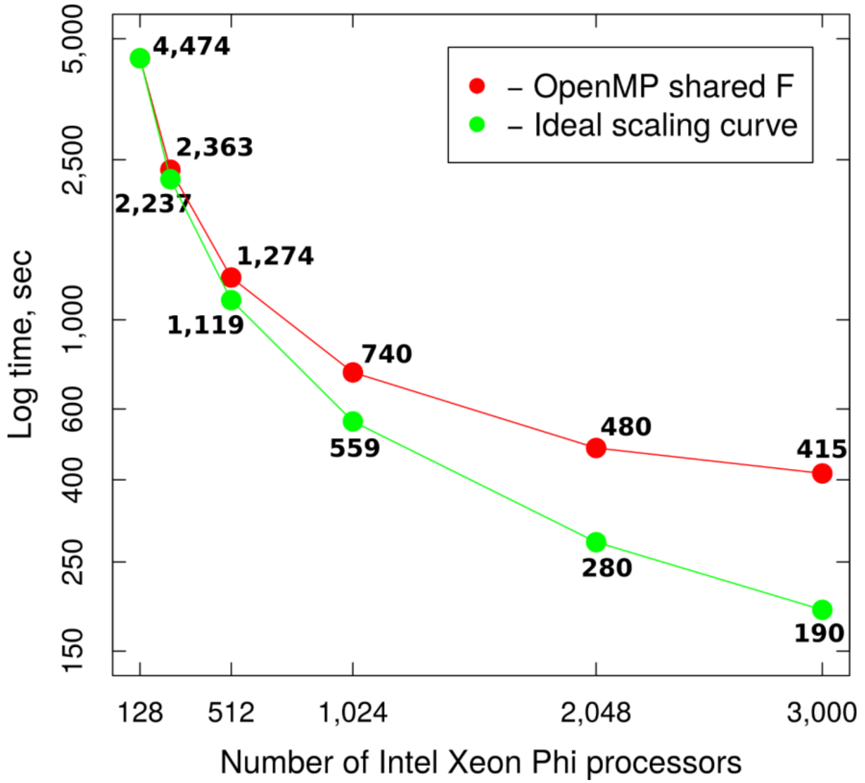
Scaling on 512 Xeon Phi processors

- Chemical system: 2.0 nm graphene bilayer (C_{356}, 5340 basis functions)
- 512 KNL processors on Theta supercomputer installed in ALCF
- Quadrant-cache KNL modes
- OpenMP shared Fock algorithm scales close to ideal
- OpenMP shared Fock algorithm runs ~6 times faster than MPI-only code
- MPI-only original GAMESS algorithm does not scale beyond 256 processors



Scaling on 3,000 Xeon Phi processors

- Chemical system: 5.0 nm graphene bilayer (2,016 atoms; 30,240 basis functions)
- Scaling is demonstrated for MPI/OpenMP shared Fock code
- Memory requirements for MPI-only code vastly larger compared to MPI/OpenMP shared Fock code
- Code scales on 3,000 Theta KNL processors
- Quadrant-cache KNL modes
- 4 MPI ranks per node, 64 threads per rank



Conclusions

- Developed new OpenMP/HF algorithms
- Sped up code up to 6 times
- Memory footprint is reduced by up to ~200 times
- Scaled code on 3,000 KNL processors (192,000 cores)
- OpenMP/HF algorithms are implemented in released version of GAMESS:
  - Hybrid MPI/OpenMP energy code for RHF, UHF, ROHF, and Coulomb part of DFT exchange-correlation energy
  - Hybrid MPI/OpenMP gradient code for RHF, UHF, ROHF, and Coulomb part of DFT exchange-correlation energy
- Code is available on GAMESS website: http://www.msg.ameslab.gov/gamess/download.html

Acknowledgments

- This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357
- We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory
- Vladimir Mironov thanks Intel® Parallel Compute Center program for funding
- The authors would like to thank the RSC Technologies staff for discussion and help related to the paper and for providing Intel Xeon Phi development platform
- The authors thank Sameer Shende for help with TAU profiling

TAU Profiles

Profile collection details:

- Theta 512 KNL processors
- All measurements were performed on MPI rank 0/OpenMP thread 0
- Original MPI code:
  128 MPI ranks per node, total 65,536 ranks
- Both OpenMP codes:
  4 MPI ranks per node and 64 threads per rank, total 2,048 ranks
- All times are exclusive and measured in seconds

Operation	Time, s		
	MPI, original	MPI/OMP priv. Fock	MPI/OMP shr. Fock
MPI_Broadact	234	70	62
MPI_AllReduce	264	109	111
MPI_Barrier	25	40	7
OpenMP region	N/A	15	50